

საქართველოს ტექნიკური უნივერსიტეტი

რომან სამხარაძე, ლია გაჩეჩილაძე

დაპროგრამება C++ ენაზე



რეგისტრირებულია სტუ-ის
სარედაქციო-საგამომცემლო საბჭოს
მიერ. 01.01.2018, ოქმი №

თბილისი
2018

უაკ: 004.43

სახელმძღვანელოში გადმოცემულია Microsoft Visual Studio გარემოში C++ პროგრამების შემუშავების საკითხები. დაწვრილებითაა განხილული C++ ენის საფუძვლები, მონაცემთა ტიპები, მმართველი ოპერატორები, მასივები, სტრიქონები, ფუნქციები, ფაილებთან მუშაობის საკითხები, სტრუქტურირებული ტიპები, ჩამოთვლები, სტრუქტურები და კლასები. წიგნში უხვადაა მაგალითები და ამოცანები ამოხსნებით.

განკუთვნილია ინფორმატიკისა და მართვის სისტემების ფაკულტეტის ბაკალავრებისა და მაგისტრებისთვის, აგრეთვე, დაპროგრამების შესწავლის ნებისმიერი მსურველისთვის.

რეცენზენტი ტექნიკის მეცნიერებათა დოქტორი, პროფესორი გ. სურგულაძე
ტექნიკის მეცნიერებათა კანდიდატი, პროფესორი მ. კიკნაძე

პროფ. გ. სურგულაძის რედაქციით

რედკოლეგია:

ა. ფრანგიშვილი (თავმჯდომარე), მ. ახობაძე, ზ. ბაიაშვილი, ზ. ბოსიკაშვილი,
ზ. გასიტაშვილი, გ. გოგიჩაიშვილი, მ. თევდორაძე, ე. თურქია, ლ. იმნაიშვილი,
თ. კაიშაური, რ. კაკუბავა, ჰ. მელაძე, თ. ლომინაძე, ნ. ლომინაძე, თ. ობგაძე,
რ. სამხარაძე, გ. სურგულაძე, გ. ჩაჩანიძე, ა. ცინცაძე, გ. ძიძიგური, ზ. წვერაძე

© © სტუ-ს "IT-კონსალტინგის სამეცნიერო ცენტრი", 2018

ISBN ISBN 978-9941-27-493-0



ყველა უფლება დაცულია. ამ წიგნის ნებისმიერი ნაწილის (ტექსტი, ფოტო, ილუსტრაცია თუ სხვა) გამოყენება არც ერთი ფორმითა და საშუალებით (ელექტრონული თუ მექანიკური), არ შეიძლება გამომცემლის წერილობითი ნებართვის გარეშე.

საავტორო უფლებების დარღვევა კანონით ისჯება.

სარჩევი	
წინასიტყვაობა	6
თავი 1. შესავალი	7
დაპროგრამების ისტორიის მოკლე მიმოხილვა	7
C++ ენის ISO/ANSI სტანდარტი	7
პროგრამის შემუშავების ინტეგრირებული გარემო	8
თავი 2. C++ ენის საფუძვლები	9
პირველი პროგრამა	9
კომენტარები	14
C++ ენის საბაზო ტიპები	15
მთელრიცხვა ტიპები	15
მთელრიცხვა ტიპების მოდიფიკატორები	15
მოდრაფერტილიანი ტიპები	17
სიმბოლური ტიპები	17
bool ტიპი	18
string ტიპი	19
ლიტერალები	20
მონაცემთა ტიპისთვის სინონიმის განსაზღვრა	21
მუდმივა	22
ცვლადი და მისი ინიციალიზება	22
ცვლადის ინიციალიზება	23
ცვლადების დინამიკური ინიციალიზება	23
ოპერატორები	24
მინიჭების ოპერატორი	24
მინიჭების შედგენილი ოპერატორი	24
ართმეტიკის ოპერატორები	25
ოპერატორების პრიორიტეტი	26
ინკრემენტისა და დეკრემენტის ოპერატორები	27
შედარებისა და ლოგიკის ოპერატორები	29
გამოსახულება. მათემატიკის ფუნქციები	31
ტიპის გარდაქმნა	33
დაყვანა მინიჭების ოპერატორებში	35
ცხადი დაყვანა	35
ტიპების დაყვანის ძველი სტილი	37
შენახვის დრო და ხილვადობის უბანი	37
ავტომატური ცვლადი	38
სტატიკური ცვლადი	39
გლობალური ცვლადი	39
თავი 3. მმართველი ოპერატორები	41
ამორჩევის ოპერატორები	41
if ოპერატორი	41
ჩადგმული if ოპერატორი	41
switch ოპერატორი	42
ჩადგმული switch ოპერატორი	45
იტერაციის ოპერატორები	46
for ოპერატორი	46
while ოპერატორი	50
do-while ოპერატორი	51

გადასვლის ოპერატორები	52
break ოპერატორი	52
continue ოპერატორი	52
goto ოპერატორი	53
ტერნარული ოპერატორი.....	54
მაგალითები	54
თავი 4. მასივები და ბიტობრივი ოპერაციები.....	57
ISO/ANSI მასივი.....	57
ერთგანზომილებიანი მასივი	57
ერთგანზომილებიანი მასივის ინიციალიზება.....	58
ორგანზომილებიანი მასივი	59
ორგანზომილებიანი მასივის ინიციალიზება.....	59
მიმთითებელი	61
მიმთითებლის ინიციალიზება.....	62
& ოპერატორი	62
მიმთითებელი და მასივი	64
მიმთითებლის არითმეტიკა	65
მიმთითებელი და მრავალგანზომილებიანი მასივი.....	68
მიმართვა	68
sizeof ოპერაცია.....	70
size() ფუნქცია	71
მეხსიერების დინამიკური გამოყოფა.....	71
მეხსიერების გროვა	72
new და delete ოპერატორები.....	72
მეხსიერების დინამიკური განაწილება მასივებისთვის.....	73
შემთხვევითი რიცხვების გენერატორი	73
ბიტობრივი ოპერატორები.....	75
&, , ^ და ~ ბიტობრივი ოპერატორები.....	75
ძვრის ოპერატორები	79
asm საკვანძო სიტყვა.....	80
თავი 5. სტრიქონები	82
სტრიქონი	82
სტრიქონებთან სამუშაო ფუნქციები.....	83
სტრიქონების მასივი	88
თავი 6. ინფორმაციის შეტანა-გამოტანა	89
შეტანა-გამოტანის ნაკადების კლასები.....	89
ფაილში სხვადასხვა ტიპის მონაცემების ჩაწერა და წაკითხვა	89
ფაილში სხვადასხვა ტიპის მასივების ჩაწერა და წაკითხვა	93
ფაილისთვის მონაცემების დამატება	98
თავი 7. ფუნქციები.....	100
ფუნქცია.....	100
პარამეტრის გამოყენება	101
ფუნქციიდან მართვის დაბრუნება.....	102
ფუნქციისთვის მასივების გადაცემა	105
პარამეტრების გადაცემა მიმთითებლებისა და მიმართვების გამოყენებით.....	108
ფუნქციის გადატვირთვა	113
გადატვირთვა და არაცალსახობა	115
ნაგულისხმევი არგუმენტი	117

თავი 8. კლასები, ობიექტები და მეთოდები	120
კლასის გამოცხადება.....	120
ობიექტებზე მიმთითებელი	125
ობიექტების მინიჭება.....	126
მეთოდი	127
ფუნქციიდან მართვის დაბრუნება.....	129
პარამეტრის გამოყენება	130
კონსტრუქტორი.....	132
კონსტრუქტორის გადატვირთვა	136
კონსტრუქტორი ნაგულისხმევი პარამეტრებით.....	138
მრავალფაილიანი პროექტის შექმნა	138
თავი 9. ფუნქციები უფრო დაწვრილებით.	147
ჩასადგმელი ფუნქცია.....	147
რეკურსია.....	150
ფუნქციის შაბლონი.....	151
კლასის შაბლონი	154
თავი 10. სტრუქტურები.....	158
სტრუქტურა	158
ჩამოთვლა.....	159
დანართი 1. სავარჯიშოები თავების მიხედვით.....	161
დანართი 2. სავარჯიშოების ამოხსნები	186
ლიტერატურა.....	246

წინასიტყვაობა

Microsoft Visual Studio წარმოადგენს დაპროგრამების თანამედროვე გარემოს. მისი საშუალებით შესაძლებელია სხვადასხვა ტიპის პროგრამა-დანართების შემუშავება, როგორცაა Windows დანართები, ქსელური პროგრამები, მონაცემთა ბაზებთან სამუშაო პროგრამები და ა.შ.

წიგნის მიზანია მკითხველს შეასწავლოს დაპროგრამების საფუძვლები C++ ენის ბაზაზე და განკუთვნილია დამწყები პროგრამისტებისთვის.

სახელმძღვანელოში გადმოცემულია C++ ენის ISO/ANSI სტანდარტი.

აუცილებელი პროგრამული უზრუნველყოფაა - Windows 7/10, Visual Studio 2010/2012/2013/2015/2017.

წიგნი C++ ენაზე დაპროგრამების საკითხების ქართულ ენაზე გადმოცემის ერთ-ერთი პირველი მცდელობაა. ამიტომ, ის არ არის დაზღვეული ხარვეზებისგან. ავტორები მაღლიერებით მიიღებენ წიგნში გადმოცემული მასალის დახვეწისა და სრულყოფის მიზნით გამოთქმულ შენიშვნებსა და მოსაზრებებს. ისინი შეგიძლიათ მოგვაწოდოთ მისამართებზე:

rsamkharadze@mail.ru , samkharadze.roman@gmail.com .

საქართველოს ტექნიკური უნივერსიტეტის
ინფორმატიკისა და მართვის სისტემების ფაკულტეტის
კომპიუტერული ინჟინერიის დეპარტამენტის
სრული პროფესორი, ტექნიკის მეცნიერებათა დოქტორი

რომან სამხარაძე

თავი 1. შესავალი

დაპროგრამების ისტორიის მოკლე მიმოხილვა

დაპროგრამების ენები გამოიყენება ისეთი მრავალფეროვანი ამოცანების გადასაწყვეტად, როგორცაა მონაცემთა საინფორმაციო სისტემების შემუშავება და მართვა, რთული მათემატიკური და ეკონომიკური ამოცანების გადაწყვეტა, მედიცინა, განათლება და ა.შ. C++ ენა, რომელიც Microsoft კომპანიამ შეიმუშავა, მთლიანად პასუხობს დაპროგრამების თანამედროვე სტანდარტებს და განკუთვნილია Windows გარემოში მომუშავე თანამედროვე კომპიუტერული სისტემების შემუშავებისთვის.

დაპროგრამების ენებს შორის არსებობს კავშირი. ყოველი ახალი ენა ადრე შექმნილი ენებისგან მემკვიდრეობით იღებს გარკვეულ თვისებებს. კერძოდ, C++ ენამ ბევრი სასარგებლო თვისება C ენისაგან მემკვიდრეობით მიიღო.

C ენა შეიქმნა ნიუ-ჯერსის შტატის ქალაქ მიურეი-ჰილის Bell Laboratories კომპანიის სისტემური პროგრამისტის დენის რიჩის მიერ 1972 წელს. თითქმის მთლიანად ამ ენაზე დაიწერა Unix, Windows და სხვა ოპერაციული სისტემების ბირთვი. შემდგომში, პროგრამების ზომისა და სირთულის ზრდამ საკმაოდ გააძნელა მათთან მუშაობა. გამოსავალი იყო სტრუქტურულ დაპროგრამებაზე გადასვლა. სწორედ 1980 წლებიდან C ენა გახდა ყველაზე ხშირად გამოყენებადი სტრუქტურული დაპროგრამების ენა.

დაპროგრამების განვითარებასთან ერთად კვლავ დადგა დიდი ზომის პროგრამებთან მუშაობის პრობლემა. აუცილებელი გახდა ახალი მიდგომების შემუშავება. ერთ-ერთი მათგანია ობიექტზე ორიენტირებული დაპროგრამება (ოოდ). ის იძლევა დიდი ზომის პროგრამებთან ეფექტური მუშაობის შესაძლებლობას. შესაბამისად, 1974 წელს იმავე Bell Laboratories კომპანიაში ბიარნ სტრაუსტრუპმა (Bjarne Stroustrup) შექმნა C ენის ობიექტზე ორიენტირებული ვერსია - "C კლასებით". მას 1983 წლიდან C++ დაერქვა. C++ მთლიანად მოიცავს C ენას და შეიცავს ობიექტზე ორიენტირებული დაპროგრამების შესაძლებლობებს. 1990 წლიდან იწყება C++ ენის მასობრივი გამოყენება და ის ხდება ყველაზე პოპულარული დაპროგრამების ენებს შორის.

C ენამ პირველი ცვლილება 1985 წელს განიცადა, მეორე კი - 1990 წელს. 1994 წელს მესამე ცვლილების შედეგად მიღებულ იქნა C++ ენის ერთიანი საერთაშორისო სტანდარტი. შემდეგ ის მნიშვნელოვნად გაფართოვდა. მას დაემატა ალექსანდრე სტეპანოვის მიერ შემუშავებული სტანდარტული შაბლონების ბიბლიოთეკა (Standart Template Library, STL).

C++ ენის ISO/ANSI სტანდარტი

წიგნში განხილულია C++ ენის ISO/ANSI სტანდარტი (International Standards Organization/American National Standards Institute, სტანდარტების საერთაშორისო ორგანიზაცია/სტანდარტების ამერიკის ეროვნული ინსტიტუტი). ეს არის C++ ენის თავდაპირველი სტანდარტი. ის აღწერს C++ ენის იმ ვერსიას, რომელიც 1998 წლიდან არსებობს და უზრუნველყოფილია ოპერაციული სისტემებისა და კომპიუტერული პლატფორმების უმეტესობით. ამ სტანდარტით განსაზღვრულ C++ ენაზე დაწერილ პროგრამას **C++ ენის მშობლიური პროგრამა** ეწოდება. ISO/ANSI C++ ენაზე დაწერილი პროგრამის გადატანა ერთი პლატფორმიდან მეორეზე შედარებით ადვილია. გადატანის სირთულეს განსაზღვრავს პროგრამაში გამოყენებული ბიბლიოთეკური ფუნქციები.

ISO/ANSI C++ სტანდარტი განკუთვნილია მაღალმწარმოებლური პროგრამების (application, приложение, პროგრამა-დანართი) შესამუშავებლად, რომლებიც სრულდება როგორც მშობლიური პროგრამები (არამართვადი C++).

პროგრამის შემუშავების ინტეგრირებული გარემო

პროგრამის შემუშავების ინტეგრირებული გარემო (Integrated Development Environment, IDE), რომელიც თან ახლავს Visual C++ სისტემას, არის C++ პროგრამების შექმნის, კომპილირების, აწყობისა და ტესტირებისთვის განკუთვნილი გარსი. მისი კომპონენტებია: რედაქტორი, კომპილატორი, ამწყობი და ბიბლიოთეკები.

რედაქტორი (editor) არის ინტეგრირებული გარემო, რომელშიც შეგვიძლია C++ ენის საწყისი კოდის შექმნა და რედაქტირება. რედაქტორი ავტომატურად აღიქვამს C++ ენის საკვანძო სიტყვებს, შეაფერადებს მათ და აქვს ტექსტური რედაქტორისთვის დამახასიათებელი ფუნქციები.

კომპილატორი (compiler) საწყის კოდს ობიექტურ კოდად გარდაქმნის და გვაუწყებს კომპილირებისას აღმოჩენილი შეცდომების შესახებ. გამოსასვლელ ობიექტურ კოდს კომპილატორი *ობიექტურ ფაილში* ათავსებს. ასეთი ფაილის გაფართოებაა .obj .

ამწყობი (builder) ახდენს კომპილატორით გენერირებული სხვადასხვა მოდულის კომბინირებას, უმატებს მათ საჭირო მოდულებს ბიბლიოთეკებიდან და ერთ ფაილში აერთიანებს. ამწყობს შეუძლია, აგრეთვე, შეცდომების აღმოჩენა და მათ შესახებ შეტყობინების გაცემა.

ბიბლიოთეკა (library) არის წინასწარ დაწერილი პროცედურების (ფუნქციების, პროგრამების) კოლექცია, რომლებიც C++ ენის შესაძლებლობებს აფართოებს. ისინი შეგვიძლია ჩავრთოთ ჩვენს პროგრამებში სტანდარტული და ხშირად გამოყენებადი ოპერაციების შესასრულებლად.

თავი 2. C++ ენის საფუძვლები

პირველი პროგრამა

C++ ენაზე შევადგინოთ უმარტივესი პროგრამა, რომელიც ორი მთელი რიცხვის შეკრებას შეასრულებს:

```
{  
//   პროგრამა 2.1  
//   ეს არის ჩვენი პირველი პროგრამა  
int ricxvi1, ricxvi2, jami;  
  
ricxvi1 = 2;  
ricxvi2 = 3;  
jami   = ricxvi1 + ricxvi2;  
}
```

როგორც ვხედავთ C++ ენაზე შედგენილი პროგრამა იწყება გამხსნელი ფიგურული ფრჩხილით "{" და მთავრდება დამხურავი ფიგურული ფრჩხილით "}". პროგრამის მეორე და მესამე სტრიქონებში მოთავსებულია კომენტარები, რომელებიც // სიმბოლოებით იწყება (კომენტარებს მოგვიანებით განვიხილავთ). მომდევნო სტრიქონში ხდება ცვლადების გამოცხადება. ცვლადის სახელი არის მესხიერების იმ უბნის სახელი, რომელიც ამ ცვლადისთვის გამოიყო და რომელშიც ამ ცვლადის მნიშვნელობა იქნება მოთავსებული. int სიტყვა (integer - მთელი) იწყებს ცვლადების გამოცხადებას და მიუთითებს, რომ ისინი მთელი რიცხვებია. მას მოსდევს ცვლადების სახელები ანუ იდენტიფიკატორები, რომლებიც ერთმანეთისგან მძიმეებით გამოიყოფა. წერტილ-მძიმე ამთავრებს ცვლადების გამოცხადებას, აგრეთვე ერთმანეთისგან გამოყოფს ოპერატორებსა და ფუნქციებს (ფუნქცია არის სახელის მქონე მცირე ზომის პროგრამა, რომელიც გარკვეულ მოქმედებებს ასრულებს). ცვლადების სახელები აუცილებლად უნდა იწყებოდეს სიმბოლოთი და უმჯობესია შევარჩიოთ შინაარსიდან გამომდინარე. C++ ენაში ნებისმიერი ცვლადი გამოცხადებული უნდა იყოს მის გამოყენებამდე.

ცვლადების გამოცხადების შემდეგ ricxvi1 ცვლადს ენიჭება მნიშვნელობა - 2. "=" სიმბოლო არის მინიჭების ოპერატორი. ის მის მარჯვნივ მოთავსებულ მნიშვნელობას გადაწერს მის მარცხნივ მოთავსებულ ცვლადში (ricxvi1 ცვლადისთვის გამოყოფილ უბანში ჩაიწერება მნიშვნელობა - 2). იგივე ეხება პროგრამის მომდევნო სტრიქონს. უკანასკნელ სტრიქონში jami ცვლადს ენიჭება ricxvi1 და ricxvi2 ცვლადების მნიშვნელობების ჯამი.

ზოგად შემთხვევაში, მინიჭების ოპერატორის ("=") მარჯვნივ მოთავსებული გამოსახულების ტიპი დაყვანილი უნდა იყოს მინიჭების ოპერატორის მარცხნივ მოთავსებული ცვლადის ტიპზე.

თვალსაჩინოების მიზნით პროგრამის თითოეულ სტრიქონში მოთავსებულია თითო ოპერატორი. თუმცა, ერთ სტრიქონში შეიძლება რამდენიმე ოპერატორის მოთავსება. მთავარია, რომ ისინი ერთმანეთისგან წერტილ-მძიმეებით იყოს გამოყოფილი.

ერთ-ერთი ნაკლი, რომელიც ამ პროგრამას აქვს ის არის, რომ ის ახდენს მხოლოდ 2-სა და 3-ის შეკრებას. გარდა ამისა, პროგრამას შედეგი ეკრანზე არ გამოაქვს. ამ ნაკლის აღმოსაფხვრელად გამოვიყენებთ მონაცემების შეტანისა და გამოტანის ფუნქციებს. კერძოდ, მონაცემების შესატანად გამოვიყენებთ cin ფუნქციას, გამოსატანად კი - cout ფუნქციას. მათი გამოყენებით 2.1 პროგრამა შემდეგ სახეს მიიღებს:

```
{
```

```
// პროგრამა 2.2
// ორი რიცხვის შეკრების პროგრამა
int ricxvi1, ricxvi2, jami;

cin >> ricxvi1 >> ricxvi2;
jami = ricxvi1 + ricxvi2;
cout << "jami = " << jami << endl;
}
```

როგორც პროგრამიდან ჩანს, cin ფუნქციით ხდება ორი მთელი რიცხვის მნიშვნელობის შეტანა. პირველი რიცხვის მნიშვნელობა მიენიჭება ricxvi1 ცვლადს, მეორე რიცხვის მნიშვნელობა კი - ricxvi2 ცვლადს. მომდევნო სტრიქონში jami ცვლადს მიენიჭება ricxvi1 და ricxvi2 ცვლადების ჯამი. cout ფუნქციას ეკრანზე ჯერ გამოაქვს ბრჭყალებში მოთავსებული სტრიქონი, შემდეგ კი ჯამის მნიშვნელობა.

ახლა შევასრულოთ ეს პროგრამა. ამისათვის, ჯერ გავუშვათ Microsoft Visual Studio .NET 2013 სისტემა. გაიხსნება Microsoft Development Environment ფანჯარა (ნახ. 2.1). Start განყოფილებაში ვაჭერთ New Project მიმართვას. გაიხსნება New Project ფანჯარა (ნახ. 2.2). Project Types სიაში მოვნიშნით Visual C++ ელემენტი. Templates ზონაში მოვნიშნით Win32 Console Application ელემენტი. Name ველში უნდა შევიტანოთ პროექტის სახელი, მაგალითად, OriRicxvisShekreba. შემდეგ დავაჭიროთ Browse კლავიშს და მოვნიშნოთ ის კატალოგი, რომელშიც უნდა შეიქმნას OriRicxvisShekreba ქვეკატალოგი (შეგვიძლია წინასწარ შევქმნათ ის კატალოგი, რომელშიც მოვათავსებთ აღნიშნულ ქვეკატალოგს). ვაჭერთ Open კლავიშს, შემდეგ კი - Ok კლავიშს. თუ კატალოგს არ ავირჩევთ, მაშინ OriRicxvisShekreba ქვეკატალოგი შეიქმნება Visual Studio 2013 კატალოგის Projects ქვეკატალოგში. ეკრანზე გამოჩნდება Win32 Application Wizard ფანჯარა (ნახ. 2.3). ვაჭერთ Finish კლავიშს.

გაიხსნება პროგრამის კოდის ზონა (ნახ. 2.4). პროგრამა 2.2 უნდა შევიტანოთ return 0; ოპერატორის წინ (ნახ. 2.5). პროგრამის ბოლოს უნდა შევიტანოთ აგრეთვე, system("pause"); ფუნქცია. ეს ფუნქცია ეკრანზე დააყოვნებს ფანჯარას, რომელშიც შედეგები ჩანს. თუ ამ ფანჯარას (ნახ. 2.4) ყურადღებით დავათვალიერებთ, შევნიშნავთ, რომ Visual C++.NET სისტემა პროგრამის კოდის ნაწილს ავტომატურად ქმნის.

stdafx.h ფაილი ავტომატურად გენერირდება, როცა ვიწყებთ პროექტის შექმნას (AFX, Application Framework eXtensions). ეს ფაილი აღწერს სტანდარტულ სისტემურ და პროექტში ჩართულ ფაილებს.

პროგრამის შეტანის შემდეგ მის შესანახად უნდა დავაჭიროთ ინსტრუმენტების პანელის Save All კლავიშს ან კლავიატურის Ctrl+S კლავიშებს. რადგან კომენტარი ქართული შრიფტით გვაქვს შეტანილი, ამიტომ გაიხსნება ფანჯარა, რომელშიც უნდა ჩავრთოთ Apply to all documents გადამრთველი (ნახ. 2.6). პროგრამის შესასრულებლად გამოიყენება F5 კლავიში. თუ პროგრამა უშეცდომოდ შევიტანეთ, მაშინ ეკრანზე გაიხსნება ფანჯარა (ნახ. 2.7), რომელშიც შეგვაქვს მთელი რიცხვები: 1 და 2. ისინი ერთმანეთისგან ინტერვალით უნდა გამოვყოთ. Enter კლავიშზე დაჭერის შემდეგ დავინახავთ შედეგს. ფანჯრის დასახურად ვაჭერთ ფორმის ზედა მარჯვენა კუთხეში მოთავსებულ კლავიშს ან კლავიატურის Esc კლავიშს.

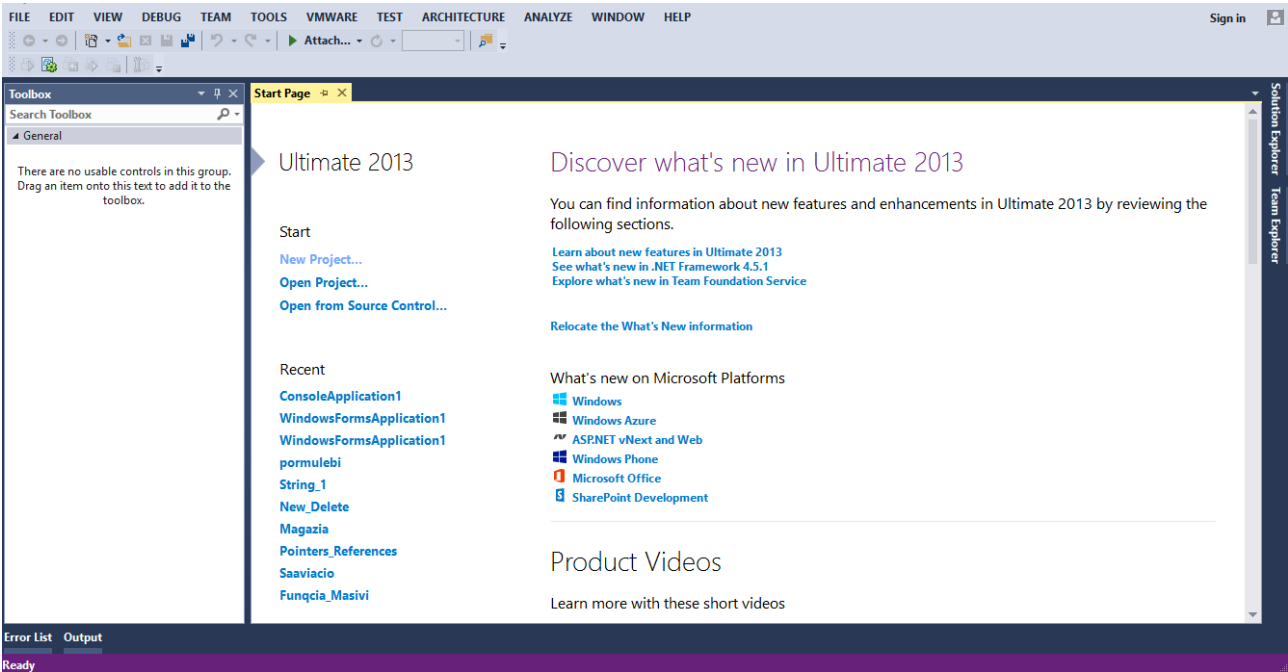
C++ პროგრამის ფაილებს აქვთ .h და .cpp გაფართოება (ტიპი). მაგალითად, იმ ფაილის სახელი, რომელშიც ჩვენი პროგრამის საწყისი კოდი შეგვაქვს, არის OriRicxvisShekreba.cpp .

ბოლოს, შევნიშნოთ, რომ C++ ენაში განსხვავებულია ზედა და ქვედა რეგისტრის სიმბოლოები. მაგალითად, განსხვავებულია Computer და computer სახელები. მაგალითი:

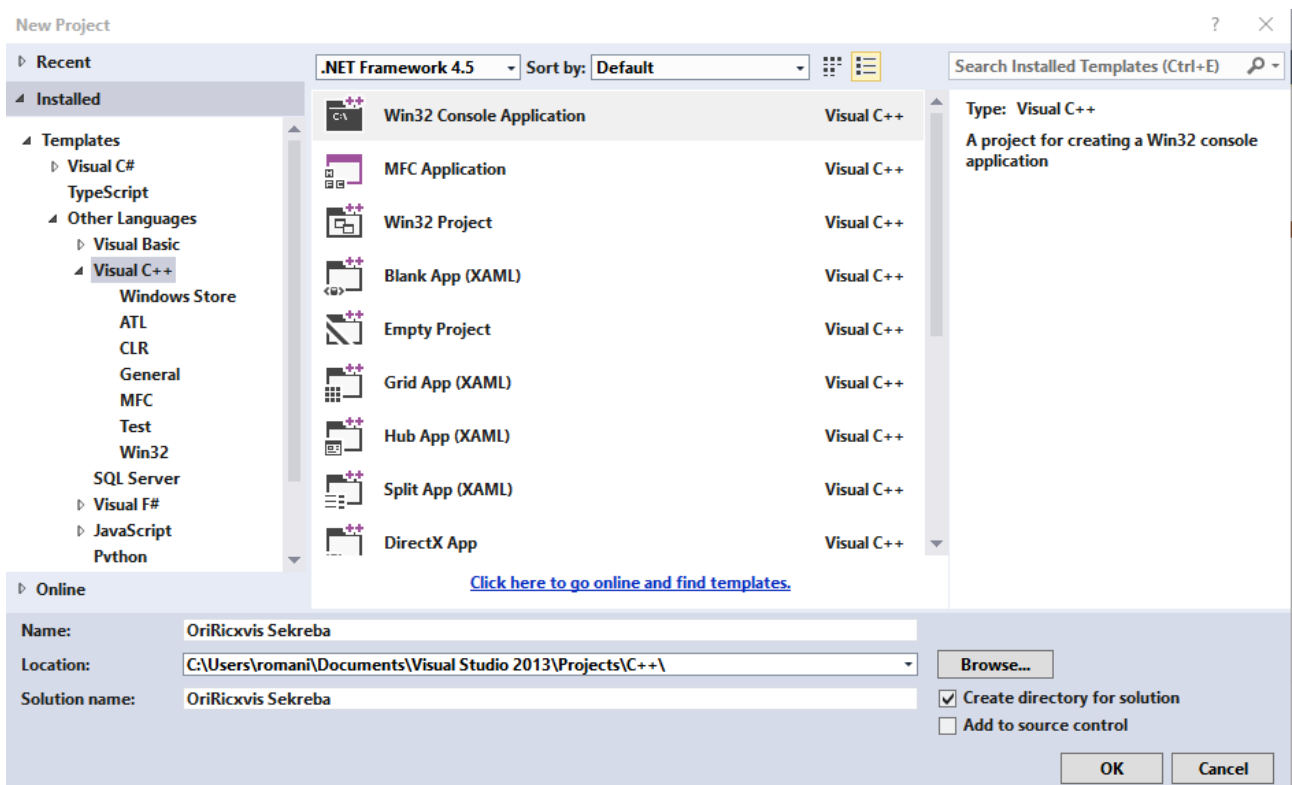
```
{
// C++ ენაში განსხვავებულია ზედა და ქვედა რეგისტრის სიმბოლოები
int ricxvi1;
```

```
Ricxvil = 5; // შეცდომა! Ricxvil ცვლადი არ არის გამოცხადებული
}
```

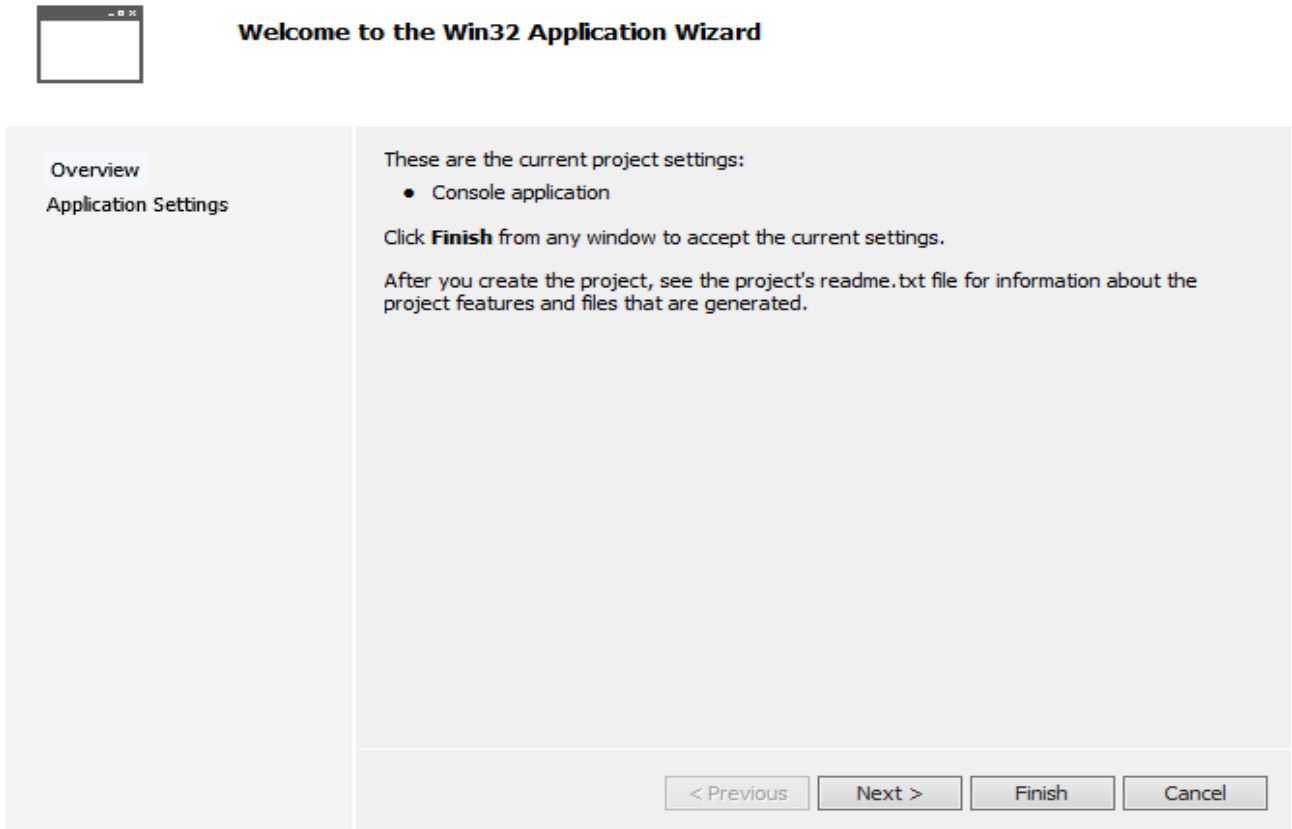
როგორც მაგალითიდან ჩანს, გამოცხადებული იყო ცვლადი, რომლის სახელია ricxvil, ხოლო გამოიყენება სახელი Ricxvi1. C++ ენისთვის ეს ორი სხვადასხვა სახელია.



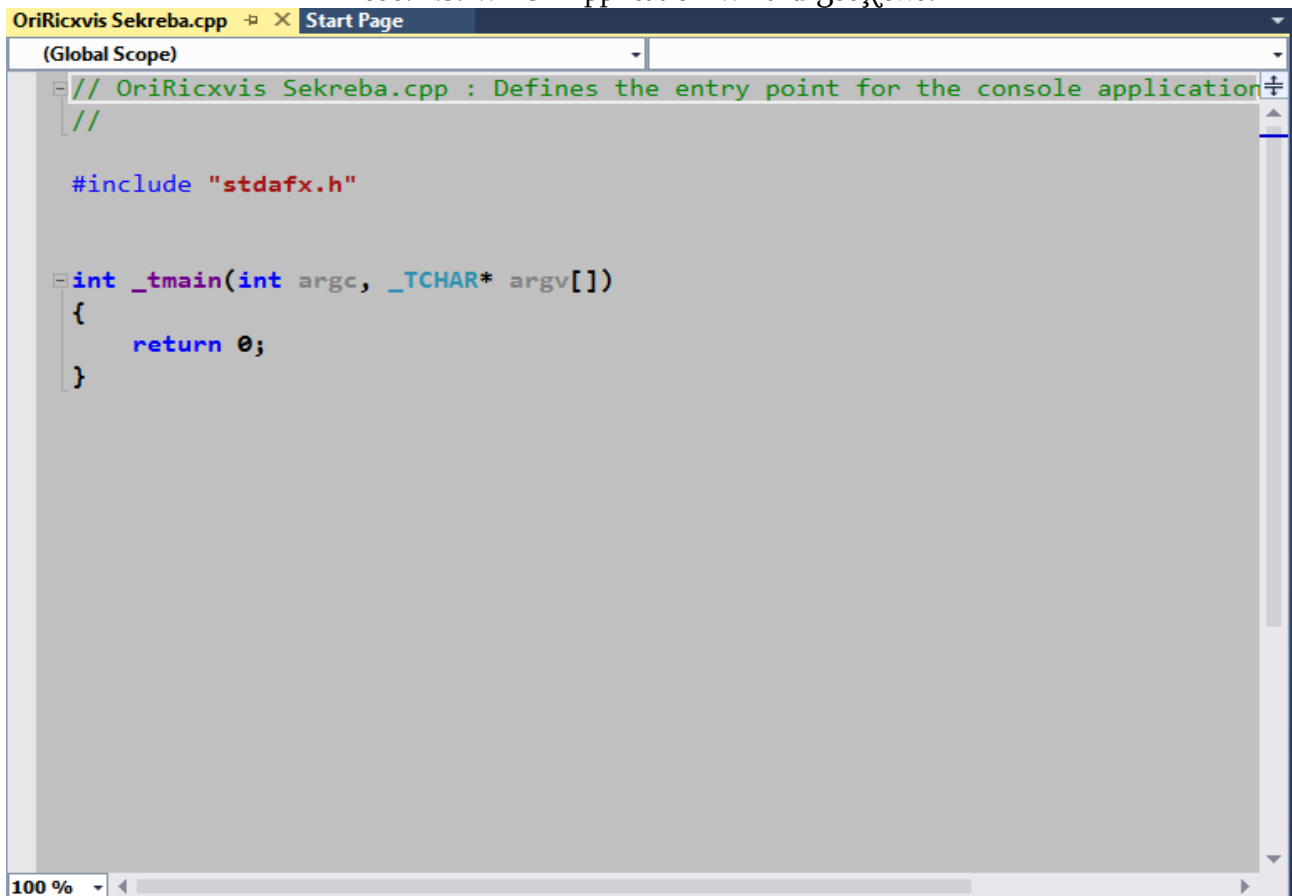
ნახ. 2.1. Start Page ფანჯარა.



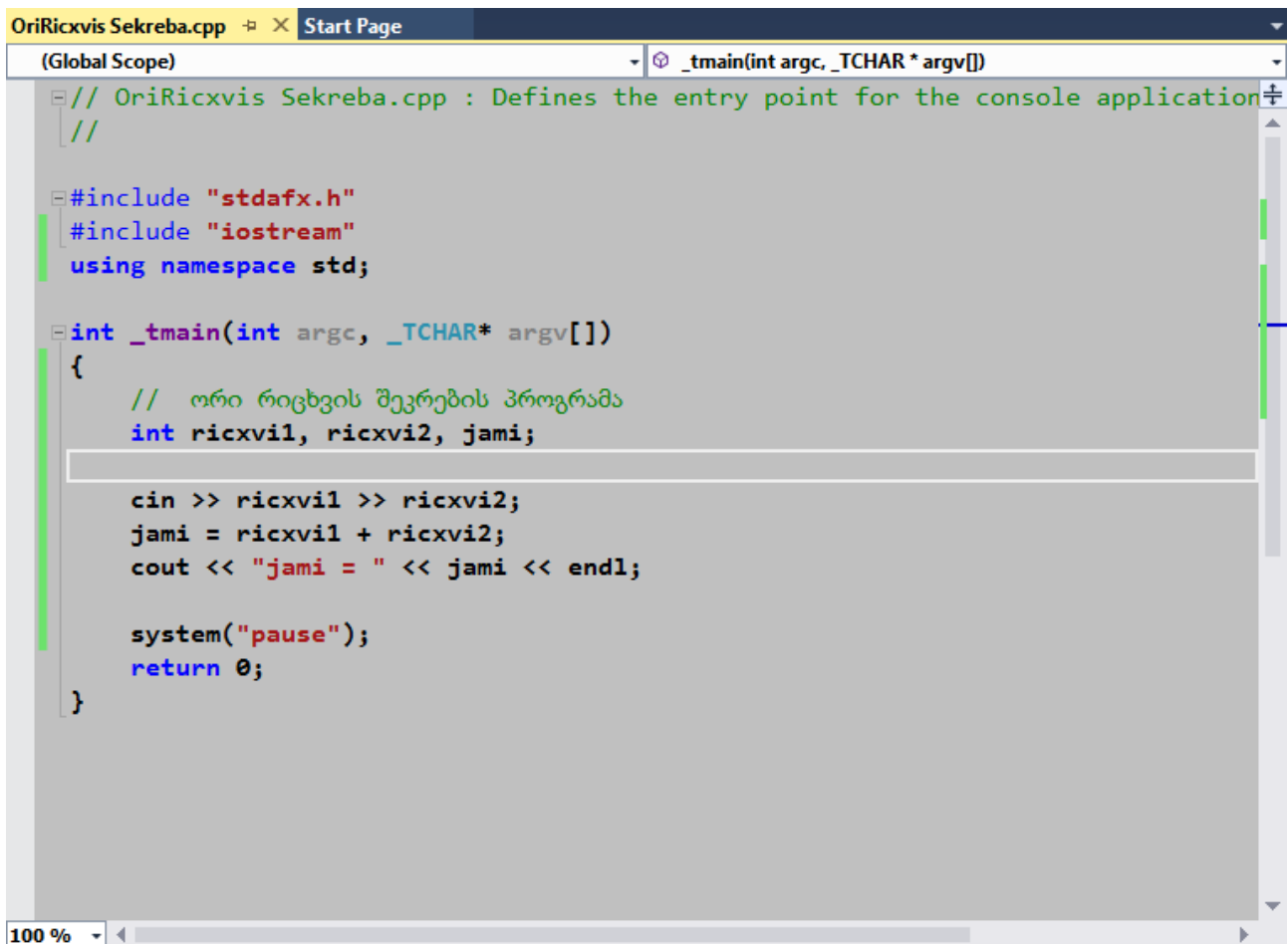
ნახ. 2.2. New Project ფანჯარა.



ნახ. 2.3. Win32 Application Wizard ფანჯარა.



ნახ. 2.4. OriRicxvisShekreba.cpp ვევერდი.



```
// OriRicxvis Sekreba.cpp : Defines the entry point for the console application.
//

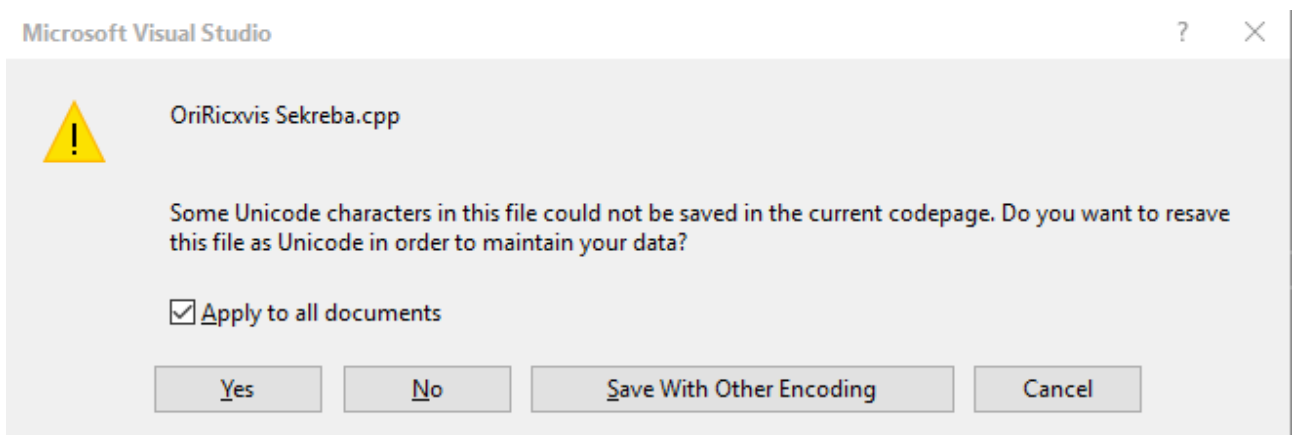
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    // ორი რიცხვის შეკრების პროგრამა
    int ricxvi1, ricxvi2, jami;

    cin >> ricxvi1 >> ricxvi2;
    jami = ricxvi1 + ricxvi2;
    cout << "jami = " << jami << endl;

    system("pause");
    return 0;
}
```

ნახ. 2.5. OriRicxvisShekreba.cpp გვერდი.



ნახ. 2.6. ფაილის შენახვა Unicode ფორმატში.

```

C:\Users\romani\Documents\Visual Studio 2013\Projects\C++\OriRicxvis Sekreba\Debug\OriRicxvis Sekreba.exe
1 2
jami = 3
Press any key to continue . . .

```

ნახ. 2.7. შედეგების ნახვა.

კომენტარები

C++ ენაში არსებობს ერთსტრიქონიანი და მრავალსტრიქონიანი კომენტარები. ერთსტრიქონიანი კომენტარი // სიმბოლოებით იწყება. მრავალსტრიქონიანი კომენტარი /* სიმბოლოებით იწყება და */ სიმბოლოებით მთავრდება. როგორც წესი, კომენტარი შეიცავს პროგრამის სხვადასხვა ნაწილის განმარტებას, ცვლადებისა და ფუნქციების დანიშნულებას და ა.შ. კომენტარების გამოყენება აადვილებს პროგრამის გარჩევას. პროგრამის შესრულებისას ხდება კომენტარების გამოტოვება. ქვემოთ მოცემულია პროგრამა, რომელიც ორივე სახის კომენტარს შეიცავს.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])

```

```

{
/*

```

```

ეს არის ჩვენი
პირველი პროგრამა
*/

```

ეს არის მრავალსტრიქონიანი კომენტარი

```

int ricxvi1, ricxvi2, jami;

```

```

// რიცხვების შეტანა cin ფუნქციის გამოყენებით
cin >> ricxvi1 >> ricxvi2;
jami = ricxvi1 + ricxvi2;
cout << "jami = " << jami << endl;

```

ეს არის ერთსტრიქონიანი კომენტარი

```

system("pause");
return 0;
}

```

C++ ენის საბაზო ტიპები

ინფორმაციის სახესხვაობას, რომელსაც შეძლება ცვლადი შეიცავდეს, *მონაცემის ტიპი* ეწოდება. პროგრამის მონაცემები, ცვლადები და მუდმივები მონაცემთა გარკვეულ ტიპს უნდა ეკუთვნოდეს. ISO/ANSI C++ სტანდარტში განსაზღვრულია *მონაცემების საბაზო ტიპები*. ისინი სამ ნაწილად იყოფა:

- მთელი რიცხვების შემცველი ტიპები;
- ტიპები, რომლებიც არამთელრიცხვა მნიშვნელობებს შეიცავენ;
- void ტიპი, რომელიც მიუთითებს მნიშვნელობების ცარიელ სიმრავლეზე ან ტიპის არარსებობაზე.

ISO/ANSI C++ ტიპები მოცემულია 2.1 ცხრილში.

მთელრიცხვა ტიპები

მთელრიცხვა ცვლადები მხოლოდ მთელ რიცხვებს ინახავენ. C++ ISO/ANSI მთელრიცხვა ტიპები მოცემულია 2.1 ცხრილში. აქედან unsigned მოდიფიკატორიანი ტიპები გამოიყენება უნიშნო მთელი რიცხვების წარმოსადგენად, დანარჩენი ტიპები კი - ნიშნის მქონე რიცხვების წარმოსადგენად. განსხვავება ნიშნის და უნიშნო მთელ რიცხვებს შორის შემდეგშია. ნიშნის რიცხვებში ნიშნის მისათითებლად უფროსი ბიტი (ორობითი თანრიგი) გამოიყენება. თუ ეს ბიტი ნულის ტოლია, მაშინ რიცხვი დადებითია, თუ ერთის ტოლი, მაშინ - უარყოფითი. უნიშნო რიცხვებში ნიშნის წარმოსადგენად უფროსი ბიტი არ გამოიყენება და შედის რიცხვის შემადგენლობაში. ამიტომ, უნიშნო რიცხვების აბსოლუტური მნიშვნელობა ორჯერ აღემატება ნიშნის რიცხვის მნიშვნელობას.

მთელრიცხვა ტიპების მოდიფიკატორები

char, short, int და long ტიპის ცვლადებისთვის იგულისხმება signed ტიპის მოდიფიკატორი. ეს იმას ნიშნავს, რომ ისინი ინახავენ ნიშნის მქონე მნიშვნელობებს ანუ დადებით და უარყოფით მნიშვნელობებს. მაგალითად, როდესაც ვწერთ int ან long ტიპს, იგულისხმება signed int ან signed long ტიპი.

თუ ცვლადი იღებს მხოლოდ დადებით მნიშვნელობებს, მაშინ მისი გამოცხადებისას შეგვიძლია მივუთითოთ unsigned მოდიფიკატორი:

```
unsigned int mteli = 87;
```

თუ ვიყენებთ მხოლოდ signed ან unsigned მოდიფიკატორს, მაშინ შესაბამისად, იგულისხმება signed int ან unsigned int:

```
signed mteli1 = 75; // იგულისხმება signed int mteli1 = 75;
```

```
unsigned mteli2 = 75; // იგულისხმება unsigned int mteli2 = 75;
```

ცხრილი 2.1. C++ ISO/ANSI ტიპები.

ტიპი	აღწერა	ბაიტების რაოდენობა	მნიშვნელობების დიაპაზონი
bool	ლოგიკური მნიშვნელობა	1	false ან true
char	ნიშნის სიმბოლო	1	-128 ÷ 127
signed char	ნიშნის სიმბოლო	1	-128 ÷ 127
unsigned char	უნიშნო სიმბოლო	1	0 ÷ 255
wchar_t	ორბაიტის char	2	0 ÷ 65,535
short	ნიშნის მოკლე მთელი რიცხვი	2	-32,768 ÷ 32,767
unsigned short	უნიშნო მოკლე მთელი რიცხვი	2	0 ÷ 65,535
long	ნიშნის გრძელი მთელი რიცხვი	4	-2,147,483,648 ÷ 2,147,483,647
int	ნიშნის მთელი რიცხვი	4	-2,147,483,648 ÷ 2,147,483,647
unsigned int	უნიშნო მთელი რიცხვი	4	0 ÷ 4,294,967,295
unsigned long	უნიშნო გრძელი მთელი რიცხვი	4	0 ÷ 4,294,967,295
float	ნიშნის წილადი	4	3.4E +/- 38 (7 ციფრი)
double	ნიშნის წილადი	8	1.7E +/- 308 (15 ციფრი)
long double	ნიშნის გრძელი წილადი	იგივეა რაც double	იგივეა რაც double
__int8	8 ბიტის ნიშნის მთელი რიცხვი	1	-128 ÷ 127
unsigned __int8	8 ბიტის უნიშნო მთელი რიცხვი	1	0 ÷ 255
__int16	ნიშნის მოკლე მთელი რიცხვი	2	-32,768 ÷ 32,767
unsigned __int16	უნიშნო მოკლე მთელი რიცხვი	2	0 ÷ 65,535
__int32	ნიშნის მთელი რიცხვი	4	-2,147,483,648 ÷ 2,147,483,647
unsigned __int32	უნიშნო მთელი რიცხვი	4	0 ÷ 4,294,967,295
__int64	ნიშნის გრძელი მთელი რიცხვი	8	-9,223,372,036,854,775,808 ÷ 9,223,372,036,854,775,807
unsigned __int64	უნიშნო გრძელი მთელი რიცხვი	8	0 ÷ 18,446,744,073,709,551,615

მოდრავწერტილიანი ტიპები

მოდრავწერტილიანი ცვლადები წილად რიცხვებს ინახავენ. ქვემოთ მოცემულ პროგრამაში ხდება წილად რიცხვებთან მუშაობის დემონსტრირება.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//    პროგრამაში ხდება წილადებთან მუშაობის დემონსტრირება
float  wiladi1, wiladi2, shedegi1;
double wiladi3, wiladi4, shedegi2;

wiladi1 = 5.5;
wiladi3 = 7.98;
cin >> wiladi2 >> wiladi4;
shedegi1 = wiladi1 + wiladi2;
shedegi2 = wiladi3 + wiladi4;
cout << "shedegi1 = " << shedegi1 << "  shedegi2 = " << shedegi2 << endl;

system("pause");
return 0;
}
```

წილადი ტიპის მუდმივების განსაზღვრისას აუცილებლად უნდა მივუთითოთ წერტილი ან ექსპონენტი, ან ორივე ერთად. წინააღმდეგ შემთხვევაში, მთელ რიცხვს მივიღებთ:

```
const double d1= 9.43;           //    d1 = 9.43
double d2 = 5.6E-3;             //    d2 = 0,0056
double d3 = 6E3;                //    d3 = 6000
```

სიმბოლური ტიპები

სიმბოლური ცვლადი ერთ სიმბოლოს ინახავს. მას char ტიპი აქვს. char ტიპის ცვლადი ერთ ბაიტს იკავებს. მაგალითი.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//    პროგრამაში ხდება სიმბოლოებთან მუშაობის დემონსტრირება
char simbolo1 = 'w';
char simbolo2;
cin >> simbolo2;

cout << "simbolo1 = " << simbolo1 << "  simbolo2 = " << simbolo2 << endl;

system("pause");
}
```

```
return 0;
}
```

სიმბოლოურ ცვლადებს შეგვიძლია, აგრეთვე მივანიჭოთ მმართველი სიმბოლოები. **მმართველია სიმბოლო**, რომელიც გამოიყენება გარკვეული მოქმედებების შესასრულებლად. ეს მოქმედებებია: ახალ სტრიქონზე გადასვლა, ჰორიზონტალური ტაბულირება, სტრიქონის დასაწყისში გადასვლა და ა.შ. მაგალითად,

```
cout << "Saba \n Ana" << endl;
```

აქ, cout ფუნქციის შესრულების შედეგად ფანჯრის პირველ სტრიქონში გამოჩნდება "Saba", მეორე სტრიქონში კი - "Ana". 2.2 ცხრილში მოცემულია მმართველი სიმბოლოები.

ცხრილი 2.2. მმართველი სიმბოლოები

მმართველი სიმბოლო	აღწერა
\'	ერთმაგი ბრჭყალი
\"	ორმაგი ბრჭყალი
\\	ირიბი დახრილი ხაზი
\0	Null (სიმბოლო, რომლის კოდია 0)
\a	ზარი
\b	უკან დაბრუნება (BackSpace)
\f	გვერდის გადაფურცვლა
\n	ახალ სტრიქონზე გადასვლა
\r	სტრიქონის დასაწყისში გადასვლა
\t	ჰორიზონტალური ტაბულირება
\v	ვერტიკალური ტაბულირება

მმართველი სიმბოლოების გამოყენების შედეგები კარგად ჩანს კონსოლურ პროგრამებთან მუშაობისას. **კონსოლური პროგრამა** ჩვენთან ურთიერთქმედებს კლავიატურისა და ეკრანის საშუალებით, რომლებიც სიმბოლოურ რეჟიმში მუშაობენ.

bool ტიპი

ბულის ტიპის ცვლადი (ლოგიკური ცვლადი) ორ მნიშვნელობას იღებს: **true** (ჭეშმარიტი) და **false** (მცდარი). მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//   პროგრამაში ხდება ლოგიკურ ცვლადთან მუშაობის დემონსტრირება
bool b1, b2;

b1 = true;
cin >> b2;
cout << "b1 = " << b1 << " b2 = " << b2 << endl;

system("pause");
return 0;
```

```
}
```

true მნიშვნელობის მაგივრად უნდა შევიტანოთ 1, false მნიშვნელობის მაგივრად კი - 0.

string ტიპი

სტრიქონებთან სამუშაოდ შეგვიძლია გამოვიყენოთ string ტიპი. ამისათვის, პროგრამის ფაილს დასაწყისში უნდა დაემატოს #include "string" დირექტივა. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//      პროგრამაში ხდება სტრიქონებთან მუშაობის დემონსტრირება
string strqoni1 = "Ana da Saba";
string strqoni2;
cin >> strqoni2;
cout << strqoni1 << endl;
strqoni1 = strqoni2;
cout << strqoni1 << endl;

system("pause");
return 0;
}
```

სამწუხაროდ, cin ფუნქციის გამოყენებით შესაძლებელია მხოლოდ ისეთი სტრიქონების შეტანა, რომლებიც ინტერვალებს არ შეიცავს. ეს ფუნქცია ინტერვალს აღიქვამს სტრიქონის დასასრულად. იმისათვის, რომ შევძლოთ ინტერვალების შემცველი სტრიქონების შეტანა, უნდა გამოვიყენოთ getline() ფუნქცია. მოყვანილი პროგრამით ხდება ამ ფუნქციასთან მუშაობის დემონსტრირება.

```
#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//      პროგრამაში ხდება ინტერვალების შემცველ სტრიქონებთან მუშაობის დემონსტრირება
string str1;
getline(cin, str1);           //      შესატანი სტრიქონი: რომან სამხარაძე
cout << str1 << endl;

system("pause");
return 0;
}
```

როგორც ვხედავთ, ამ ფუნქციას ორი პარამეტრი აქვს. პირველია cin ფუნქცია, მეორე კი - სტრიქონული ცვლადი (str1), რომელშიც ჩაიწერება შეტანილი სტრიქონი.

ლიტერალები

ნებისმიერი სახის ფიქსირებულ მნიშვნელობას **ლიტერალი** ეწოდება. ლიტერალი (2.3 ცხრილი) შეიძლება იყოს მთელი რიცხვი, წილადი, სტრიქონი, სიმბოლო ან ლოგიკური მნიშვნელობა. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int mteli1 = -4;           // ლიტერალია - -4
    long mteli2 = 45L;       // ლიტერალია - 45L
    double wiladi = 9.21;   // ლიტერალია - 9.21
    bool logikuri = true;   // ლიტერალია - true
    char simbolo = 'R';     // ლიტერალია - 'R'
    string striqoni = "Saba Samkharadze"; // ლიტერალია - "Saba Samkharadze"
    cout << mteli1 << " " << mteli2 << " "
         << wiladi << " " << logikuri << " "
         << simbolo << " " << striqoni << endl;

    system("pause");
    return 0;
}
```

ცხრილი 2.3. სხვადასხვა ტიპის ლიტერალი.

ტიპი	მაგალითები
char, signed char, unsigned char	'A', 'S', '#', '5'
int	-54, 29713, 0x8FC
unsigned int	25U, 32900U
long	-54L, 29713L
unsigned long	4UL, 987654321UL
float	8.03f
double	6.32
long double	1.73L
bool	true, false

კიდევ ერთი მაგალითი.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
```

```

char c1 = 'a';
signed char sc1 = '5';
unsigned char uc1 = '2';
unsigned int ui1 = 1;
long l1 = 6;
unsigned long ul1 = 7;
float f1 = 4.4;
double d1 = 5.5;
long double ld1 = 2.2;
bool b1 = true;
cout << "c1 = " << c1 << " sc1 = " << sc1 << " uc1 = " << uc1 << endl
    << "ui1 = " << ui1 << " l1 = " << l1 << " ul1 = " << ul1 << endl
    << "f1 = " << f1 << " d1 = " << d1 << " ld1 = " << ld1 << endl
    << "b1 = " << b1 << endl;

system("pause");
return 0;
}

```

მონაცემთა ტიპისთვის სინონიმის განსაზღვრა

მონაცემთა არსებულ ტიპს შეგვიძლია დავარქვათ ჩვენთვის საჭირო სახელი. ამ მიზნით გამოიყენება საკვანძო სიტყვა **typedef**. მისი სინტაქსია

typedef ტიპი სინონიმი

საკვანძო სიტყვა typedef შეგვიძლია მივუთითოთ "int _tmain(int argc, _TCHAR* argv[])"

სტრუქტურის წინ:

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
typedef unsigned int Ricxvebi;
```

```
typedef long long Didi_Mteli;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
Ricxvebi mteli1 = 9;
```

```
Didi_Mteli mteli2 = 795LL;
```

```
cout << "mteli1 = " << mteli1 << " mteli2 = " << mteli2 << endl;
```

```
system("pause");
```

```
return 0;
```

```
}
```

აქ, ფაილის დასაწყისში, იქმნება unsigned int ტიპის ფსევდონიმი - Ricxvebi. შემდეგ ის გამოიყენება პროგრამაში unsigned int ტიპის ნაცვლად. ანალოგიურად ვქმნით და ვიყენებთ Didi_Mteli ფსევდონიმს.

ხშირად ფსევდონიმების გამოყენება ამარტივებს რთულ გამოცხადებებს ერთი სახელის გამოყენების გზით და აადვილებს საწყისი კოდის წაკითხვას.

მუდმივა

მუდმივა (კონსტანტა, constant) არის ფიქსირებული მნიშვნელობა. მისი გამოცხადება იწყება **const** სიტყვით. მუდმივა შეიძლება იყოს ნებისმიერი ტიპის მქონე მნიშვნელობა, მაგალითად მთელი რიცხვები: 5, -10; წილადი რიცხვები: 23.54, 1.09; სიმბოლოები: 'Q', 'g'; სტრიქონი: "Visual C++" და ა.შ. მოცემული პროგრამით ხდება მუდმივასთან მუშაობის დემონსტრირება.

```
#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//    პროგრამით ხდება მუდმივასთან მუშაობის დემონსტრირება
//    მუდმივას განსაზღვრა
const int mteli_mudmiva = 1024;
const char simbolo_mudmiva = 'a';
const float wiladi_mudmiva1 = 12.09;
const double wiladi_mudmiva2 = 456.302;
const string str1 = "C++";
int cvladi, jami;
double shedegi;

cin >> cvladi;
jami = cvladi + mteli_mudmiva;
shedegi = wiladi_mudmiva1 + wiladi_mudmiva2;
cout << jami << endl;
cout << simbolo_mudmiva << endl;
cout << wiladi_mudmiva1 << endl;
cout << shedegi << endl;
cout << str1 << endl;

system("pause");
return 0;
}
```

ცვლადი და მისი ინიციალიზება

ცვლადი (variable) არის მეხსიერების სახელდებული უბანი, რომელშიც მნიშვნელობა იწვება. ეს მნიშვნელობა შეიძლება შეიცვალოს პროგრამის მუშაობის პროცესში. ოპერატორს, რომლის საშუალებითაც ცვლადის გამოცხადება ხდება, შემდეგი სინტაქსი აქვს:

ტიპი ცვლადის_სახელი;

სადაც, **ტიპი** ცვლადის ტიპია, **ცვლადის_სახელი** - კი მისი სახელი. ნებისმიერი ცვლადი უნდა

გამოცხადდეს მის გამოყენებამდე. წინააღმდეგ შემთხვევაში, აღიძვრება შეცდომა. ამასთან, ცვლადს უნდა მიენიჭოს მხოლოდ შესაბამისი ტიპის მნიშვნელობა. მაგალითად, bool ტიპის ცვლადს უნდა მიენიჭოს true ან false მნიშვნელობა და არა წილადი ან სხვა.

ცვლადის ინიციალიზება

ცვლადის გამოცხადებისას მისთვის მნიშვნელობის მინიჭებას ცვლადის *ინიციალიზება* ეწოდება. ცვლადის ინიციალიზების ოპერატორის სინტაქსია:

ტიპი ცვლადის_სახელი = მნიშვნელობა;

მნიშვნელობის ტიპი უნდა ემთხვეოდეს ცვლადის ტიპს. როგორც აღვნიშნეთ, ცვლადს მნიშვნელობა მის გამოყენებამდე უნდა მივანიჭოთ. ამის გაკეთება შეიძლება ცვლადის გამოცხადებისას ან გამოცხადების შემდეგ. ამ პროგრამით ხდება ცვლადების ინიციალიზების დემონსტრირება.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
{
//      პროგრამით ხდება ცვლადების ინიციალიზების დემონსტრირება
int ricxvi3, ricxvi1 = 25, ricxvi2 = 30;
char simbolo = L'S';
double wiladi1 = 150.75;
cout << ricxvi1 << " " << wiladi1 << " " << simbolo << endl;

system("pause");
return 0;
}
```

ცვლადების დინამიკური ინიციალიზება

ცვლადების ინიციალიზება შესაძლებელია, აგრეთვე, დინამიკურად, ე.ი. პროგრამის შესრულების დროს. ამ შემთხვევაში, ცვლადის ინიციალიზება ხდება არა პროგრამის დასაწყისში, არამედ მის ნებისმიერ ადგილას. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
{
//      პროგრამით ხდება ცვლადის დინამიკური ინიციალიზების დემონსტრირება
int simagle = 5, sigane, sigrdze;

cin >> sigane >> sigrdze;
// moculoba ცვლადის ინიციალიზება დინამიკურად ხდება
int moculoba = simagle * sigane * sigrdze;
cout << moculoba;
```

```
system("pause");
return 0;
}
```

ოპერატორები

ოპერატორი არის სიმბოლო, რომელიც კომპილატორს ატყობინებს თუ რა ოპერაცია უნდა შესრულდეს. C++ ენაში არსებობს ოპერატორების ოთხი ძირითადი კლასი: არითმეტიკის, ლოგიკის, შედარებისა და ბიტობრივი.

მინიჭების ოპერატორი

მისი სინტაქსია:

ცვლადი = გამოსახულება;

ცვლადს და გამოსახულებას ერთნაირი ტიპი უნდა ჰქონდეს. მინიჭების ოპერატორი ცვლადს ანიჭებს გამოსახულების მნიშვნელობას. შეგახსენებთ, რომ მინიჭების ოპერატორის მარჯვნივ მოთავსებულ გამოსახულებას უნდა ჰქონდეს მინიჭების ოპერატორის მარცხნივ მოთავსებული ცვლადის ტიპი.

მინიჭების ოპერატორი საშუალებას იძლევა, აგრეთვე შევქმნათ მინიჭებების მიმდევრობა:

```
int ricxvi1, ricxvi2, ricxvi3;
ricxvi1 = ricxvi2 = ricxvi3 = 50;
```

აქ სამივე ცვლადს ენიჭება მნიშვნელობა 50. ასეთი მინიჭება საშუალებას გვაძლევს რამდენიმე ცვლადს ერთდროულად მივანიჭოთ ერთი და იგივე მნიშვნელობა.

მინიჭების შედგენილი ოპერატორი

მინიჭების შედგენილ (შემოკლებულ) ოპერატორში არითმეტიკის ოპერატორები მოთავსებულია მინიჭების ოპერატორის მარცხნივ. მისი სინტაქსია:

ცვლადი ოპერატორი= გამოსახულება;

სადაც, **ოპერატორი** არის არითმეტიკის ან ლოგიკის ოპერატორი. მაგალითად, გამოსახულება:

```
jami = jami + 25;
```

შედგენილი ოპერატორის გამოყენებით შეგვიძლია ასე ჩავწეროთ:

```
jami += 25;
```

+= ოპერატორების წყვილი მიუთითებს, რომ jami ცვლადს უნდა მიენიჭოს მნიშვნელობა jami + 25.

შედგენილი (შემოკლებული) ოპერატორებია:

+= -= *= /= %= &= |= ^=

მინიჭების შედგენილი ოპერატორები მინიჭების ჩვეულებრივ ოპერატორზე კომპაქტურია. ეს, განსაკუთრებით იგრძნობა მაშინ, როდესაც გრძელ სახელებს ვიყენებთ. ამ შემთხვევაში, სახელის შეტანა ერთხელ გვიწევს. მეორეც, მათი გამოყენება აჩქარებს კოდის კომპილაციას, რადგან ოპერანდის შეფასება მხოლოდ ერთხელ ხდება.

არითმეტიკის ოპერატორები

არითმეტიკის ოპერატორები (2.4 ცხრილი) შეგვიძლია გამოვიყენოთ როგორც მთელი, ისე წილადი რიცხვისთვის.

მოვიყვანოთ ზოგიერთი განმარტება. როდესაც გაყოფის ოპერატორს ვიყენებთ მთელი რიცხვებისთვის, მაშინ შედეგი იქნება მთელი რიცხვი, ნაშთი კი უარიყოფა. მაგალითად, 20 / 6 გაყოფის შედეგი იქნება მთელი რიცხვი 3. იმისათვის, რომ განაყოფი იყოს წილადი საჭიროა, რომ მრიცხველი ან მნიშვნელი ან ორივე ერთად იყოს წილადი. ნაშთის მისაღებად უნდა გამოვიყენოთ % ოპერატორი. მაგალითად, 20 % 6 ოპერაციის შედეგი იქნება 2. % ოპერატორის გამოყენებით შეგვიძლია დავადგინოთ კენტია რიცხვი თუ - ლუწი. მაგალითად, R % 2 გამოსახულების გამოთვლისას თუ ნაშთი 1-ის ტოლია, მაშინ R რიცხვი კენტია, თუ ნაშთი 0-ის ტოლია, მაშინ R რიცხვი ლუწია. ანალოგიურად, შეგვიძლია დავადგინოთ ერთი რიცხვი მეორის ჯერადია თუ არა.

ცხრილი 2.4. არითმეტიკის ოპერატორები

ოპერატორი	მნიშვნელობა
+	შეკრება
-	გამოკლება (და უნარული მინუსი)
*	გამრავლება
/	გაყოფა
%	მოდულით გაყოფა (ნაშთის გამოყოფა)
++	ინკრემენტი
--	დეკრემენტი

მოცემული პროგრამებით ხდება არითმეტიკის ოპერატორების გამოყენების დემონსტრირება მთელი რიცხვებისათვის.

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
// პროგრამით ხდება არითმეტიკის ოპერაციების მუშაობის
```

```
// დემონსტრირება მთელი რიცხვებისათვის
```

```
int ricxvi1, ricxvi2, jami, sxvaoba, namravli, ganayofi, nashti;
```

```
cin >> ricxvi1 >> ricxvi2;
```

```
jami = ricxvi1 + ricxvi2;
```

```
sxvaoba = ricxvi1 - ricxvi2;
```

```
namravli = ricxvi1 * ricxvi2;
```

```
ganayofi = ricxvi1 / ricxvi2;
```

```
nashti = ricxvi1 % ricxvi2;
```

```
cout << "jami = " << jami << " sxvaoba = " << sxvaoba << endl;
```

```
cout << "namravli = " << namravli << endl;
```

```
cout << "ganayofi = " << ganayofi << " nashti = " << nashti << endl
```

```
system("pause");
```

```
return 0;
}
```

ქვემოთ მოცემული პროგრამით ხდება არითმეტიკის ოპერატორების მუშაობის დემონსტრირება წილადი რიცხვებისთვის.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// პროგრამით ხდება არითმეტიკის ოპერატორების მუშაობის
// დემონსტრირება წილადი რიცხვებისათვის
double wiladi1, wiladi2, jami, sxvaoba, namravli, ganayofi, nashti;
int ricxvi1, ricxvi2;

cin >> ricxvi1 >> ricxvi2;
cin >> wiladi1 >> wiladi2;
jami = wiladi1 + wiladi2;
sxvaoba = wiladi1 - wiladi2;
namravli = wiladi1 * wiladi2;
ganayofi = wiladi1 / wiladi2;
nashti = ricxvi1 % ricxvi2;
cout << "jami = " << jami << " sxvaoba = " << sxvaoba << endl;
cout << "namravli = " << namravli << endl;
cout << "ganayofi = " << ganayofi << " nashti = " << nashti << endl;

system("pause");
return 0;
}
```

ოპერატორების პრიორიტეტი

ჩვეულებრივ, გამოსახულებაში ჯერ სრულდება პრიორიტეტული ოპერაციები, შემდეგ კი - ნაკლებად პრიორიტეტული. 2.5 ცხრილში ოპერაციები დალაგებულია პრიორიტეტების კლების მიხედვით. შესაბამისად, ცხრილის ზედა ნაწილში მოთავსებულია მაღალი პრიორიტეტის ოპერაციები, ქვედა ნაწილში კი - დაბალი პრიორიტეტის. ერთ სტრიქონში მოთავსებულ ოპერაციებს ერთნაირი პრიორიტეტი აქვს. თუ გამოსახულებაში გამოყენებული არ არის ფრჩხილები, მაშინ თანაბარი პრიორიტეტის მქონე ოპერაციები შესრულდება იმ მიმდევრობით, რომელსაც განსაზღვრავს მათი **ასოციაციურობა**. „მარცხენა“ ასოციაციურობის შემთხვევაში გამოსახულების ყველაზე მარცხენა ოპერაცია შესრულდება პირველ რიგში, შემდეგ კი მიმდევრობით სრულდება ყველა ოპერაცია მარცხნიდან მარჯვნივ. მაგალითად,

```
shedegi = ricxvi1 + ricxvi2 + ricxvi3;
```

გამოსახულებაში შეკრების ოპერაციები შესრულდება იმ მიმდევრობით, რა მიმდევრობითაც არის ჩაწერილი, რადგანაც + ოპერაციას აქვს მარცხენა ასოციაციურობა. ანუ ჯერ შეიკრიბება ricxvi1 და ricxvi2. მიღებულ ჯამს დაემატება ricxvi3.

როგორც ცხრილიდან ჩანს, *, / და % ოპერატორებს უფრო მაღალი პრიორიტეტი აქვთ, ვიდრე + და - . განვიხილოთ გამოსახულება:

$$y = x1 + x2 * x3 - x4 / x5 ;$$

ის შემდეგნაირად გამოითვლება. გამოსახულებაში პირველია + ოპერატორი. სანამ ეს ოპერატორი შესრულდება კომპილატორი ამოწმებს მის მარჯვნივ რომელი ოპერატორია. ესაა *, რომელსაც უფრო მაღალი პრიორიტეტი აქვს, ვიდრე + ოპერატორს. შემდეგ, კომპილატორი ამოწმებს * ოპერატორის მარჯვნივ მდებარე ოპერატორს. ესაა -, რომელსაც უფრო დაბალი პრიორიტეტი აქვს, ვიდრე * ოპერატორს. ამიტომ პირველ რიგში შესრულდება * ოპერატორი. რადგან + და - ოპერატორებს თანაბარი პრიორიტეტები აქვს და + რიგით პირველია გამოსახულებაში, ამიტომ შემდეგ შესრულდება + ოპერატორი. სანამ კომპილატორი შეასრულებს - ოპერატორს, ის ამოწმებს მის მარჯვნივ რომელი ოპერატორია მოთავსებული. ესაა / ოპერატორი, რომელსაც უფრო მაღალი პრიორიტეტი აქვს ვიდრე - ოპერატორს. ამიტომ, ჯერ შესრულდება / ოპერატორი, შემდეგ კი - ოპერატორი.

ახლა განვიხილოთ მეორე გამოსახულება:

$$y = (x1 + x2) * x3 - (x4 + x5) / x6 ;$$

აქ ოპერაციები შემდეგი მიმდევრობით შესრულდება: 1) პირველ მრგვალ ფრჩხილებში მოთავსებული გამოსახულება; 2) * ოპერატორი; 3) მეორე მრგვალ ფრჩხილებში მოთავსებული გამოსახულება; 4) / ოპერატორი; 5) - ოპერატორი.

როგორც ვხედავთ, პრიორიტეტების ცოდნა საჭიროა იმისთვის, რომ სწორად ჩავწეროთ გამოსახულება და შესაბამისად, მივიღოთ სწორი შედეგი.

უნარულია ოპერაცია, რომელსაც ერთი ოპერანდი აქვს. **ბინარულია** ოპერაცია, რომელსაც ორი ოპერანდი აქვს. **ტერნარულია** ოპერაცია, რომელსაც სამი ოპერანდი აქვს. უნარულ ოპერაციას ყოველთვის უფრო მაღალი პრიორიტეტი აქვს ვიდრე - ბინარულს.

ცხრილი 2.5. ოპერაციების პრიორიტეტები

ოპერაცია	ასოციაციურობა
::	მარცხენა
() [] ->	მარცხენა
! “ + (უნარული) -(უნარული) ++ -- &(უნარული) *(უნარული) (ტიპის გარდაქმნა) static_cast const_cast dynamic_cast reinterpret_cast sizeof new delete ... typeid	მარჯვენა
.*(უნარული) ->*	მარცხენა
*/%	მარცხენა
+ -	მარცხენა
<< >>	მარცხენა
< <= > >=	მარცხენა
== !=	მარცხენა
&	მარცხენა
^	მარცხენა
	მარცხენა
&&	მარცხენა
	მარცხენა
?:(პირობის ოპერაცია)	მარჯვენა
= *= /= %= += -= &= ^= = <<= >>=	მარჯვენა
‘	მარცხენა

ინკრემენტისა და დეკრემენტის ოპერატორები

ინკრემენტის ოპერატორი (++) ერთით ზრდის თავისი ოპერანდის (არგუმენტის)

მნიშვნელობას. დეკრემენტის ოპერატორი (--) ერთით ამცირებს თავისი ოპერანდის მნიშვნელობას. ოპერანდი არის ცვლადი, რომელზეც სრულდება ინკრემენტის ან დეკრემენტის ოპერაცია. განვიხილოთ მაგალითები.

```
x = x + 1;
```

ოპერატორი ასრულებს იმავე მოქმედებებს, რასაც

```
x++; ან ++x;
```

ოპერატორი. ანალოგიურად,

```
x = x - 1;
```

ოპერატორი ასრულებს იმავე მოქმედებებს, რასაც

```
x--; ან --x;
```

ოპერატორი.

როგორც ვხედავთ ეს ოპერატორები შეიძლება მიეთითოს როგორც ოპერანდამდე (პრეფიქსი), ისე ოპერანდის შემდეგ (პოსტფიქსი). ახლა ვნახოთ რა განსხვავებაა ამ პოზიციებს შორის. თუ ინკრემენტის ან დეკრემენტის ოპერატორი ოპერანდის წინაა, მაშინ ჯერ გაიზრდება ან შემცირდება ოპერანდის მნიშვნელობა, ხოლო შემდეგ მოხდება მისი გამოყენება გამოსახულებაში. თუ ინკრემენტის ან დეკრემენტის ოპერატორი მითითებულია ოპერანდის შემდეგ, მაშინ ჯერ მოხდება ამ ოპერანდის გამოყენება გამოსახულებაში და შემდეგ მისი მნიშვნელობის გაზრდა ან შემცირება.

ქვემოთ მოცემული პროგრამით ხდება ინკრემენტის ოპერატორის გამოყენების დემონსტრირება.

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
// ++ ოპერატორის მუშაობის დემონსტრირება
```

```
int incr, shedegi;
```

```
incr = 5;
```

```
// პრეფიქსური ინკრემენტი
```

```
shedegi = ++incr; // incr = 6, shedegi = 6
```

```
cout << "incrementi = " << incr << " shedegi = " << shedegi << endl;
```

```
incr = 5;
```

```
// პოსტფიქსური ინკრემენტი
```

```
shedegi = incr++; // shedegi = 5, incr = 6
```

```
cout << "incrementi = " << incr << " shedegi = " << shedegi << endl;
```

```
system("pause");
```

```
return 0;
```

```
}
```

თავდაპირველად, incr ცვლადს ენიჭება მნიშვნელობა 5. მომდევნო სტრიქონში ამ ცვლადის მარცხნივ წერია ++, ამიტომ მისი მნიშვნელობა ჯერ გაიზრდება ერთით და გახდება 6-ის ტოლი, შემდეგ კი მიენიჭება shedegi ცვლადს. შემდეგ, incr ცვლადს კვლავ ენიჭება მნიშვნელობა 5. მომდევნო სტრიქონში ++ მოთავსებულია incr ცვლადის მარჯვნივ. ამიტომ, მისი მნიშვნელობა ჯერ მიენიჭება shedegi ცვლადს, შემდეგ კი გაიზრდება ერთით და გახდება 6-ის ტოლი.

ქვემოთ მოცემული პროგრამით ხდება დეკრემენტის ოპერატორის გამოყენების

დემონსტრირება.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//      -- ოპერატორის მუშაობის დემონსტრირება
int decr, shedegi;

decr = 5;
//      პრეფიქსური დეკრემენტი
shedegi = --decr;                          //      decr = 4, shedegi = 4
cout << "decrementi = " << decr << "shedegi = " << shedegi << endl;
decr = 5;
//      პოსტფიქსური დეკრემენტი
shedegi = decr--;                          //      shedegi = 5, decr = 4
cout << "decrementi = " << decr << "shedegi = " << shedegi << endl;

system("pause");
return 0;
}
```

შედარებისა და ლოგიკის ოპერატორები

შედარების ოპერატორი (2.6 ცხრილი) ადარებს ორ მნიშვნელობას და გაცემს bool ტიპის true ან false მნიშვნელობას. ლოგიკის ოპერატორი (2.7 ცხრილი) ასრულებს ლოგიკის ოპერაციას bool ტიპის მნიშვნელობებზე.

C++ ენაში == და != ოპერატორები შეიძლება გამოვიყენოთ ყველა ობიექტის მიმართ მათი შედარებისთვის ტოლობაზე ან უტოლობაზე. შედარების დანარჩენი ოპერატორები შეგვიძლია გამოვიყენოთ მხოლოდ მონაცემების ჩამოთვლადი ტიპების მიმართ, რომლებიც მოწესრიგებულია თავიანთ სტრუქტურაში. ასეთია მაგალითად, რიცხვების მოწესრიგებული სტრუქტურა 1, 2, 3 და ა.შ., ან ანბანის მიხედვით დალაგებული სიმბოლოები. შედეგად, შედარების ყველა ოპერატორი შეგვიძლია გამოვიყენოთ მონაცემთა ყველა რიცხვითი ტიპის მიმართ. bool ტიპის მნიშვნელობების შედარება შეიძლება მხოლოდ ტოლობაზე ან უტოლობაზე.

2.8 ცხრილში მოცემულია ლოგიკის ოპერატორების ჭეშმარიტების ცხრილი. როგორც ცხრილიდან ჩანს & ოპერატორი გაცემს true მნიშვნელობას მხოლოდ მაშინ, როდესაც მისი ორივე ოპერანდის მნიშვნელობაა true. წინააღმდეგ შემთხვევაში, ის გაცემს false მნიშვნელობას. | ოპერატორი გაცემს true მნიშვნელობას, თუ მისი ერთ-ერთი ოპერანდის მნიშვნელობაა true. წინააღმდეგ შემთხვევაში, ის გაცემს false მნიშვნელობას. ! ოპერატორი გაცემს თავისი ოპერანდის ლოგიკურად საწინააღმდეგო მნიშვნელობას. ^ ოპერატორი გაცემს false მნიშვნელობას თუ მის ორივე ოპერანდს ერთნაირი მნიშვნელობა აქვს, წინააღმდეგ შემთხვევაში, გაცემს true მნიშვნელობას.

ცხრილი 2.6. შედარების ოპერატორები

ოპერატორი	მნიშვნელობა
==	ტოლია
!=	არ არის ტოლი
>	მეტია
<	ნაკლებია
>=	მეტია ან ტოლი
<=	ნაკლებია ან ტოლი

ცხრილი 2.7. ლოგიკის ოპერატორები

ოპერატორი	მნიშვნელობა
&	AND (და)
	OR (ან)
^	XOR (გამომრიცხავი ან)
&&	Short-circuit AND (სწრაფი და ოპერატორი)
	Short-circuit OR (სწრაფი ან ოპერატორი)
!	NOT (არა)

ცხრილი 2.8. ლოგიკის ოპერატორების ჭეშმარიტების ცხრილი

B1	B2	B1 & B2	B1 B2	B1 ^ B2	!B1
true	true	true	true	false	false
true	false	false	true	true	false
false	true	false	true	true	true
false	false	false	false	false	true

ამ პროგრამით ხდება ლოგიკის ოპერატორების მუშაობის დემონსტრირება.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ლოგიკის ოპერატორების მუშაობის დემონსტრირება
bool b1, b2, b3, b4, b5, b6;

b1 = true;
b2 = false;
b3 = b1 & b2; // b3 = false
b4 = b1 | b2; // b4 = true
b5 = b1 ^ b2; // b5 = true
b6 = !b1; // b6 = false
cout << b3 << endl;
cout << b4 << endl;
cout << b5 << endl;
cout << b6 << endl;
```

```
system("pause");
return 0;
}
```

მოცემული პროგრამით ხდება შედარების ოპერატორების მუშაობის დემონსტრირება.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// შედარების ოპერატორების მუშაობის დემონსტრირება
int ricxvi1 = 5, ricxvi2 = 10;
bool shedegi;

shedegi = ricxvi1 > 0; // shedegi = true
cout << shedegi << endl;
shedegi = ( ricxvi1 < 2 ) && ( ricxvi2 > 7 ); // shedegi = false
cout << shedegi << endl;
shedegi = ( ricxvi1 == 3 ) || ( ricxvi2 != 5 ); // shedegi = true
cout << shedegi << endl;

system("pause");
return 0;
}
```

ამ პროგრამის „shedegi = ricxvi1 > 0;“ სტრიქონში ricxvi1 > 0 გამოსახულება true მნიშვნელობას გასცემს, რადგან ricxvi1 = 5. true მნიშვნელობა shedegi ცვლადს მიენიჭება. „shedegi = (ricxvi1 < 2) && (ricxvi2 > 7);“ სტრიქონში ricxvi1 < 2 გამოსახულება false მნიშვნელობას გასცემს, რადგან ricxvi1 = 5, ხოლო ricxvi2 > 7 გამოსახულება კი - true მნიშვნელობას, რადგან ricxvi2 = 10. მივიღებთ false && true გამოსახულებას, რომელიც false მნიშვნელობას გასცემს. ეს მნიშვნელობა shedegi ცვლადს მიენიჭება. ანალოგიურად გამოითვლება „shedegi = (ricxvi1 == 3) || (ricxvi2 != 5);“ გამოსახულების მნიშვნელობა.

&& და || ლოგიკის სწრაფი ოპერატორებია. მათი გამოყენებით პროგრამა უფრო სწრაფად სრულდება. თუ && ოპერატორის შესრულებისას პირველმა ოპერანდმა მიიღო false მნიშვნელობა, მაშინ შედეგი იქნება false მიუხედავად მეორე ოპერანდის მნიშვნელობისა. თუ || ოპერატორის შესრულებისას პირველმა ოპერანდმა მიიღო true მნიშვნელობა, მაშინ შედეგი იქნება true მიუხედავად მეორე ოპერატორის მნიშვნელობისა. ასეთ შემთხვევებში აღარ არის საჭირო მეორე ოპერანდის შეფასება, რადგან მისი მნიშვნელობა საბოლოო შედეგს ვერ შეცვლის. შედეგად, პროგრამა უფრო სწრაფად სრულდება.

გამოსახულება. მათემატიკის ფუნქციები

გამოსახულება არის ოპერატორების, ცვლადებისა და ფუნქციების კომბინაცია.

მაგალითად,

$y = x + z - 5;$

$y = x - \sin(x) + 10.97;$

პროგრამის კითხვა რომ გაუმჯობესდეს გამოსახულებებში ტაბულირების სიმბოლოები

და ინტერვალები გამოიყენება. მაგალითად, მოცემული ორი გამოსახულებიდან მეორის წაკითხვა უფრო ადვილია, ვიდრე პირველის:

$$y = x/5 + z * (w - 2);$$

$$y = x / 5 + z * (w - 2);$$

მრგვალი ფრჩხილები ზრდის მასში მოთავსებული ოპერაციების პრიორიტეტს. ისინი აქ ისევე გამოიყენება, როგორც ალგებრაში. მრგვალი ფრჩხილები გამოიყენება, აგრეთვე, პროგრამის კოტხვის გასაუმჯობესებლად. მაგალითად, მოცემული ორი სტრიქონიდან მეორე უკეთესად იკითხება:

$$y = x * 5 + z / 7.2 - w;$$

$$y = (x * 5) + (z / 7.2) - w;$$

მათემატიკის ფუნქციები აღწერილია `math.h` სტანდარტულ ბიბლიოთეკაში (2.9 ცხრილი). თუმცა, `#include "math.h"` დირექტივის მითითება აუცილებელი არ არის. მათემატიკის ფუნქციების უმრავლესობა გასცემს `double` ტიპის შედეგს და აქვს `double` ტიპის პარამეტრი.

ცხრილი 2.9. მათემატიკის ფუნქციები

ფუნქცია	დანიშნულება
<code>abs(x)</code>	აბსოლუტური მნიშვნელობა
<code>sin(x)</code>	სინუსი
<code>cos(x)</code>	კოსინუსი
<code>tan(x)</code>	ტანგენსი
<code>sqrt(x)</code>	კვადრატული ფესვი
<code>exp(x)</code>	ექსპონენტა
<code>log(x)</code>	ლოგარითმი
<code>log10(x)</code>	ათობითი ლოგარითმი
<code>pow(x, y)</code>	ახარისხება
<code>floor(x)</code>	დამრგვალება ნაკლებობით
<code>ceil(x)</code>	დამრგვალება მეტობით

შევადგინოთ პროგრამა, რომლის შეასრულებს კვადრატული ფესვის ამოღებას რიცხვიდან.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// კვადრატული ფესვის ამოღება
double ricxvi, shedegi;

cin >> ricxvi;
shedegi = sqrt(ricxvi);
cout << "shedegi = " << shedegi << endl;

system("pause");
return 0;
}
```


შევადგინოთ კიდეც ერთი პროგრამა, რომელიც გამოთვლის მოცემული გამოსახულების მნიშვნელობას:

$$y = \frac{\sqrt{x^5 + \sin x}}{1 - e^x}$$

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ფორმულის გამოთვლა
double ricxvi, shedegi;

cin >> ricxvi;
shedegi = sqrt(pow(ricxvi,5) + sin(ricxvi)) / ( 1 - exp(ricxvi) );
cout << shedegi << endl;

system("pause");
return 0;
}
```

დანართი 1-ში მოცემულია ამოცანები თავების მიხედვით.

ტიპის გარდაქმნა

გამოთვლები შეიძლება შესრულდეს მხოლოდ ერთი ტიპის მნიშვნელობებს შორის. როდესაც გამოსახულება შეიცავს სხვადასხვა ტიპის ცვლადებსა და კონსტანტებს (მუდმივებს), მაშინ თითოეული ოპერაციის შესრულების წინ კომპილატორს მოუწევს ტიპების გარდაქმნის შესრულება. ტიპების გარდაქმნის პროცესს *დაყვანა* ეწოდება. მაგალითად, თუ გვინდა შევკრიბოთ int და double ტიპის რიცხვები, მაშინ int ტიპის რიცხვი ჯერ გარდაიქმნება double ტიპად და შემდეგ შესრულდება შეკრება. int ტიპის რიცხვის მნიშვნელობა არ იცვლება. უბრალოდ წილადად გარდაქმნილი მნიშვნელობა მეხსიერებაში ინახება გამოთვლების დამთავრებამდე.

გამოსათვლელი გამოსახულება იყოფა ოროპერანდიან რამდენიმე ოპერაციად. მაგალითად,

```
shedegi = 8 / 2 + 9 - 3;
```

გამოსახულება შემდეგი ოროპერანდიანი ოპერაციებისგან შედგება:

- 1) 8 / 2. შედეგად მიიღება 4;
- 2) შემდეგ, შესრულდება 4 + 9 ოპერაცია. შედეგად მიიღება 13;
- 3) შემდეგ, შესრულდება 13 - 3 ოპერაცია. შედეგად მიიღება 10.

როგორც ვხედავთ, ოპერანდების დაყვანის ოპერაცია საჭიროა მხოლოდ ოპერანდების წყვილების მიმართ გადაწყვეტილების მიღების დროს. ამიტომ, სხვადასხვა ტიპის ოპერანდის წყვილისთვის მოწმდება ქვემოთ მოცემული წესები იმ მიმდევრობით, როგორც არიან ისინი მოცემული. თუ წესის გამოყენება შეიძლება ოპერანდების კონკრეტული წყვილის მიმართ, მაშინ მოხდება ამ წესის გამოყენება. ოპერანდების დაყვანის ეს წესებია:

1. თუ ერთი ოპერანდის ტიპია long double, მაშინ მეორე ოპერანდიც გარდაიქმნება ამ ტიპის ოპერანდად.

2. თუ ერთი ოპერანდის ტიპია double, მაშინ მეორე ოპერანდიც გარდაიქმნება ამ ტიპის ოპერანდად.
3. თუ ერთი ოპერანდის ტიპია float, მაშინ მეორე ოპერანდიც გარდაიქმნება ამ ტიპის ოპერანდად.
4. char, signed char, unsigned char, short ან unsigned short ტიპის ოპერანდი გარდაიქმნება int ტიპის ოპერანდად.
5. ჩამოთვლადი ტიპი გარდაიქმნება int, unsigned int, long ან unsigned long ტიპად, იმაზე დამოკიდებულებით, თუ რომელი ტიპია საკმარისი, რომ დაიტოს ჩამოთვლების დიაპაზონი.
6. თუ ერთი ოპერანდის ტიპია unsigned long, მაშინ მეორე ოპერანდიც გარდაიქმნება ამ ტიპის ოპერანდად.
7. თუ ერთი ოპერანდის ტიპია long, მეორე ოპერანდი ტიპი კი - unsigned int, მაშინ ორივე ოპერანდი გარდაიქმნება unsigned long ტიპის ოპერანდად.
8. თუ ერთი ოპერანდის ტიპია long, მაშინ მეორე ოპერანდიც გარდაიქმნება ამ ტიპის ოპერანდად.

ტიპების გარდაქმნის საბაზო პრინციპი ასეთია: ყოველთვის გარდაიქმნება ის ოპერანდი, რომლის ტიპის დიაპაზონი ნაკლებია მეორე ოპერანდის ტიპის დიაპაზონზე. მაგალითად, თუ პირველი ოპერანდის ტიპია int და მეორე ოპერანდის ტიპი - long, მაშინ პირველი ოპერანდის ტიპი გარდაიქმნება long ტიპად. ეს, თავის მხრივ, ზრდის სწორი შედეგის მიღების ალბათობას. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
double wiladi_d = 25.0;
int mteli = 10;
float wiladi_f = 4.0f;
char simbolo = 5;
```

```
wiladi_d = ( wiladi_d + mteli ) / ( mteli - simbolo ) * wiladi_f - simbolo * wiladi_f;
cout << wiladi_d << endl;
```

```
system("pause");
return 0;
}
```

პროგრამაში გამოითვლება შემდეგი გამოსახულების მნიშვნელობა:

```
wiladi_d = ( wiladi_d + mteli ) / ( mteli - simbolo ) * wiladi_f - simbolo * wiladi_f;
```

პირველად გამოითვლება (wiladi_d + mteli) გამოსახულების მნიშვნელობა. + ოპერაციის ოპერანდებს სხვადასხვა ტიპი აქვს, კერძოდ კი double და int. ამიტომ, მისი გამოთვლისას გამოყენებული იქნება მეორე წესი. ამ წესის თანახმად mteli ცვლადის ტიპი გარდაიქმნება double ტიპად. ამ გარდაქმნის შემდეგ შესრულდება შეკრების ოპერაცია. შედეგი იქნება 35.0 .

შემდეგ გამოითვლება (mteli - simbolo) გამოსახულების მნიშვნელობა. - ოპერაციის ოპერანდებს აქვს int და char ტიპი, ამიტომ გამოყენებული იქნება მეოთხე წესი. ამ წესის თანახმად simbolo ცვლადის ტიპი გარდაიქმნება int ტიპად. ამ გარდაქმნის შემდეგ შესრულდება გამოკლების ოპერაცია. შედეგი იქნება 5.

შემდეგ, 35.0 გაიყოფა 5-ზე. მეორე წესის თანახმად 5 გარდაიქმნება წილადად (5.0) და შესრულდება გაყოფის ოპერაცია. შედეგი იქნება 7.0. შემდეგ, ეს წილადი უნდა გამრავლდეს 4.0-ზე. ამ შემთხვევაშიც, ვიყენებთ მეორე წესს, რომლის თანახმად 4.0 უნდა გარდაიქმნას double ტიპის წილადად. გარდაქმნის შემდეგ შესრულდება გამრავლების ოპერაცია. შედეგი იქნება 28.0.

შემდეგ, გამოითვლება simbolo * wiladi_f გამოსახულება. აქ გამოყენებული იქნება მესამე წესი. შედეგად, simbolo ოპერანდის ტიპი გარდაიქმნება float ტიპად. გამოთვლის შედეგი იქნება 20.0.

ბოლოს შესრულდება გამოკლების ოპერაცია. რადგან 28.0 რიცხვის ტიპია double, ამიტომ გამოყენებული იქნება მეორე წესი. რიცხვი 20.0 float ტიპიდან გარდაიქმნება double ტიპად. გამოკლების ოპერაციის შედეგი იქნება 8.0. ეს მნიშვნელობა მიენიჭება wiladi_d ცვლადს.

დაყვანა მინიჭების ოპერატორებში

ჩვენ შეგვიძლია გამოვიწვიოთ არაცხადი გარდაქმნა, თუ მინიჭების ოპერატორის მარჯვნივ მოვათავსებთ გამოსახულებას, რომლის ტიპი განსხვავდება მინიჭების ოპერატორის მარცხნივ მოთავსებული ცვლადის ტიპისგან. ამან შეიძლება გამოიწვიოს მნიშვნელობის შეცვლა და შესაბამისად, მონაცემების დაკარგვა. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//      პროგრამით ხდება არაცხადი გარდაქმნის დემონსტრირება
int mteli = 0;
float wiladi = 3.7f;
mteli = wiladi;                //      mteli ცვლადს მიენიჭება 3
cout << mteli << endl;

system("pause");
return 0;
}
```

აქ mteli მთელი ტიპის ცვლადს ენიჭება wiladi წილადი ცვლადის მნიშვნელობა. ამ მინიჭების დროს წილადი ნაწილი იკარგება (0.7) და mteli ცვლადს ენიჭება მნიშვნელობა 3.

ცხადი დაყვანა

თუ გამოსახულება შეიცავს სხვადასხვა ტიპის ცვლადებს, მაშინ კომპილატორი, საჭიროების შემთხვევაში, ავტომატურად ასრულებს ტიპების დაყვანას. არსებობს ტიპების დაყვანის ოთხი სახე:

1. **static_cast** - სტატიკური დაყვანა. ტიპის დაყვანა სრულდება საწყისი კოდის კომპილირებისას. პროგრამის შესრულებისას დაყვანის უსაფრთხოების შემოწმება აღარ შესრულდება.
2. **dynamic_cast** - დინამიკური დაყვანა. ტიპის დაყვანის უსაფრთხოება შემოწმდება პროგრამის შესრულებისას.
3. **const_cast** - გამორიცხავს გამოსახულების მუდმივობას.
4. **reinterpret_cast** - უპირობო დაყვანა.

ჩვენ შეგვიძლია იძულებით მივუთითოთ ერთი ტიპის მეორე ტიპზე დაყვანა, რასაც **ცხადი დაყვანა** ეწოდება. მისი სინტაქსია:

static_cast <დაყვანის_ტიპი> (გამოსახულება)

აქ **static_cast** საკვანძო სიტყვა მიუთითებს, რომ სრულდება **სტატიკური დაყვანა**. **გამოსახულების** მნიშვნელობა დაყვანილი უნდა იყოს **დაყვანის_ტიპზე**. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
// პროგრამით ხდება ცხადი დაყვანის დემონსტრირება
```

```
double wiladi1 = 45.8;
```

```
double wiladi2 = 20.3;
```

```
int mteli;
```

```
mteli = static_cast <int> (wiladi1) - static_cast <int> (wiladi2); // mteli = 25
```

```
cout << mteli << endl;
```

```
system("pause");
```

```
return 0;
```

```
}
```

პროგრამით გამოითვლება შემდეგი გამოსახულების მნიშვნელობა:

```
mteli = static_cast <int> (wiladi1) - static_cast <int> (wiladi2);
```

გამოკლების შესრულებამდე სრულდება ტიპების დაყვანა. **wiladi1** ცვლადი გარდაიქმნება მთელ რიცხვად და მისი მნიშვნელობა გახდება 45. ადგილი აქვს მნიშვნელობის დაკარგვას. იგივე ეხება **wiladi2** ცვლადს. მისი მნიშვნელობა გახდება 20-ის ტოლი. გამოკლების შედეგად მიიღება მთელი რიცხვი 25, რომელიც მიენიჭება **mteli** ცვლადს.

ტიპების დაყვანისას უნდა გვახსოვდეს, რომ შეიძლება დაიკარგოს ინფორმაცია.

მაგალითი:

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
// ინფორმაციის დაკარგვის დემონსტრირება
```

```
double wiladi1 = 0.8;
```

```
float wiladi2 = 0.7f;
```

```
long mteli1;
```

```
int mteli2;
```

```
mteli1 = wiladi1; // mteli1 = 0
```

```
mteli2 = wiladi2; // mteli2 = 0
```

```
cout << "mteli1 = " << mteli1 << " mteli2 = " << mteli2 << endl;
```

```
system("pause");
```

```
return 0;
}
```

ამ პროგრამაში,

```
mteli1 = wiladi1;
mteli2 = wiladi2;
```

მინიჭებების შედეგად mteli1 და mteli2 ცვლადებს მიენიჭებათ 0-ები და შესაბამისად, წილადი ნაწილები იკარგება.

double ტიპის დაყვანამ float ტიპზე შეიძლება გამოიწვიოს ინფორმაციის დაკარგვა. ანალოგიურად, long ტიპის დაყვანამ int ტიპზე შეიძლება გამოიწვიოს ინფორმაციის დაკარგვა და ა.შ. ამიტომ, უმჯობესია ტიპების დაყვანა გამოვიყენოთ მხოლოდ აუცილებლობის შემთხვევაში.

ტიპების დაყვანის ძველი სტილი

ტიპების დაყვანის ძველი სტილის სინტაქსია:

(დაყვანის_ტიპი) გამოსახულება;

მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
// ტიპების დაყვანის ძველი სტილის დემონსტრირება
```

```
double wiladi1 = 45.8;
```

```
double wiladi2 = 20.3;
```

```
int mteli;
```

```
mteli = (int) wiladi1 - (int) wiladi2;           // mteli = 25
```

```
cout << mteli << endl;
```

```
system("pause");
```

```
return 0;
```

```
}
```

შენახვის დრო და ხილვადობის უბანი

პროგრამის შესრულებისას ყველა ცვლადს აქვს სიცოცხლის (არსებობის) შეზღუდული დრო. ისინი არსებობას იწყებენ მათი გამოცხადების მომენტიდან და არსებობენ (ცოცხლობენ) გარკვეულ მომენტამდე, მაგრამ არაუგვიანეს პროგრამის დასრულების მომენტისა. ცვლადის არსებობის პერიოდი განისაზღვრება *შენახვის დროით* (storage duration). არსებობს ცვლადების შენახვის სამი სახე:

- ავტომატური;
- სტატიკური;
- დინამიკური.

იმაზე დამოკიდებულებით თუ როგორ ვქმნით ცვლადს, ის შეიძლება იყოს ავტომატური, სტატიკური ან დინამიკური. გარდა ამისა, ცვლადებს ახასიათებს ხილვადობის უბანი (scope). *ცვლადის ხილვადობის უბანი* არის პროგრამის ნაწილი, რომელშიც ცვლადის

სახელი განსაზღვრულია. ხილვადობის უბნის გარეთ ჩვენ არ შეგვიძლია მივმართოთ ცვლადს, თუმცა ის შეიძლება არსებობდეს ამ უბნის გარეთ.

ავტომატური ცვლადი

ცვლადს, რომელსაც ვაცხადებდით ბლოკში ანუ ფიგურულ ფრჩხილებში *ავტომატური* ეწოდება. ბლოკი არის ოპერატორების ერთობლიობა. ის შეიძლება შედგებოდეს ერთი ან მეტი ოპერატორისგან. ბლოკი იწყება "{" სიმბოლოთი და მთავრდება "}" სიმბოლოთი. ავტომატურ ცვლადს აქვს *ლოკალური ხილვადობის უბანი* ანუ *ბლოკის ხილვადობის უბანი*. ავტომატური ცვლადი ხილულია დაწყებული გამოცხადების ადგილიდან (წერტილიდან) მისი გამოცხადების შემცველი ბლოკის ბოლომდე. მეხსიერების უბანი, რომელსაც ავტომატური ცვლადი იკავებს, ავტომატურად გამოიყოფა სტეკში.

სტეკი არის სია (მეხსიერების უბანი), რომლის ელემენტებთან მიმართვა სრულდება პრინციპით - „უკანასკნელი მოვიდა - პირველი წავიდა“ (LIFO – last-in, first-out). მაგალითად, თუ სტეკში ჯერ ჩავწერთ 1-ს, შემდეგ 7-ს და ბოლოს 5-ს, მაშინ წაკითხვისას, ჯერ წაკითხება 5, შემდეგ 7 და ბოლოს - 4. თითოეულ პროგრამას საკუთარი სტეკი გამოეყოფა.

ავტომატური ცვლადი იქმნება მისი გამოცხადების დროს და არსებობას ამთავრებს მისი გამოცხადების შემცველი ბლოკის ბოლოს ანუ იქ, სადაც იმყოფება დამხურავი ფიგურული ფრჩხილი. ყოველთვის, როდესაც სრულდება ოპერატორების ბლოკი, რომელიც შეიცავს ავტომატური ცვლადის გამოცხადებას, ავტომატური ცვლადი ხელახლა იქმნება და თუ განსაზღვრულია მისი საწყისი მნიშვნელობა, ის ხელახლა ინიცირდება ამ მნიშვნელობით ყოველი შექმნის დროს. როდესაც ავტომატური ცვლადი იშლება, მის მიერ დაკავებული მეხსიერება თავისუფლდება.

ავტომატური ცვლადის აღნიშვნისთვის შეგვიძლია **auto** საკვანძო სიტყვა გამოვიყენოთ. თუმცა, მისი მითითება აუცილებელი არ არის, რადგან ის ავტომატურად იგულისხმება. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
{
// ავტომატურ ცვლადებთან მუშაობის დემონსტრირება
int ricxv1 = 10;
int ricxv2 = 20;
{
// ხილვადობის ახალი უბნის დასაწყისი
int ricxv1 = 30;
int ricxv3 = 40;
ricxv1 += 30; // შიგა ricxv1 = 60
ricxv3 += 40; // შიგა ricxv3 = 80
cout << "shida ricxv1 = " << ricxv1 << "\nshida ricxv3 = " << ricxv3 << endl;
// ხილვადობის ახალი უბნის დასასრული
}
ricxv1 += 10; // გარე ricxv1 = 20
ricxv2 += 20; // გარე ricxv2 = 40
cout << "gare ricxv1 = " << ricxv1 << "\ngare ricxv2 = " << ricxv2 << endl;
```

```
system("pause");
return 0;
}
```

როგორც ვხედავთ, ხილვადობის ახალ უბანში გამოცხადებულია ricxvi1 ცვლადი. ის განსხვავდება ადრე გამოცხადებული ricxvi1 ცვლადისაგან. ორივე ეს ცვლადი ოპერატიული მეხსიერების სხვადასხვა უბანშია მოთავსებული, ამიტომ სხვადასხვა ცვლადია და შესაბამისად, სხვადასხვა მნიშვნელობა აქვს. შიგა ricxvi1 ცვლადი ფარავს გარე ricxvi1 ცვლადს. ამიტომ, ხილვადობის ახალ უბანში ჩვენ ვერ მივმართავთ გარე ricxvi1 ცვლადს. გარე ricxvi1 ცვლადი ხილული გახდება ხილვადობის უბნიდან გამოსვლისთანავე.

სტატიკური ცვლადი

სტატიკურია ცვლადი, რომელიც განსაზღვრულია როგორც ლოკალური, მაგრამ აგრძელებს არსებობას იმ ბლოკიდან გამოსვლის შემდეგ, რომელშიც ის არის გამოცხადებული. სტატიკური ცვლადის გამოცხადება ხდება static სპეციფიკატორის გამოყენებით. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// სტატიკურ ცვლადთან მუშაობის დემონსტრირება
static int ricxvi1 = 5;
{
static int ricxvi2 = 10;
ricxvi2++;
cout << ricxvi2 << endl;
}
ricxvi1++;
cout << ricxvi1 << endl;

system("pause");
return 0;
}
```

სინამდვილეში სტატიკური ცვლადი არსებობს პროგრამის შესრულების მთელი დროის განმავლობაში, თუნდაც ის გამოცხადებული იყოს ბლოკის შიგნით. სტატიკურ ცვლადს აქვს ბლოკის ხილვადობის უბანი და შენახვის სტატიკური დრო. თუ სტატიკურ ცვლადს არ მივანიჭებთ საწყის მნიშვნელობას, მაშინ მას გაჩუმებით 0 მიენიჭება.

გლობალური ცვლადი

ცვლადს, რომელიც გამოცხადებულია ყველა ბლოკისა და კლასის გარეთ, *გლობალური* ეწოდება. მას აქვთ *გლობალური ხილვადობის უბანი*, რომელსაც აგრეთვე, *გლობალური სახელების სივრცის ხილვადობის უბანი* ან *ფაილის ხილვადობის უბანი* ეწოდება. გლობალური ცვლადი მისაწვდომია ამ ფაილში განსაზღვრული ყველა ფუნქციიდან დაწყებული ამ ცვლადის განსაზღვრის ადგილიდან. თუ გლობალური ცვლადი განსაზღვრულია ფაილის დასაწყისში,

მაშინ იგი მისაწვდომი იქნება ფაილის ნებისმიერი ადგილიდან.

გლობალურ ცვლადს ნაგულისხმევი აქვს *სიცოცხლის სტატიკური დრო*. ამიტომ, ის არსებობს პროგრამის დაწყების მომენტიდან მისი დამთავრების მომენტამდე. თუ გლობალურ ცვლადს არ აქვს მინიჭებული საწყისი მნიშვნელობა, მაშინ მისი ნაგულისხმევი მნიშვნელობაა ნული. გლობალური ცვლადის გამოცხადება შეგვიძლია `int _tmain(int argc, _TCHAR* argv[])` სტრუქტურის წინ. მაგალითი:

```
// გლობალურ ცვლადებთან მუშაობის დემონსტრირება
#include "stdafx.h"
#include "iostream"
using namespace std;

int ricxvi1 = 10;
static int ricxvi2 = 20;
int _tmain(int argc, _TCHAR* argv[])
{
int ricxvi3, ricxvi4, shedegi;

ricxvi3 = 30;
ricxvi4 = 40;
shedegi = ricxvi1 + ricxvi2 + ricxvi3 + ricxvi4;           // shedegi = 100
cout << shedegi << endl;

system("pause");
return 0;
}
```

აქ `ricxvi1` და `ricxvi2` გლობალური ცვლადებია. შეგვიძლია პროგრამის ყველა ცვლადი გამოვაცხადოთ როგორც გლობალური. ასეთ შემთხვევაში გაიზრდება მათი არასწორად შეცვლის რისკი. როდესაც ცვლადი ავტომატურია, ის არსებობს მხოლოდ მაშინ, როდესაც საჭიროა მასთან მუშაობა.

თუ გლობალური ცვლადის სახელი ემთხვევა ლოკალური ცვლადის სახელს, მაშინ ლოკალური ცვლადის სახელი ფარავს გლობალური ცვლადის სახელს იმ ბლოკის შიგნით, რომელშიც ლოკალური ცვლადებია გამოცხადებული. ამის დემონსტრირება ხდება მოცემული პროგრამით:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int ricxvi1 = 10;
static int ricxvi2 = 20;
int _tmain(int argc, _TCHAR* argv[])
{
int ricxvi1 = 88;           // ლოკალური ricxvi1 ცვლადი ფარავს გლობალურ ricxvi1 ცვლადს
int ricxvi2 = 77;           // ლოკალური ricxvi2 ცვლადი ფარავს გლობალურ ricxvi2 ცვლადს
int ricxvi3 = ricxvi1 + ricxvi2;
cout << ricxvi3 << endl;           // გაიცემა 165
system("pause");
return 0;
}
```


თავი 3. მმართველი ოპერატორები

არსებობს სამი კატეგორიის მმართველი ოპერატორი - ამორჩევის (if და switch), იტერაციისა (for, while, do-while და for each) და გადასვლის (break, continue, goto და return).

ამორჩევის ოპერატორები

if ოპერატორი

if ოპერატორი ამოწმებს ლოგიკურ პირობას და თუ ის ჭეშმარიტია, მაშინ შესრულდება საწყისი კოდის მითითებული ბლოკი. მისი სინტაქსია:

```
if ( პირობა ) { ბლოკი1; }  
[ else { ბლოკი2; } ]
```

აქ პირობა არის ლოგიკური გამოსახულება, რომელიც იღებს true ან false მნიშვნელობას. თუ პირობა ჭეშმარიტია, მაშინ შესრულდება ბლოკი1, წინააღმდეგ შემთხვევაში - ბლოკი2. კვადრატული ფრჩხილები ნიშნავს, რომ else სიტყვის ჩაწერა აუცილებელი არ არის. ასეთ შემთხვევაში, if ოპერატორის სინტაქსი იქნება:

```
if ( პირობა ) { ბლოკი1; }
```

თუ პირობა იღებს true მნიშვნელობას, მაშინ შესრულდება ბლოკი1, წინააღმდეგ შემთხვევაში კი - პროგრამის მომდევნო ოპერატორი.

if ოპერატორის მუშაობის საჩვენებლად შევადგინოთ პროგრამა, რომელიც განსაზღვრავს კენტია რიცხვი თუ - ლუწი.

```
#include "stdafx.h"  
#include "iostream"  
using namespace std;  
  
int _tmain(int argc, _TCHAR* argv[])  
{  
// პროგრამა განსაზღვრავს კენტია რიცხვი თუ - ლუწი  
int ricxvi;  
  
cin >> ricxvi;  
if ( ricxvi % 2 == 1 ) cout << "ricxvi kentia" << endl;  
else cout << "ricxvi luwia" << endl;  
  
system("pause");  
return 0;  
}
```

ჩადგმული if ოპერატორი

ჩადგმულია if ოპერატორი, რომელიც მოთავსებულია სხვა if ოპერატორში. ჩადგმული if ოპერატორი იმ შემთხვევაში გამოიყენება, როდესაც მოქმედების შესასრულებლად საჭიროა რამდენიმე პირობის შემოწმება, რომელთა მითითება ერთი if ოპერატორით შეუძლებელია. ჩადგმულ if ოპერატორში else ნაწილი ყოველთვის ეკუთვნის უახლოეს if ოპერატორს.

შევადგინოთ პროგრამა, რომელიც დაადგენს ორ რიცხვს შორის რომელია დადებითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//    პროგრამა დაადგენს ორ რიცხვს შორის რომელია დადებითი
int ricxvi1, ricxvi2;

cin >> ricxvi1 >> ricxvi2;
if ( ricxvi1 >= 0 ) cout << "dadebitia pirveli ricxvi" << endl;
    else    if ( ricxvi2 >= 0 ) cout << "dadebitia meore ricxvi" << endl;
        else cout << "arc erti ricxvi ar aris dadebiti" << endl;

system("pause");
return 0;
}
```

თუ პირველი if ოპერატორის პირობა ჭეშმარიტია, მაშინ ეკრანზე გამოჩნდება შეტყობინება "dadebitia pirveli ricxvi" და პროგრამა მუშაობას დაამთავრებს. წინააღმდეგ შემთხვევაში, შემოწმდება მეორე if ოპერატორის პირობა. თუ ის ჭეშმარიტია მაშინ ეკრანზე გამოჩნდება შეტყობინება "dadebitia meore ricxvi" და პროგრამა მუშაობას დაამთავრებს. თუ არც ერთი პირობა არ არის ჭეშმარიტი, მაშინ ეკრანზე გამოჩნდება შეტყობინება "arc erti ricxvi ar aris dadebiti" და პროგრამა მუშაობას დაამთავრებს.

switch ოპერატორი

switch ოპერატორი საშუალებას გვაძლევს ვარიანტების სიიდან ამოვარჩიოთ საჭირო მოქმედება. მისი სინტაქსია:

```
switch ( გამოსახულება )
{
    case მუდმივა1 :      ოპერატორების მიმდევრობა_1;
                        break;
    case მუდმივა2 :      ოპერატორების მიმდევრობა_2;
                        break;
    ...
    [ default :          ოპერატორების მიმდევრობა_n;
                        break; ]
}
```

აქ *გამოსახულება*-ს უნდა ჰქონდეს მთელი რიცხვა ტიპი - char, short ან int, ან სტრიქონული ტიპი - string. გამოსახულების მნიშვნელობას არ უნდა ჰქონდეს მცურავწერტილიანი ტიპი (double, float). switch ოპერატორის case განშტოებების მუდმივები უნდა იყოს ლიტერალები და ჰქონდეთ გამოსახულებების ტიპთან თავსებადი ტიპი. ამასთან, switch ოპერატორის ერთ ბლოკში მათ არ შეიძლება ერთნაირი მნიშვნელობები ჰქონდეთ.

switch ოპერატორი შემდეგნაირად მუშაობს: გამოსახულების მნიშვნელობა მიმდევრობით შედარდება case სიაში მითითებულ თითოეულ მუდმივას. ერთ-ერთ მუდმივასთან მნიშვნელობის დამთხვევისას შესრულდება მასთან დაკავშირებული

ოპერატორების მიმდევრობა.

ერთ case განშტოებასთან დაკავშირებულმა ოპერატორების მიმდევრობამ მართვა არ უნდა გადასცეს მომდევნო case განშტოების ოპერატორებს. ამის თავიდან ასაცილებლად თითოეული case განშტოების ოპერატორების მიმდევრობა break ოპერატორით უნდა დავამთავროთ.

default განშტოების შესაბამისი ოპერატორების მიმდევრობა მაშინ შესრულდება, როდესაც case განშტოებების არც ერთი მუდმივა არ შეესაბამება გამოსახულების მნიშვნელობას. default განშტოების მითითება აუცილებელი არ არის. თუ ის არ არის მითითებული და switch ოპერატორში არც ერთი მუდმივა არ შეესაბამება გამოსახულების მნიშვნელობას, მაშინ არავითარი მოქმედება არ შესრულდება და მართვა გადაეცემა switch ოპერატორის შემდეგ მოთავსებულ ოპერატორს. თუ გამოსახულების მნიშვნელობა დაემთხვა მუდმივას მნიშვნელობას, მაშინ შესრულდება ამ case განშტოებასთან ასოცირებული ოპერატორების მიმდევრობა break ოპერატორამდე. break ოპერატორი მართვას გადასცემს switch ოპერატორის შემდეგ მოთავსებულ ოპერატორს. დავწეროთ პროგრამა, რომელიც ციფრებს დააფიქსირებს 0 - 2 დიაპაზონში.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//    პროგრამა აფიქსირებს ციფრებს 0 - 2 დიაპაზონში
int cifri;

cin >> cifri;
switch (cifri )
{
    case 0 : cout << "Shetania cifri - " << cifri << endl;
            break;
    case 1 : cout << "Shetania cifri - " << cifri << endl;
            break;
    case 2 : cout << "Shetania cifri - " << cifri << endl;
            break;
    default : cout << "Shetani cifri ar aris diapazonshi: 0 - 2" << endl;
            break;
}

system("pause");
return 0;
}
```

ამ პროგრამაში switch ოპერატორის მართვა ხდება int ტიპის cifri ცვლადის მეშვეობით.

ახლა შევადგინოთ პროგრამა, სადაც switch ოპერატორის მართვა ხორციელდება char ტიპის გამოსახულებით. ამ შემთხვევაში case განშტოების მუდმივები char ტიპისა უნდა იყვნენ. პროგრამა გამოთვლებს ასრულებს იმის მიხედვით თუ არითმეტიკის რომელ ოპერატორს შევიტანთ.

```
#include "stdafx.h"
#include "iostream"
```

```

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//      კალკულატორი
int ricxvi1, ricxvi2, shedegi;
char operacia;

cin >> ricxvi1 >> ricxvi2 >> operacia;
switch ( operacia )
{
case '+':      shedegi = ricxvi1 + ricxvi2;
               cout << shedegi << endl;
               break;
case '-':      shedegi = ricxvi1 - ricxvi2;
               cout << shedegi << endl;
               break;
case '*':      shedegi = ricxvi1 * ricxvi2;
               cout << shedegi << endl;
               break;
case '/':      shedegi = ricxvi1 / ricxvi2;
               cout << shedegi << endl;
               break;
case '%':      shedegi = ricxvi1 % ricxvi2;
               cout << shedegi << endl;
               break;
}

system("pause");
return 0;
}

```

switch ოპერატორის ორი ან მეტი განშტოება შეიძლება დაკავშირებული იყოს ოპერატორების ერთსა და იმავე მიმდევრობასთან. ამას სხვანაირად *ჩავარდნა* ეწოდება. ქვემოთ მოცემული პროგრამა ახდენს ჩავარდნის დემონსტრირებას. პროგრამა განასხვავებს ქართული ანბანის ხმოვან ასოებს.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//      პროგრამით ხდება ჩავარდნის დემონსტრირება
char simbolo;

cin >> simbolo;
switch ( simbolo )
{

```

```

    case 'a' :
    case 'i' :
    case 'o' :
    case 'e' :
    case 'u' : cout << "Es aso khmovania" << endl; break;
    default : cout << "Es aso khmovani ar aris" << endl; break;
}

```

```

system("pause");
return 0;
}

```

თუ simbolo ცვლადი იღებს მნიშვნელობას 'a', 'e', 'i', 'o' ან 'u', მაშინ შესრულდება cout << "Es aso khmovania" << endl; ოპერატორი, წინააღმდეგ შემთხვევაში კი cout << "Es aso khmovani ar aris" << endl; ოპერატორი. switch ოპერატორი შეიძლება ასეც ჩავწეროთ:

```

switch ( simbolo )
{
    case 'a' : case 'i' : case 'o' : case 'e' : case 'u' :
        cout << "Es aso khmovania" << endl; break;
    default : cout << "Es aso khmovani ar aris" << endl; break;
}

```

ჩადგმული switch ოპერატორი

ერთი switch ოპერატორი შეიძლება მოთავსებული იყოს გარე switch ოპერატორში. შიგა switch ოპერატორს ეწოდება *ჩადგმული*. შიგა და გარე switch ოპერატორების case განშტოებების მუდმივებს შეიძლება ერთნაირი მნიშვნელობები ჰქონდეს. ეს არ გამოიწვევს კონფლიქტს პროგრამის შესრულებისას. მაგალითი:

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])

```

```

{
//      პროგრამით ხდება ჩადგმული switch ოპერატორის მუშაობის დემონსტრირება
char simbolo1, simbolo2;

```

```

cin >> simbolo1 >> simbolo2;

```

```

switch ( simbolo1 )

```

```

{
    case 'a' :
        cout << "'a' mudmiva ekutvnis gare switch operators" << endl;
        switch (simbolo2 )
        {
            case 'a' :
                cout << "'a' mudmiva ekutvnis shiga switch operators" << endl;
                break;

```

```

        case 'b' :
            cout << "'b' mudmiva ekutvnis shiga switch operators" << endl;
            break;
    }
    break;
case 'b' :
    cout << "'b' mudmiva ekutvnis gare switch operators" << endl;
    break;
}

system("pause");
return 0;
}

```

იტერაციის ოპერატორები

for ოპერატორი

გამოიყენება ერთი ოპერატორის ან ოპერატორების ბლოკის მრავალჯერ შესასრულებლად. მისი სინტაქსია:

```
for ( [ ინიციალიზატორი ]; [ პირობა ]; [ იტერატორი ] ) [ ციკლის კოდი ];
```

ინიციალიზატორი არის გამოსახულება, რომელიც ციკლის დაწყების წინ გამოითვლება. აქ ხდება ციკლის მმართველი ცვლადის ინიციალიზება. **პირობა** არის ლოგიკური გამოსახულება, რომელიც იღებს true ან false მნიშვნელობას. ის გამოითვლება ციკლის ყოველი იტერაციის წინ. for ციკლის გამეორება ხდება მანამ, სანამ პირობა იღებს true მნიშვნელობას. თუ მან მიიღო false მნიშვნელობა, მაშინ ციკლი მთავრდება და მართვა გადაეცემა for ოპერატორის შემდეგ მოთავსებულ ოპერატორს. **იტერატორი** არის გამოსახულება, რომელიც ზრდის ან ამცირებს (ციკლის) მმართველი ცვლადის მნიშვნელობას. ის გამოითვლება ციკლის ყოველი იტერაციის შემდეგ. **ციკლის კოდი** (ციკლის ტანი) შეიძლება შეიცავდეს ერთ ან მეტ ოპერატორს.

for ციკლი შემდეგნაირად მუშაობს. ჯერ შესრულდება ციკლის მმართველი ცვლადის ინიციალიზება. შემდეგ მოწმდება პირობა. თუ ის ჭეშმარიტია, შესრულდება ციკლის კოდი. შემდეგ იცვლება ციკლის ცვლადის მნიშვნელობა. კვლავ მოწმდება პირობა და ა.შ.

შევადგინოთ პროგრამა, რომელიც ანგარიშობს 1-დან 10-მდე რიცხვების კვადრატს. პროგრამაში ციკლის მმართველი ცვლადი ერთით იზრდება.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
// პროგრამა ანგარიშობს 1-დან 10-მდე რიცხვების კვადრატს
```

```
double ricxvi, kvadrati;
```

```
for ( ricxvi = 1; ricxvi <= 10; ricxvi++ )
```

```
{
```

```

    kvadrati = pow(ricxvi, 2);
    cout << ricxvi << " - " << kvadrati << endl;
}

```

```

system("pause");
return 0;
}

```

ეს პროგრამა შეგვიძლია ასეც ჩავწეროთ:

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{

```

```

//   პროგრამა ანგარიშობს 1-დან 10-მდე რიცხვების კვადრატს
double ricxvi, kvadrati;

```

```

for ( ricxvi = 1; ricxvi <= 10; ricxvi++ )
{
    kvadrati = pow(ricxvi, 2);
    cout << ricxvi << " - " << kvadrati << endl;
}

```

```

system("pause");
return 0;
}

```

ამ შემთხვევაში ფაილის დასაწყისში უნდა მოვათავსოთ #include <cmath> დირექტივა.

ციკლის ცვლადი შეიძლება ზრდიდეს ან ამცირებდეს თავის მნიშვნელობას ნებისმიერი სიდიდით. მოცემულ პროგრამაში ციკლის ცვლადი მცირდება 10 ერთეულით.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{

```

```

//   ციკლის მმართველი ცვლადი იცვლება უარყოფითი ბიჯით
double ricxvi, kvadrati;

```

```

for ( ricxvi = 100; ricxvi >= 1; ricxvi -= 10 )
{
    kvadrati = pow(ricxvi, 2);
    cout << ricxvi << " - " << kvadrati << endl;
}

```

```

system("pause");
return 0;
}

```

როგორც ვიცით, for ციკლის პირობა ციკლის დასაწყისში მოწმდება. ეს იმას ნიშნავს, რომ თუ პირობა თავიდანვე იღებს false მნიშვნელობას, მაშინ ციკლის კოდი არასოდეს არ შესრულდება, მაგალითად,

```
for ( ricxvi1 = 5; ricxvi1 < 3; ricxvi1++ )
    ricxvi2 = ricxvi1 * ricxvi1;
```

ამ ციკლის კოდი არასოდეს არ შესრულდება, რადგან პირობის გამოსახულება თავიდანვე იღებს false მნიშვნელობას.

for ციკლში ერთის ნაცვლად შეგვიძლია ორი მმართველი ცვლადის გამოყენება, მაგალითად,

```
{
int ricxvi1, ricxvi2;

for ( ricxvi1 = 0, ricxvi2 = 10; ricxvi1 < ricxvi2; ricxvi1++, ricxvi2-- )
    cout << ricxvi1 << " " << ricxvi2 << endl;
}
```

აქ ინიციალიზების ორი ოპერატორი გამოყოფილია მძიმით. ასევე, ორი იტერაციული გამოსახულება გამოყოფილია მძიმით. როდესაც ციკლი იწყებს მუშაობას ორივე ცვლადს ენიჭება საწყისი მნიშვნელობები. ყოველი იტერაციის შემდეგ ricxvi1 ცვლადი მნიშვნელობა ერთით იზრდება, ricxvi2 ცვლადის მნიშვნელობა კი - ერთით მცირდება. for ციკლის მართვა რამდენიმე მმართველი ცვლადის გამოყენებით ზოგჯერ საკმაოდ მოხერხებულია და ამარტივებს ალგორითმებს. ციკლში დასაშვებია ნებისმიერი რაოდენობის მმართველი ცვლადის გამოყენება, მაგრამ პრაქტიკულად, ორ ცვლადზე მეტი იშვიათად გამოიყენება.

for ციკლის ჩაწერისას შეგვიძლია არ მივუთითოთ ინიციალიზების, პირობის ან იტერაციის ნაწილი. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
// პროგრამაში for ოპერატორი გამოყენებულია იტერაციის ნაწილის გარეშე
int ricxvi1;
```

```
// ამ ციკლში მითითებული არ არის იტერაციის ნაწილი
```

```
for ( ricxvi1 = 0; ricxvi1 < 10; )
{
    cout << ricxvi1 << endl;
    ricxvi1++;
}
```

```
system("pause");
```

```
return 0;
```

```
}
```

აქ არ არის მითითებული იტერაციის ნაწილი, სამაგიეროდ, მმართველი ცვლადი ერთით იზრდება ციკლის ტანში.

მოცემულ პროგრამაში გამოყენებულია for ციკლი, რომელშიც არ არის მითითებული ინიციალიზებისა და იტერაციის ნაწილები.


```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//    პროგრამაში for ოპერატორი გამოყენებულია ინიციალიზებისა და იტერაციის
//    ნაწილების გარეშე
int ricxvil = 0;

//    ციკლი ინიციალიზატორისა და იტერატორის გარეშე
for ( ; ricxvil < 10; )
{
    cout << ricxvil << endl;
    ricxvil++;
}

system("pause");
return 0;
}

```

აქ ricxvil ცვლადის ინიციალიზება ხდება ციკლამდე. ასე ძირითადად ვიქცევით მაშინ, როდესაც მმართველი ცვლადი საწყის მნიშვნელობას იღებს რაიმე რთული გამოსახულების გამოთვლის შედეგად.

თუ for ციკლის არც ერთი ნაწილი არ არის მითითებული, მაშინ მივიღებთ უსასრულო ციკლს:

```

for ( ; ; )
{
// ციკლის კოდი
}

```

უსასრულო ციკლის შესაწყვეტად შეგვიძლია break ოპერატორის გამოყენება.

for ოპერატორს შეიძლება არ ჰქონდეს, აგრეთვე, ციკლის კოდი ანუ ის იყოს ცარიელი. ეს შესაძლებელია, რადგან ნულოვანი ოპერატორი სინტაქსურად დასაშვებია. ქვემოთ მოცემულია პროგრამა, რომელშიც ასეთი ციკლი გამოიყენება 1-დან 5-მდე რიცხვების შესაკრებად.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//    პროგრამაში გამოყენებულია for ოპერატორი ციკლის კოდის გარეშე
int ricxvi, jami = 0;

for ( ricxvi = 1; ricxvi <= 5; jami += ricxvi++ );           //    ამ ციკლში კოდი არ არის
cout << jami << endl;
system("pause");
return 0;
}

```

პროგრამიდან ჩანს, რომ ჯამის გამოთვლა ხდება for ოპერატორის სათაურში. შედეგად, ციკლის კოდი საჭირო არ არის. რაც შეეხება გამოსახულებას `jami += ricxvi++;` ის შემდეგნაირად მუშაობს. `jami` ცვლადს ენიჭება მისსავე მნიშვნელობას დამატებული `ricxvi` ცვლადის მნიშვნელობა. შემდეგ, `ricxvi` მნიშვნელობა ერთით იზრდება. ეს ოპერატორი შემდეგი ორი ოპერატორის ანალოგიურია:

```
jami = jami + ricxvi;
ricxvi++;
```

ხშირად მმართველი ცვლადი გამოიყენება მხოლოდ for ოპერატორის შიგნით. ასეთ შემთხვევაში, მისი გამოცხადება ხდება ციკლის საინიციალიზაციო ნაწილში. განვიხილოთ პროგრამა, რომელიც შეკრებს პირველ ხუთ რიცხვს.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
{
//    პროგრამაში გამოყენებულია for ოპერატორი, რომელშიც გამოცხადებულია
//    მმართველი ცვლადი
int jami = 0;

for ( int ricxvi = 1; ricxvi <= 5; ricxvi++ )
    jami += ricxvi;
cout << jami << endl;

system("pause");
return 0;
}
```

`ricxvi` მმართველი ცვლადი გამოცხადებულია for ციკლის საინიციალიზაციო ნაწილში, ამიტომ ის ხილულია მხოლოდ ამ ციკლის ფარგლებში. ციკლის გარეთ ის უხილავია. თუ ვაპირებთ მმართველი ცვლადის გამოყენებას პროგრამის სხვა ნაწილში, მაშინ ის უნდა გამოვაცხადოთ for ციკლის გარეთ.

while ოპერატორი

გამოიყენება ერთი ოპერატორის ან ოპერატორების ბლოკის მრავალჯერ შესასრულებლად. მისი სინტაქსია:

```
while ( პირობა )
{
ციკლის კოდი
}
```

აქ **პირობა** არის ლოგიკური გამოსახულება, რომელიც გასცემს true ან false მნიშვნელობას. თუ ის იღებს false მნიშვნელობას, მაშინ ხდება ციკლიდან გამოსვლა და სრულდება მომდევნო ოპერატორი. თუ ის იღებს true მნიშვნელობას, მაშინ შესრულდება ციკლის კოდი.

ამ პროგრამას ეკრანზე გამოაქვს ინგლისური ანბანის ასოები.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```

int _tmain(int argc, _TCHAR* argv[])
{
//      პროგრამას ეკრანზე გამოაქვს ინგლისური ანბანის ასოები
char simbolo = 'a';

while ( simbolo <= 'z' )
{
    cout << simbolo << endl;
    simbolo++;
}

system("pause");
return 0;
}

```

do-while ოპერატორი

გამოიყენება ერთი ოპერატორის ან ოპერატორების ბლოკის მრავალჯერ შესასრულებლად. for და while ციკლებისაგან განსხვავებით, რომლებშიც ჯერ მოწმდება პირობა და შემდეგ სრულდება ციკლის კოდი, do-while ციკლში ჯერ სრულდება ციკლის კოდი, შემდეგ კი შემოწმდება პირობა. მისი სინტაქსია:

```

do
{
ციკლის კოდი
}
while ( პირობა )

```

თუ ციკლში ერთი ოპერატორი სრულდება, მაშინ ფიგურული ფრჩხილების გამოყენება აუცილებელი არ არის, მაგრამ მათ ხშირად იყენებენ do-while ციკლის წაკითხვის გაუმჯობესების მიზნით, რადგან ის ადვილად შეიძლება აგვერიოს while ციკლში.

ქვემოთ მოცემულია პროგრამა, რომელსაც ეკრანზე გამოაქვს ინგლისური ანბანის ასოები 'a'-დან 'k'-მდე.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//      პროგრამას ეკრანზე გამოაქვს ინგლისური ანბანის ასოები
char simbolo = 'a';

do
{
    cout << simbolo << endl;
    simbolo++;
}
while ( simbolo != 'k' );

```

```

system("pause");
return 0;
}

```

გადასვლის ოპერატორები

break ოპერატორი

გვაძლევს ციკლის შესრულების იძულებით შეწყვეტისა და ციკლიდან გამოსვლის საშუალებას. ამ დროს, ციკლის დანარჩენი ოპერატორები არ შესრულდება. მართვა გადაეცემა ციკლის შემდეგ მოთავსებულ ოპერატორს. ქვემოთ მოცემული პროგრამა რიცხვებს ამრავლებს მანამ, სანამ ნამრავლი არ გახდება 30-ზე მეტი.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//    პროგრამა რიცხვებს ამრავლებს მანამ, სანამ ნამრავლი არ გახდება 30-ზე მეტი
int indexi, namravli = 1;

for ( indexi = 1; indexi < 50; indexi++ )
{
    namravli *= indexi;
    if ( namravli > 30 ) break;
    cout << namravli<< endl;
}
indexi--;
cout << "cikli shesrulda " << indexi<< "-jer" << endl;

system("pause");
return 0;
}

```

ციკლი უნდა შესრულდეს 50-ჯერ, მაგრამ break ოპერატორის შესრულება იწვევს მის ვადამდე შეწყვეტას. ციკლი შესრულდება მხოლოდ ოთხჯერ.

continue ოპერატორი

საშუალებას გვაძლევს იძულებით გადავიდეთ მომდევნო იტერაციაზე. continue ოპერატორის შემდეგ მოთავსებული ოპერატორები არ შესრულდება. განვიხილოთ პროგრამა, რომელსაც ეკრანზე გამოაქვს კენტი რიცხვები 1-დან 21-მდე.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
//      პროგრამას ეკრანზე გამოაქვს კენტი რიცხვები 1-დან 21-მდე
int cvladi;

for ( cvladi = 1; cvladi <= 21; cvladi++ )
{
    if ( ( cvladi % 2 ) == 0 ) continue;           //      მომდევნო იტერაციაზე გადასვლა
    cout << cvladi << endl;
}

system("pause");
return 0;
}

```

while და do-while ციკლებში continue ოპერატორის გამოყენება იწვევს მართვის გადაცემას უშუალოდ იტერატორისათვის, რის შემდეგაც ციკლის შესრულება გრძელდება.

goto ოპერატორი

goto არის უპირობო გადასვლის ოპერატორი. მისი სინტაქსია:

goto ჭდე:

სადაც, ჭდე არის იდენტიფიკატორი, რომელსაც ორწერტილი (:) მოსდევს. ის მიუთითებს იმ ადგილს, საიდანაც უნდა გაგრძელდეს პროგრამის შესრულება. თანამედროვე დაპროგრამებაში goto ოპერატორი ნაკლებად გამოიყენება, რადგან მისი გამოყენებისას პროგრამა ხდება ნაკლებად მოდულური და სტრუქტურირებული. თუმცა მისი გამოყენება ზოგჯერ საკმაოდ მოხერხებულია. ჭდე მოქმედებს მხოლოდ მოცემული ფუნქციის შიგნით. მოცემულ პროგრამაში ნაჩვენებია თუ როგორ ხდება ციკლის ორგანიზება goto ოპერატორის გამოყენებით:

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//      goto ოპერატორთან მუშაობა
int ricxvi = 1;

cikli_1:
    ricxvi++;
    if ( ricxvi < 8 ) goto cikli_1;
cout << ricxvi << endl;

system("pause");
return 0;
}

```

ტერნარული ოპერატორი

? ოპერატორს ეწოდება ტერნარული, რადგან მას სამი ოპერანდი აქვს. მისი სინტაქსია:

გამოსახულება1? გამოსახულება2: გამოსახულება3;

აქ **გამოსახულება1** არის ლოგიკური გამოსახულება, ხოლო **გამოსახულება2** და **გამოსახულება3** გამოსახულებებია, რომლებსაც ერთნაირი ტიპი უნდა ჰქონდეს. თავდაპირველად შეფასდება **გამოსახულება1**. თუ მისი მნიშვნელობაა true, მაშინ გაიცემა **გამოსახულება2**-ის მნიშვნელობა, წინააღმდეგ შემთხვევაში კი - **გამოსახულება3**-ის მნიშვნელობა. მაგალითად,

```
max = ricxvi1 > ricxvi2 ? ricxvi1 : ricxvi2;
```

```
min = ricxvi1 < ricxvi2 ? ricxvi1 : ricxvi2;
```

პირველი გამოსახულება ricxvi1 და ricxvi2 რიცხვებიდან ამოირჩევს მაქსიმალურს, მეორე კი - მინიმალურს.

მოცემულ პროგრამაში ხდება ორი რიცხვის გაყოფა ისე, რომ თავიდან ავიცილოთ ნულზე გაყოფა.

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
// 50 იყოფა -3-დან 5-მდე მოთავსებულ რიცხვებზე ისე, რომ ხდება 0-ზე გაყოფის
```

```
// თავიდან ავიღებ
```

```
int shedegi;
```

```
for ( int cvladi = -3; cvladi < 5; cvladi++ )
```

```
{
```

```
    shedegi = cvladi != 0 ? 50 / cvladi : 0;
```

```
    if ( cvladi != 0 ) cout << cvladi << " " << shedegi << endl;
```

```
}
```

```
system("pause");
```

```
return 0;
```

```
}
```

მაგალითები

შევადგინოთ პროგრამა, რომელიც შეკრებს რიცხვებს მითითებულ დიაპაზონში.

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
// მითითებულ დიაპაზონში რიცხვების შეკრების პროგრამა
```

```
int ricxvi, min, max, jami = 0;
```

```

cin >> min >> max;
for ( ricxvi = min; ricxvi <= max; ricxvi++ )
    jami += ricxvi;
cout << jami << endl;

```

```

system("pause");
return 0;
}

```

შევადგინოთ ფაქტორიალის გამოთვლის პროგრამა.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
//    ფაქტორიალის გამოთვლის პროგრამა
int cvladi, ricxvi, shedegi = 1;

```

```

cin >> ricxvi;
for ( cvladi = 1; cvladi <= ricxvi; cvladi++ )
    shedegi *= cvladi;
cout << shedegi << endl;

```

```

system("pause");
return 0;
}

```

შევადგინოთ პროგრამა, რომელიც გამოთვლის მოცემული მწკრივის მნიშვნელობას

$$y = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} .$$

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
//    მწკრივის გამოთვლის პროგრამა
double shedegi = 0, x;
int minusi = -1;

```

```

cin >> x;
for ( int ricxvi = 1; ricxvi <= 9; ricxvi += 2 )
{
    minusi *= -1;
    shedegi += minusi * pow(x, ricxvi) / ricxvi;
}
cout << shedegi << endl;

```

```
system("pause");  
return 0;  
}
```


თავი 4. მასივები და ბიტობრივი ოპერაციები

ISO/ANSI მასივი

მასივი არის ერთი ტიპის მქონე ობიექტების კოლექცია (ერთობლიობა). ის გამოიყენება ერთნაირი ტიპის მქონე რამდენიმე ცვლადის ან ობიექტის შესანახად. მასივის ელემენტთან მიმართვისათვის უნდა მივუთითოთ მასივის სახელი და ელემენტის ინდექსი. მასივი შეიძლება იყოს როგორც ერთგანზომილებიანი, ისე მრავალგანზომილებიანი. იგი მრავალი ამოცანის გადასაწყვეტად გამოიყენება, რადგან ერთი ტიპის მქონე მონაცემების დაჯგუფების მოხერხებული საშუალებაა. მასივში შეგვიძლია შევინახოთ თანამშრომლების გვარები, ხელფასი, ასაკი და ა.შ.

მასივი მიმართვითი ტიპის ობიექტია. მონაცემები მასივში ისეა ორგანიზებული, რომ ადვილია ამ მონაცემებით მანიპულირება. სწორედ ესაა მასივის მთავარი დანიშნულება.

ერთგანზომილებიანი მასივი

ერთგანზომილებიანი მასივში კონკრეტული ელემენტის მდებარეობა ერთი ინდექსით განისაზღვრება. ერთგანზომილებიანი მასივის სინტაქსია:

ტიპი მასივის_სახელი [ზომა];

აქ **ტიპი** განსაზღვრავს მასივის ტიპს ანუ მისი თითოეული ელემენტის ტიპს. **ზომა** არის მასივში ელემენტების რაოდენობა. მასივი რეალიზებულია ობიექტების სახით. ქვემოთ მოცემულ სტრიქონში იქმნება int ტიპის მასივი, რომელიც 10 ელემენტისაგან შედგება და მიმართვითი ტიპის masivi ცვლადს ენიჭება მასზე მიმართვა:

```
int masivi[10];
```

masivi ცვლადი ინახავს მიმართვას მეხსიერების უბანზე (ანუ ამ უბნის მისამართს), რომელიც მასივს გამოეყო. გამოყოფილი მეხსიერების ზომა საკმარისია მასში 10 ცალი int ტიპის ელემენტის მოსათავსებლად. რადგან int ტიპის ცვლადი 4 ბაიტს (32 ბიტს) იკავებს, ამიტომ masivi მასივისათვის მეხსიერებაში სულ $4 \times 10 = 40$ ბაიტი გამოიყოფა.

მასივის ცალკეულ ელემენტთან მიმართვა ხორციელდება ინდექსის გამოყენებით. მისი მეშვეობით ვუთითებთ ელემენტის პოზიციას მასივის ფარგლებში. ნებისმიერი მასივის პირველ ელემენტს აქვს ნულოვანი ინდექსი. რადგან masivi მასივი 10 ელემენტისაგან შედგება, ამიტომ ამ მასივის ინდექსის მნიშვნელობები მოთავსებული იქნება $0 \div 9$ დიაპაზონში. შედეგად, masivi მასივის პირველი ელემენტი აღინიშნება ასე - masivi[0], უკანასკნელი კი - ასე masivi[9].

ქვემოთ მოცემულ პროგრამაში სრულდება masivi მასივის შევსება ციფრებით 0-დან 9-მდე და შემდეგ მათი გამოტანა ეკრანზე.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
{
// მასივის შევსება რიცხვებით და ეკრანზე მათი გამოტანა
int masivi[10]; // masivi მასივის გამოცხადება
int indexi;
// masivi მასივის შევსება რიცხვებით 0-დან 9-მდე
for ( indexi = 0; indexi < 10; indexi++ )
```

```

        masivi[indexi] = indexi;
//      masivi მასივის ელემენტების გამოტანა ეკრანზე
for ( indexi = 0; indexi < 10; indexi++ )
        cout << masivi[indexi] << " ";
cout << endl;

system("pause");
return 0;
}

```

ერთგანზომილებიანი მასივის ინიციალიზება

მასივის ინიციალიზება შესაძლებელია მისი შექმნისთანავე. ერთგანზომილებიანი მასივის ინიციალიზების სინტაქსია:

ტიპი მასივის_სახელი [] { სიდიდე1, სიდიდე2, ..., სიდიდეN};

აქ საწყისი მნიშვნელობები მოცემულია: *სიდიდე1, ... სიდიდეN* სიდიდეებით. მასივის ელემენტებს მნიშვნელობები ენიჭება რიგრიგობით, მარცხნიდან მარჯვნივ, დაწყებული ნულოვანი ინდექსის მქონე ელემენტიდან. ქვემოთ მოცემულია პროგრამა, რომელშიც სრულდება მასივის ინიციალიზება. პროგრამა პოულობს masivi მასივის მაქსიმალურ და მინიმალურ ელემენტებს.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//      პროგრამა პოულობს მასივის მინიმალურ და მაქსიმალურ ელემენტებს
//      masivi მასივის ინიციალიზება
int masivi[] = { 2, -5, 50, 15, -20, 25, 12, -1, 120, 10 };
int indexi, max, min;

max = min = masivi[0];
for ( indexi = 1; indexi < 10; indexi++ )
{
        if ( masivi[indexi] > max ) max = masivi[indexi];
        if ( masivi[indexi] < min ) min = masivi[indexi];
}
cout << "maximaluri mnishvneloba = " << max<< endl <<
"minimaluri mnishvneloba = " << min << endl;

system("pause");
return 0;
}

```

მასივის ინდექსის მნიშვნელობა არ უნდა გასცდეს ამ მასივის ინდექსის მაქსიმალურ მნიშვნელობას, წინააღმდეგ შემთხვევაში, აღიძვრება შესაბამისი შეცდომა.

ჩვენ განვიხილეთ მთელირიცხვითი მასივის მაგალითი. ბუნებრივია, შეგვიძლია შევქმნათ წილადი, ლოგიკური და ა.შ. ტიპის მასივიც:

```
double wiladi_masivi[] = { 1.2, 3.9, 4.7, 6.0, 8.6 };
```

```
bool logikuri_masivi[] { true, false, true };
```

ორგანზომილებიანი მასივი

ორგანზომილებიანი მასივში კონკრეტული ელემენტის მდებარეობა განისაზღვრება ორი ინდექსით. მასივის გამოცხადებისას პირველი ინდექსი მიუთითებს სტრიქონების რაოდენობას, მეორე კი - სვეტების რაოდენობას. მაგალითად, ორგანზომილებიანი მთელრიცხვა cxrili მასივი ზომით 5x4 შეგვიძლია შემდეგნაირად გამოვაცხადოთ:

```
int cxrili[5][4];
```

რიცხვი 5 განსაზღვრავს სტრიქონების რაოდენობას, რიცხვი 4 - კი სვეტების რაოდენობას. მეხსიერება გამოიყოფა $5 \times 4 = 20$ რიცხვისთვის. რადგან მთელი რიცხვი იკავებს 4 ბაიტს, ამიტომ cxrili მასივისთვის მეხსიერებაში გამოიყოფა $4 \times 5 \times 4 = 80$ ბაიტი.

ორგანზომილებიანი მასივის კონკრეტულ ელემენტთან მიმართვისათვის აუცილებელია ორი ინდექსის მითითება:

```
cxrili[3][2] = 15;
```

აქ რიცხვი 3 არის სტრიქონის ნომერი, რიცხვი 2 - კი სვეტის ნომერი.

ქვემოთ მოცემულია პროგრამა, რომელიც ჯერ ავსებს ორგანზომილებიანი მასივს მთელი რიცხვებით, შემდეგ კი ისინი ეკრანზე გამოაქვს:

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
// ორგანზომილებიანი მასივის შევსება რიცხვებით და
```

```
// ეკრანზე მათი გამოტანა
```

```
int cxrili[5][4];
```

```
int striqoni, sveti;
```

```
for ( striqoni = 0; striqoni < 5; striqoni++ )
```

```
    for ( sveti = 0; sveti < 4; sveti++ )
```

```
        cxrili[striqoni][sveti] = ( striqoni * 4 ) + ( sveti + 1 );
```

```
for ( striqoni = 0; striqoni < 5; striqoni++ )
```

```
{
```

```
for ( sveti = 0; sveti < 4; sveti++ )
```

```
    cout << cxrili[striqoni][sveti] << " ";
```

```
cout << endl;
```

```
}
```

```
system("pause");
```

```
return 0;
```

```
}
```

ორგანზომილებიანი მასივის ინიციალიზება

ორგანზომილებიანი მასივების ინიციალიზებისათვის თითოეული განზომილების მნიშვნელობების სია უნდა მოვათავსოთ ცალკე ბლოკში, რომელიც, თავის მხრივ, ფიგურულ ფრჩხილებში უნდა იყოს მოთავსებული. მაგალითად,

```
int cxrili[5][2] = {
    {1,1},
    {2,4},
    {3,9},
    {4,16},
    {5,25}
};
```

აქ ყოველი შიგა ბლოკი მნიშვნელობას ანიჭებს შესაბამისი სტრიქონის ელემენტებს. სტრიქონის ფარგლებში პირველი მნიშვნელობა მოთავსდება სტრიქონის ნულოვან ელემენტში, მეორე მნიშვნელობა - პირველ ელემენტში და ა.შ. ბლოკები ერთმანეთისაგან მძიმეებით გამოიყოფა. ქვემოთ მოცემულ პროგრამაში ხდება ორგანზომილებიანი მასივის ინიციალიზება და მისი ელემენტების ჯამის გამოთვლა.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// პროგრამა გამოთვლის ორგანზომილებიანი მასივის ელემენტების ჯამს
int striqoni, sveti, jami = 0;

// masivi მასივის ინიციალიზება
int masivi[5][2] = {
    {1,1},
    {2,4},
    {3,9},
    {4,16},
    {5,25}
};

// მასივის ელემენტებისა და მათი ჯამის ეკრანზე გამოტანა
for ( striqoni = 0; striqoni < 5; striqoni++ )
{
    for ( sveti = 0; sveti < 2; sveti++ )
    {
jami += masivi[striqoni][sveti];
cout << masivi[striqoni][sveti] << " ";
    }
    cout << endl;
}
cout << jami << endl;

system("pause");
return 0;
}
```

მიმთითებელი

მიმთითებელი (ცვლადი-მიმთითებელი) არის ცვლადი, რომელიც შეიცავს რაიმე ტიპის სხვა ცვლადის მისამართს. მას აქვს სახელი და ტიპი. ტიპი განსაზღვრავს თუ რა ტიპის მონაცემზე მიუთითებს მიმთითებელი. მიმთითებელს, რომელიც მეხსიერებაში double ტიპის რიცხვზე მიუთითებს „double ტიპზე მიმთითებელი“ ეწოდება. თვითონ მიმთითებელი ყოველთვის მთელი რიცხვია.

მიმთითებლები გამოიყენება მასივის ელემენტებზე ოპერაციების შესრულებისას (ხშირად ოპერაციები მასივის ელემენტებზე უფრო სწრაფად სრულდება მიმთითებლების გამოყენებით) და ფუნქციიდან ფუნქციის გარეთ განსაზღვრულ მასივებთან მიმართვისას. ბოლოს, რაც მთავარია, დინამიკურად გამოყოფილ მეხსიერებასთან მუშაობის ერთადერთი გზაა მიმთითებლის გამოყენება.

მიმთითებლის გამოცხადებისას ტიპის სახელის შემდეგ ან ცვლადის სახელის წინ * სიმბოლო უნდა დავწეროთ:

```
double* pricxv1;
```

ან

```
double *pricxv1;
```

აქ, pricxv1 არის რაიმე double ტიპის რიცხვზე მიმთითებელი ანუ pricxv1 უნდა შეიცავდეს რაიმე double ტიპის რიცხვის მისამართს.

C++ ენაში არსებობს შეთანხმება, რომლის თანახმად მიმთითებლის სახელი p ასოთი იწყება. შედეგად, ადვილდება პროგრამაში მიმთითებლის გამოყოფა.

მიმთითებლის გამოყენების ძირითადი არსი ისაა, რომ შესაძლებელია იმ მონაცემებთან მიმართვა, რომლებზეც ის მიუთითებს. ეს ხორციელდება * ოპერაციის (ირიბი მიმართვის ოპერაცია) საშუალებით. მაგალითი:

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
// * ოპერაციასთან მუშაობა
```

```
int ricxv1 = 25, ricxv2;
```

```
int *pricxv1 = &ricxv1;
```

```
*pricxv1 *= 2; // ricxv1 = 50
```

```
(*pricxv1)++; // ricxv1 = 51
```

```
cout << (*pricxv1) << endl;
```

```
ricxv2 = --*pricxv1; // ricxv2 = 50
```

```
cout << ricxv2 << endl;
```

```
system("pause");
```

```
return 0;
```

```
}
```

ამ პროგრამის მეორე სტრიქონში ხდება pricxv1 მიმთითებლის გამოცხადება და მისთვის ricxv1 ცვლადის მისამართის მინიჭება. & ოპერატორი გასცემს შესაბამისი ცვლადის მისამართს, ჩვენს შემთხვევაში - ricxv1 ცვლადის მისამართს.

მომდევნო *pricxv1 *= 2; სტრიქონი. აქ pricxv1 მისამართით მოთავსებული რიცხვი გამრავლდება 2-ზე და შედეგი ჩაიწერება ამავე მისამართით.

როგორც ვხედავთ, ცვლადის გამოცხადებისას * სიმბოლო მიუთითებს, რომ ხდება მიმთითებლის გამოცხადება, ხოლო შემდეგ, პროგრამაში ის მიუთითებს იმას, რომ უნდა მოხდეს ამ მისამართით მოთავსებულ მონაცემთან მიმართვა.

მომდევნოა (*pricxvi)++; სტრიქონი. აქ pricxvi მისამართით მოთავსებული რიცხვი ერთით გაიზრდება. შემდეგ, ხდება ამ რიცხვის გამოტანა ეკრანზე.

ricxvi2 = --*pricxvi; სტრიქონში pricxvi მისამართით მოთავსებული რიცხვი ერთით შემცირდება და მიღებული შედეგი ricxvi2-ს მიენიჭება.

მიმთითებლის ინიციალიზება

არაინიციალიზებული მიმთითებლის გამოყენება ხშირად არის შეცდომის წყარო. არაინიციალიზებულია მიმთითებელი, რომელიც არ შეიცავს ცვლადის მისამართს. როდესაც მიმთითებლის ინიციალიზებას ვახდენთ სხვა ცვლადის მისამართით, ეს ცვლადი გამოცხადებული უნდა იყოს მიმთითებლის გამოცხადებამდე.

მიმთითებლის გამოცხადებისას არ არის აუცილებელი მისი ინიციალიზება მისამართით. მაგალითი:

```
int* pricxvi1 = 0;
```

```
int* pricxvi2;
```

pricxvi1 მიმთითებელი არაფერზე არ მიუთითებს, რადგან 0 მისამართით არ შეიძლება ობიექტების მოთავსება. pricxvi2 მიმთითებელიც არაფერზე არ მიუთითებს.

იმისათვის, რომ შევამოწმოთ შეიცავს თუ არა მიმთითებელი კონკრეტულ მისამართს, უნდა გამოვიყენოთ შედარების ოპერატორი:

```
if ( pricxvi1 == 0 ) cout << "pricxvi - mimtitebeli misamarts ar sheicavs";
```

ეს ოპერატორი შეგვიძლია ასეც ჩავწეროთ:

```
if ( !pricxvi2 ) cout << "pricxvi - mimtitebeli misamarts ar sheicavs";
```

მიმთითებლის ინიციალიზების მაგალითი:

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
int ricxvi = 25;
```

```
int *pricxvi;
```

```
pricxvi = &ricxvi;
```

```
cout << ((int)pricxvi) << endl; // 2812604
```

```
cout << pricxvi << endl; // 25
```

```
system("pause");
```

```
return 0;
```

```
}
```

პროგრამის მესამე სტრიქონში ხდება pricxvi მიმთითებლის ინიციალიზება და მას ენიჭება ricxvi ცვლადის მისამართი.

& ოპერატორი

ცვლადის მისამართის მისაღებად გამოიყენება & ოპერატორი (მისამართის მიღების

ოპერატორი). მიმთითებლისთვის ცვლადის მისამართის მინიჭების მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi = 25;
    int* pricxvi;

    pricxvi = &ricxvi;           // pricxvi მიმთითებელს ენიჭება ricxvi ცვლადის მისამართი
    cout << ((int)pricxvi) << endl; // ricxvi ცვლადის მისამართი - 1375988
    cout << pricxvi << endl;      // ricxvi ცვლადის მისამართი - 0014FEF4
    cout << (&ricxvi) << endl;   // ricxvi ცვლადის მისამართი - 0014FEF4
    cout << (*pricxvi) << endl;  // ricxvi ცვლადის მნიშვნელობა - 25

    system("pause");
    return 0;
}
```

აქ, ეკრანზე გამოჩნდება pricxvi მიმთითებელში მოთავსებული მისამართი. მისამართი ჯერ გამოჩნდება თვლის ათობით სისტემაში - 2812604, შემდეგ კი თექვსმეტობით სისტემაში - 0014FEF4. ბოლოს გამოჩნდება pricxvi მისამართით მოთავსებული მნიშვნელობა - 25.

მიმთითებელი შეიძლება მიმართავდეს ნებისმიერი ტიპის მონაცემს და ასევე შესაძლებელია ნებისმიერი ტიპის მონაცემის მისამართის მიღება. განვიხილოთ რამდენიმე მაგალითი.

მოყვანილ პროგრამაში მიმთითებელი წილადს მიმართავს:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    double ricxvi = 25.75;
    double *pricxvi;

    pricxvi = &ricxvi;           // pricxvi არის წილადზე მიმთითებელი
    cout << ((int)pricxvi) << endl; // ricxvi ცვლადის მისამართი - 2424084
    cout << pricxvi << endl;      // ricxvi ცვლადის მისამართი - 0024FD14
    cout << (&ricxvi) << endl;   // ricxvi ცვლადის მისამართი - 0024FD14
    cout << (*pricxvi) << endl;  // ricxvi ცვლადის მნიშვნელობა - 25.75

    system("pause");
    return 0;
}
```

მოყვანილ პროგრამაში მიმთითებელი სიმბოლოს მიმართავს:

```
#include "stdafx.h"
#include "iostream"
```

```

using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
char simbolo = 'a';
char *pchar;

pchar = &simbolo; // pchar არის სიმბოლოზე მიმთითებელი
cout << ((int)pchar) << endl; // simbolo ცვლადის მისამართი - 3112604
cout << pchar << endl; // simbolo ცვლადის მისამართი -
cout << (&simbolo) << endl; // simbolo ცვლადის მისამართი -
cout << (*pchar) << endl; // simbolo ცვლადის მნიშვნელობა - a

system("pause");
return 0;
}

```

მოყვანილ პროგრამაში მიმთითებელი სტრიქონს მიმართავს:

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
string striqoni = "roman samkharadze";
string *pstriqoni; // pstriqoni არის სტრიქონზე მიმთითებელი

pstriqoni = &striqoni;
cout << ((int)pstriqoni) << endl; // 1240742 - striqoni ცვლადის მისამართი
cout << (*pstriqoni) << endl; // roman samkharadze

system("pause");
return 0;
}

```

იმისათვის, რომ ამ პროგრამამ კორექტულად იმუშაოს, ფაილის დასაწყისში უნდა დავუმატოთ #include "string" დირექტივა.

მიმთითებელი და მასივი

მასივის სახელი ზოგჯერ იქცევა მიმთითებლის მსგავსად. ხშირად, როდესაც ვიყენებთ ერთგანზომილებიანი მასივის სახელს, ის ავტომატურად გარდაიქმნება ამ მასივის პირველ ელემენტზე მიმთითებლად. ეს არ ეხება იმ შემთხვევას, როდესაც მასივის სახელი არის sizeof ოპერაციის ოპერანდი.

თუ გვაქვს შემდეგი გამოცხადებები:

```

int* pmasivi;
int masivi[10];

```

მამინ შეგვიძლია დავწეროთ ასეთი მინიჭების ოპერატორი:


```
pmasivi = masivi;
```

ამ დროს masivi მასივის პირველი ელემენტის მისამართი მიენიჭება pmasivi მიმთითებელს. მასივის სახელის გამოყენება თავისთავად ნიშნავს ამ მასივის მისამართზე მიმართვას. თუ მასივის სახელს ვიყენებთ ინდექსთან ერთად, მაშინ ეს ნიშნავს მიმართვას იმ ელემენტის მნიშვნელობასთან, რომელიც ინდექსის მნიშვნელობას შეესაბამება. ამიტომ თუ გვინდა, რომ მიმთითებელში შევინახოთ მასივის ელემენტის მისამართი, უნდა გამოვიყენოთ მისამართის მიღების ოპერაცია:

```
pmasivi = &masivi[2];
```

აქ pmasivi მიმთითებელს მიენიჭება masivi მასივის რიგით მესამე ელემენტის მისამართი (რომლის ინდექსია 2). მოცემული პროგრამით ხდება ზემოთ ნათქვამის დემონსტრირება.

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
// მასივზე მიმთითებელთან მუშაობა
```

```
int* pmasivi;
```

```
int masivi[10] = { 1, 2, 3, 4, 5 };
```

```
pmasivi = masivi;
```

```
cout << pmasivi << endl;
```

```
// გამოჩნდება მასივის პირველი ელემენტის
```

```
// მისამართი - 0019FDBC
```

```
pmasivi = &masivi[2];
```

```
cout << pmasivi << endl;
```

```
// გამოჩნდება მასივის ინდექსად მეორე (რიგით მესამე)
```

```
// ელემენტის მისამართი - 0019FDC4
```

```
system("pause");
```

```
return 0;
```

```
}
```

როგორც ვხედავთ, ეკრანზე გამოჩნდება მასივის პირველი ელემენტის მისამართი (მისი ინდექსია 0). შემდეგ, გამოჩნდება მასივის იმ ელემენტის მისამართი, რომლის ინდექსია 2.

მიმთითებლის არითმეტიკა

მიმთითებლებზე შეგვიძლია შევასრულოთ შეკრების, გამოკლებისა და შედარების ოპერაციები. მიმთითებლების არითმეტიკა არაცხადად გულისხმობს, რომ მიმთითებელი მიუთითებს მასივზე და არითმეტიკის ოპერაციები სრულდება იმ მისამართზე, რომელიც მიმთითებელშია მოთავსებული. დაფუძვით, შევასრულოთ მინიჭების ოპერატორი:

```
pmasivi = &masivi[3];
```

ამ შემთხვევაში, pmasivi+1 მიმართავს masivi[4] ელემენტს. ამიტომ, (pmasivi + 1)-ის ნაცვლად შეგვიძლია ჩავწეროთ pmasivi++;

ეს ოპერატორი pmasivi მიმთითებელში მოთავსებულ მისამართს ზრდის ბაიტების იმ რაოდენობით, რომელსაც იკავებს masivi მასივის თითოეული ელემენტი. ზოგად შემთხვევაში, pmasivi + n გამოსახულება, სადაც n ნებისმიერი მთელი რიცხვია, pmasivi მიმთითებელში მოთავსებულ მისამართს უმატებს n * sizeof(int) მნიშვნელობას. როგორც ვხედავთ, სრულდება მამტაბირება ტიპის მიხედვით. მაგალითი:

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])  
{  
int *pmasivi;  
int masivi[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

```
pmasivi = masivi;  
pmasivi = &masivi[2];  
*pmasivi++; // მიმართვა მოხდება მასივის მომდევნო, მესამე ინდექსის მქონე ელემენტთან - 4  
cout << (*pmasivi) << endl;
```

```
system("pause");  
return 0;  
}
```

თუ `pmasivi = &masivi[0]`; , მაშინ ეკვივალენტურია `*pmasivi` და `masivi[0]`; `(*pmasivi+1)` და `masivi[1]`; `(*pmasivi+2)` და `masivi[2]` და ა.შ. ჩანაწერები.

იგივე ეხება `double`, `char` და ა.შ. ტიპის მასივებს. `double` ტიპის შემთხვევაში გვექნება `n*sizeof(double)` და მისამართის ინკრემენტის ან დეკრემენტის შემთხვევაში ის 8 ერთეულით გაიზრდება ან შემცირდება. `char` ტიპის შემთხვევაში გვექნება `n * sizeof(char)` და მისამართის ინკრემენტის ან დეკრემენტის შემთხვევაში ის 2 ერთეულით გაიზრდება ან შემცირდება.

მაგალითი.

```
#include "stdafx.h"  
#include "iostream"  
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])  
{  
int ricxvi = 25;  
int* pricxvi;  
  
pricxvi = &ricxvi;  
cout << ((int)pricxvi) << endl; // 1240742  
cout << (*pricxvi) << endl; // 25  
pricxvi++;  
cout << ((int)pricxvi) << endl; // 1240746  
pricxvi += 2;  
cout << ((int)pricxvi) << endl; // 1240754  
  
system("pause");  
return 0;  
}
```

აქ, ეკრანზე გამოგვაქვს `ricxvi` ცვლადის მისამართი, რომელიც `pricxvi` მიმთითებელშია მოთავსებული. შემდეგ გამოგვაქვს `pricxvi` მისამართით მოთავსებული მნიშვნელობა. ამის შემდეგ, სრულდება ინკრემენტის ოპერაცია `pricxvi` მიმთითებლის მიმართ. შედეგად, მისამართი 4 ერთეულით იზრდება, რადგან `int` ტიპი 4 ბაიტს იკავებს მეხსიერებაში. მისამართის ახალი მნიშვნელობა ეკრანზე გამოგვაქვს. შემდეგ, მისამართს ემატება რიცხვი 2, ამიტომ ის გაიზრდება

2 x 4 = 8 ბაიტით.

მაგალითი.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
double wiladi = 125.75;
double *pwiladi = &wiladi;

cout << ((int)pwiladi) << endl;           // 1240740
cout << (*pwiladi) << endl;             // 125.75
pwiladi++;
cout << ((int)pwiladi) << endl;         // 1240748
pwiladi += 2;
cout << ((int)pwiladi) << endl;         // 1240764

system("pause");
return 0;
}
```

აქ, ეკრანზე გამოგვაქვს wiladi ცვლადის მისამართი, რომელიც pwiladi მიმთითებელშია მოთავსებული. შემდეგ გამოგვაქვს pwiladi მისამართით მოთავსებული მნიშვნელობა. ამის შემდეგ სრულდება ინკრემენტის ოპერაცია pwiladi მიმთითებლის მიმართ. შედეგად, მისამართი 8 ერთეულით იზრდება, რადგან double ტიპი 8 ბაიტს იკავებს მეხსიერებაში. მისამართის ახალი მნიშვნელობა ეკრანზე გამოგვაქვს. შემდეგ, მისამართს ემატება რიცხვი 2, ამიტომ ის გაიზრდება $2 \times 8 = 16$ ბაიტით.

მაგალითი.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
char simbolo = 'R';
char *pchar = &simbolo;

cout << ((int)pchar) << endl;           // 1767519
cout << (*pchar) << endl;             // R
pchar++;
cout << ((int)pchar) << endl;         // 1767520
pchar += 2;
cout << ((int)pchar) << endl;         // 1767522

system("pause");
return 0;
}
```

აქ, ეკრანზე გამოგვაქვს simbolo ცვლადის მისამართი, რომელიც pchar მიმთითებელშია მოთავსებული. შემდეგ გამოგვაქვს pchar მისამართით მოთავსებული მნიშვნელობა. ამის შემდეგ სრულდება ინკრემენტის ოპერაცია pchar მიმთითებლის მიმართ. შედეგად, მისამართი 2 ერთეულით იზრდება, რადგან char ტიპი 2 ბაიტს იკავებს მეხსიერებაში. მისამართის ახალი მნიშვნელობა ეკრანზე გამოგვაქვს. შემდეგ, მისამართს ემატება რიცხვი 2, ამიტომ ის გაიზრდება $2 \times 2 = 4$ ბაიტით.

მიმთითებელი და მრავალგანზომილებიანი მასივი

გამოვაცხადოთ ორგანზომილებიანი masivi მასივი და pmasivi მიმთითებელი. მიმთითებელს მივანიჭოთ ამ მასივის პირველი ელემენტის მისამართი:

```
int masivi[5][3];
int *pmasivi;
pmasivi = &masivi[0][0];
```

შედეგად, მიმთითებელი მიმართავს მასივის პირველ ელემენტს. შეგვიძლია, მიმთითებელს მივანიჭოთ masivi მასივის პირველი სტრიქონის მისამართი:

```
pmasivi = masivi[0];
```

რადგან, მიმთითებლის ტიპია int*, ხოლო მასივის ტიპი - int[5][3], ამიტომ დაუშვებელია ასეთი მინიჭება:

```
pmasivi = masivi; // შეცდომა!
```

იმისათვის, რომ მიმთითებელში მოვათავსოთ ორგანზომილებიანი მასივის მისამართი, მისი ტიპი უნდა იყოს int*[3]:

```
int masivi[5][3];
int (*pmasivi)[3];
```

ამ შემთხვევაში, დასაშვებია მინიჭება pmasivi = masivi ;

```
int masivi[5][3];
int (*pmasivi)[3];
pmasivi = masivi;
```

ორგანზომილებიან masivi მასივის ელემენტთან მიმართვა ორი გზით შეიძლება:

1. მასივის სახელისა და ორი ინდექსის მითითებით - masivi[striqoni][sveti];
2. მასივის სახელის, როგორც როგორც მიმთითებლის გამოყენებით - (*(masivi + striqoni) + sveti).

ორივე გზა ეკვივალენტურია. რაც შეეხება, მეორე ჩანაწერს, აქ *(masivi + striqoni) არის striqoni სტრიქონის პირველი ელემენტის მისამართი, ხოლო *(masivi + striqoni) + sveti კი - striqoni სტრიქონის ელემენტი, რომლის წანაცვლებაა sveti. ამიტომ, (*(masivi + striqoni) + sveti) გამოსახულება მიმართავს masivi მასივის კონკრეტულ ელემენტს.

მიმართვა

მიმართვა (reference, ссылка) არის სხვა ცვლადის *ფსევდონიმი*. მას აქვს სახელი, რომელიც შეგვიძლია ცვლადის სახელის ნაცვლად გამოვიყენოთ. მიმართვის გამოცხადებისას უნდა მივუთითოთ იმ ცვლადის სახელი, რომლისთვისაც ის იქმნება. არ შეიძლება მიმართვის შეცვლა იმ მიზნით, რომ მან სხვა ცვლადს მიმართოს. მიმართვის გამოცხადებისათვის & სიმბოლო გამოიყენება. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```

int _tmain(int argc, _TCHAR* argv[])
{
// მიმართვასთან მუშაობა
int ricxvi = 5;
int &mimartva_ricxvi = ricxvi; // მიმართვის გამოცხადება

cout << mimartva_ricxvi<< endl; // 5
cout << ricxvi<< endl; // 5
mimartva_ricxvi *= 2;
cout << mimartva_ricxvi<< endl; // 10
cout << ricxvi<< endl; // 10

system("pause");
return 0;
}

```

აქ mimartva_ricxvi არის მიმართვა, რომელიც ricxvi მთელი ტიპის ცვლადის ფსევდონიმია. ამ გამოცხადების შემდეგ მიმართვა შეგვიძლია გამოვიყენოთ ცვლადის მაგივრად.

მიმართვა შეგვიძლია გამოვიყენოთ წილად რიცხვთან სამუშაოდ.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
double ricxvi = 5.5;
double &mimartva_ricxvi = ricxvi; // მიმართვის გამოცხადება

cout << mimartva_ricxvi<< endl; // 5.5
cout << ricxvi<< endl; // 5.5
mimartva_ricxvi *= 3;
cout << mimartva_ricxvi<< endl; // 16.5
cout << ricxvi<< endl; // 16.5

system("pause");
return 0;
}

```

მიმართვა შეგვიძლია გამოვიყენოთ სიმბოლოსთან სამუშაოდ.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
char simbolo = 'R';
char &mimartva_simbolo = simbolo; // მიმართვის გამოცხადება

```

```

cout << mimartva_simbolo<< endl;           // R
cout << simbolo<< endl;                     // R
mimartva_simbolo = 'S';
cout << mimartva_simbolo<< endl;           // S
cout << simbolo<< endl;                     // S

```

```

system("pause");
return 0;
}

```

მიმართვა შეგვიძლია გამოვიყენოთ სტრიქონთან სამუშაოდ.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{

```

```

string striqoni = "Romani";
string &mimartva_striqoni = striqoni;           // მიმართვის გამოცხადება

```

```

cout << mimartva_striqoni << endl;           // Romani
cout << striqoni << endl;                     // Romani
mimartva_striqoni = "Samkharadze";
cout << mimartva_striqoni << endl;           // Samkharadze
cout << striqoni << endl;                     // Samkharadze

```

```

system("pause");
return 0;
}

```

აქ იმართვა გამოიყენება სტრიქონთან სამუშაოდ.

sizeof ოპერაცია

sizeof ოპერაციის შესრულებით ვღებულობთ size_t ტიპის მთელ მნიშვნელობას. ის მიუთითებს ბაიტების რაოდენობას, რომელსაც მისი ოპერანდი იკავებს. size_t ტიპი განსაზღვრულია სტანდარტულ ბიბლიოთეკაში და ეფუძნება unsigned int საბაზო ტიპს.

მოცემულ მაგალითში გაიცემა ricxvi ცვლადის ზომა:

```

int ricxvi;
cout << (sizeof ricxvi) << endl;

```

შედეგი იქნება 4, რადგან ricxvi არის int ტიპის ცვლადი.

sizeof ოპერაცია შეგვიძლია გამოვიყენოთ მასივის ელემენტის ან მთლიანად მასივის მიმართ. როდესაც მას ვიყენებთ მასივის სახელის მიმართ, მაშინ გაიცემა ბაიტების რაოდენობა, რომელსაც მთელი მასივი იკავებს. როდესაც მას ვიყენებთ მასივის ელემენტის მიმართ, მაშინ გაიცემა მოცემული ელემენტის მიერ დაკავებული ბაიტების რაოდენობა. მაგალითი:

```

#include "stdafx.h"

```

```

#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//      sizeof ოპერაციასთან მუშაობა
double masivi [ ] = { 1, 2, 3, 4, 5 };

cout << "mteli masivis zoma = " << (sizeof masivi) << endl;           //      40
cout << "masivis mesame elementis zoma = " << (sizeof masivi[2]) << endl; //      8
cout << "masivshi elementebis raodenoba = "
    << ( (sizeof masivi) / (sizeof masivi[0]) ) << endl;           //      5

system("pause");
return 0;
}

    sizeof ოპერაცია შეგვიძლია გამოვიყენოთ ტიპის მიმართაც:
size_t zoma = sizeof(double);
cout << zoma << endl;           //      zoma = 8

```

size() ფუნქცია

სტრიქონში სიმბოლოების რაოდენობის მისაღებად შეგვიძლია size() ფუნქცია გამოვიყენოთ. მოყვანილი პროგრამით ხდება size() ფუნქციასთან მუშაობის დემონსტრაცია:

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
int raodenoba;
string striqoni = "Romani";

raodenoba = striqoni.size();
cout << "raodenoba = " << raodenoba << endl;

system("pause");
return 0;
}

```

მეხსიერების დინამიკური გამოყოფა

პროგრამის შესრულებისას ხშირად წამოიჭრება სხვადასხვა ტიპის ცვლადისთვის მეხსიერების უბნის დინამიკურად გამოყოფის აუცილებლობა. რადგან, დინამიკურად განაწილებული ცვლადები არ შეიძლება განისაზღვროს პროგრამის კომპილაციის დროს, ამიტომ მათ არ აქვთ სახელები პროგრამის საწყის კოდში. დინამიკურად განაწილებული

ცვლადების შექმნისას მათი იდენტიფიცირება ხდება მეხსიერების მისამართებით, რომლებიც მიმთითებლებში ინახება.

მეხსიერების გროვა

ხშირად, პროგრამის შესრულებისას კომპიუტერის ოპერატიული მეხსიერების ნაწილი თავისუფალია. ამ გამოუყენებელ მეხსიერებას C++ სისტემაში *გროვა* ან *თავისუფალი სათავსო* ეწოდება. ამ გროვაში შეგვიძლია გამოვყოთ მეხსიერება მოცემული ტიპის ახალი ცვლადისთვის. ამ მიზნით გამოიყენება `new` ოპერატორი. ის, აგრეთვე გასცემს გამოყოფილი უბნის მისამართს. `new` ოპერატორის მიერ გამოყოფილ მეხსიერების უბანს ათავისუფლებს `delete` ოპერატორი.

გროვაში შეგვიძლია რომელიმე ცვლადისთვის გამოვყოთ მეხსიერების უბანი. შემდგომ, როდესაც ეს ცვლადი საჭირო აღარ იქნება, შესაბამისი უბანი შეგვიძლია გავათავისუფლოთ და გროვას დავუბრუნოთ. შედეგად, შესაძლებელი გახდება ამ უბნის გამოყენება ამავე პროგრამის მიერ, ოღონდ უკვე სხვა ცვლადისთვის.

გროვაში მეხსიერების გამოყოფის აუცილებლობა დგება ყოველთვის, როდესაც საჭიროა პროგრამის შესრულებისას განსაზღვრული ელემენტებისთვის მეხსიერების გამოყოფა. მაგალითად, ასეთი აუცილებლობა დგება მომხმარებლის მიერ სტრიქონების შეტანისას. რადგან ჩვენ წინასწარ არ ვიცით თითოეული სტრიქონის სიგრძე, ამიტომ მათთვის მეხსიერების გამოყოფა უმჯობესია `new` ოპერატორის გამოყენებით.

მეხსიერების დინამიკური განაწილების შესაძლებლობა იძლევა მეხსიერების ეფექტურად გამოყენების საშუალებას. ამ დროს, შეგვიძლია მეხსიერება გამოვყოთ მხოლოდ საჭიროების შემთხვევაში და გავათავისუფლოთ მაშინ, როდესაც ის აღარ გვჭირდება. მეხსიერების გათავისუფლებული უბანი შეგვიძლია სხვა ობიექტების მოსათავსებლად გამოვიყენოთ.

`new` და `delete` ოპერატორები

დავუშვათ, `int` ტიპის ცვლადისთვის გვინდა მეხსიერების გამოყოფა. ჩვენ შეგვიძლია გამოვაცხადოთ `int` ტიპზე მიმთითებელი და შემდეგ მოვითხოვოთ მისთვის მეხსიერების გამოყოფა პროგრამის შესრულების დროს. ეს ხორციელდება `new` ოპერატორის გამოყენებით:

```
int *pcvladi = NULL;           // მიმთითებელი ინიციალიზებული როგორც NULL
pcvladi = new int;           // მეხსიერების მოთხოვნა int ტიპის ცვლადისთვის
*pcvladi = 55;
```

აქ `new` ოპერატორი გასცემს გამოყოფილი მეხსიერების მისამართს, რომელიც `pcvladi` ცვლადს მიენიჭება.

შეგვიძლია ცვლადის ინიციალიზება მისთვის მეხსიერების გამოყოფისთანავე:

```
pcvladi = new int(77);
```

როდესაც დინამიკურად განაწილებული ცვლადი საჭირო აღარ არის, მაშინ შეგვიძლია მის მიერ დაკავებული მეხსიერების გათავისუფლება:

```
delete pcvladi;
```

ეს საშუალებას გვაძლევს გამოთავისუფლებული მეხსიერება გამოვიყენოთ მასში სხვა ცვლადების მოსათავსებლად. თუ `delete` ოპერაცია არ შევასრულებთ და შემდგომ `pcvladi` ცვლადს სხვა მისამართი მივანიჭებთ, შეუძლებელი გახდება მეხსიერების გათავისუფლება ან მასში მოთავსებული ცვლადის გამოყენება, რადგან მას ვეღარ მივმართავთ. ამას *მეხსიერების გაჟონვა* (memory leak) ეწოდება.

მეხსიერების დინამიკური განაწილება მასივებისთვის

char ტიპის მასივისთვის დინამიკურად გამოვყოთ მეხსიერება:

```
char *pstriqoni;
```

```
pstriqoni = new char[100];
```

აქ მეხსიერება 100-სიმბოლოიანი მასივისთვის გამოიყოფა. მისი მისამართი pstriqoni მიმთითებელში იწახება.

ასეთნაირად გამოყოფილი მეხსიერების გასათავისუფლებლად და გროვაში დასაბრუნებლად delete ოპერატორი უნდა გამოვიყენოთ:

```
delete [ ] pstriqoni;
```

კვადრატული ფრჩხილები მიუთითებს, რომ ხდება მასივის წაშლა. delete ოპერაციის შემდეგ pstriqoni მიმთითებელი შეიძლება შეიცავდეს მეხსიერების რომელიმე შემთხვევითი უბნის მისამართს, რომლის გამოყენება არ შეიძლება. ამიტომ, მიმთითებელს უნდა მივანიჭოთ ნულოვანი მნიშვნელობა:

```
pstriqoni = 0;
```

დავუშვათ, გვაქვს ორგანზომილებიანი მასივი:

```
int masivi[5][3];
```

```
int (*pmasivi)[3];
```

მაშინ მისთვის შეგვიძლია გამოვყოთ მეხსიერება:

```
pmasivi = new int[5][3];
```

ანალოგიურად ვიქცევით სამგანზომილებიანი მასივისთვის:

```
int masivi[5][3][4];
```

```
int (*pmasivi)[3][4];
```

```
pmasivi = new int[5][3][4];
```

ნებისმიერი განზომილების მასივის წასაშლელად გამოიყენება delete ოპერაცია.

შემთხვევითი რიცხვების გენერატორი

C++ ენაში განსაზღვრულია rand() ფუნქცია, რომელიც შემთხვევით რიცხვებს გასცემს. ქვემოთ მოყვანილი პროგრამით ხდება ერთგანზომილებიანი მასივის შემთხვევითი რიცხვებით შევსების დემონსტრირება დიაპაზონში [5 - 15].

```
// ერთგანზომილებიანი მასივის შევსება შემთხვევითი რიცხვებით დიაპაზონში [5 - 15]
```

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
#include "ctime"
```

```
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
int masivi[10];
```

```
int indexi;
```

```
for ( indexi = 0; indexi < 10; indexi++ )
```

```
    masivi[indexi] = rand() % (16 - 5) + 5;
```

```
for ( indexi = 0; indexi < 10; indexi++ )
```

```
    cout << masivi[indexi] << " ";
```

```
cout << endl;
```

```

system("pause");
return 0;
}

```

თუ ამ პროგრამას სხვადასხვა დროს შევასრულებთ დავინახავთ, რომ მასივში ყოველთვის ერთი და იგივე შემთხვევითი რიცხვები ჩაიწერება. იმისათვის, რომ პროგრამის ყოველი შესრულებისას სხვადასხვა შემთხვევითი რიცხვები მივიღოთ, პროგრამის კოდის დასაწყისში უნდა მოვათავსოთ `srand(time(NULL));` ფუნქცია:

// **ორგანზომილებიანი მასივის შევსება შემთხვევითი რიცხვებით დიაპაზონში [5 - 15]**

```

#include "stdafx.h"
#include "iostream"
#include "ctime"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
srand(time(NULL));
int masivi[10];
int indexi;

for ( indexi = 0; indexi < 10; indexi++ )
    masivi[indexi] = rand() % (16 - 5) + 5;
for ( indexi = 0; indexi < 10; indexi++ )
    cout << masivi[indexi] << " ";
cout << endl;

system("pause");
return 0;
}

```

ქვემოთ მოყვანილი პროგრამით ხდება ორგანზომილებიანი მასივის შემთხვევითი რიცხვებით შევსების დემონსტრირება.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ორგანზომილებიანი მასივის შევსება შემთხვევითი რიცხვებით დიაპაზონში [30 - 70]
srand(time(NULL));
int masivi[3][3];
int striqoni, sveti;

```

```

for ( int striqoni = 0; striqoni < 3; striqoni++ )
    for ( int sveti = 0; sveti < 3; sveti++ )
        masivi[striqoni][sveti] = rand() % (71 - 30) + 30;

```

```

for ( int striqoni = 0; striqoni < 3; striqoni++ )

```

```

{
    for ( int sveti = 0; sveti < 3; sveti++ )
        cout << masivi[striqoni][sveti]<< " ";
    cout << endl;
}

system("pause");
return 0;
}

```

ბიტობრივი ოპერატორები

ბიტობრივი (თანრიგობრივი) ოპერატორის ოპერანდები შეიძლება იყოს მთელი რიცხვები, აგრეთვე, bool ტიპის მნიშვნელობები. ეს ოპერატორები არ შეიძლება გამოვიყენოთ, float ან double ტიპის მნიშვნელობების, აგრეთვე, კლასის ობიექტების მიმართ. ასეთ ოპერატორებს ბიტობრივი ეწოდება იმიტომ, რომ ისინი გამოიყენება მთელი რიცხვების ბიტების შესამოწმებლად, დასაყენებლად (1 მდგომარეობაში გადაყვანა) და ძვრისთვის. ბიტობრივი ოპერაციები ხშირად გამოიყენება სისტემურ დონეზე დაპროგრამებისთვის, მაგალითად, როდესაც საჭიროა ინფორმაციის მიღება მოწყობილობის მდგომარეობის შესახებ და ა.შ. ბიტობრივი ოპერაციები მოცემულია 4.1 ცხრილში.

ცხრილი 4.1. ბიტობრივი ოპერაციები

ოპერატორი	აღწერა
&	ბიტობრივი ოპერატორი AND
	ბიტობრივი ოპერატორი OR
^	ბიტობრივი ოპერატორი XOR
>>	მარჯვნივ ძვრის ოპერატორი
<<	მარცხნივ ძვრის ოპერატორი
~	უნარული ოპერატორი NOT

&, |, ^ და ~ ბიტობრივი ოპერატორები

&, |, ^ და ~ ბიტობრივი ოპერატორები ისეთივე ოპერაციებს ასრულებენ, როგორც მათი ბულის ეკვივალენტები. განსხვავება ისაა, რომ ბიტობრივი ოპერატორები ოპერანდებზე ზემოქმედებენ ბიტების დონეზე. 4.2 ცხრილში მოცემულია თითოეული ოპერაციის შესრულების შედეგები.

ცხრილი 4.2. ბიტობრივი ოპერაციები

b1	b2	b1&b2	b1 b2	b1 ^ b2	~b1
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

ბიტობრივი & ოპერაცია შეგვიძლია გამოვიყენოთ როგორც ბიტის განულების (ჩამოგდების) საშუალება. ეს იმას ნიშნავს, რომ თუ რომელიმე ოპერანდის ერთ-ერთ ბიტს აქვს მნიშვნელობა 0, მაშინ შედეგის შესაბამის ბიტსაც ექნება მნიშვნელობა 0. მაგალითად,

11010011
&
10101010
10000010

მოცემულ პროგრამით ნაჩვენებია & ოპერატორის გამოყენების მაგალითი. პროგრამა ქვედა რეგისტრის სიმბოლოებს გარდაქმნის ზედა რეგისტრის სიმბოლოებად მეექვსე ბიტის განულების გზით. Unicode/ASCII სიმბოლოების სტანდარტულ ნაკრებში ქვედა რეგისტრის სიმბოლოების მნიშვნელობა მეტია ზედა რეგისტრის სიმბოლოების შესაბამის მნიშვნელობაზე 32 ერთეულით ($32_{10}=00100000_2$). შედეგად, ქვედა რეგისტრის სიმბოლოების გადასაყვანად ზედა რეგისტრში საკმარისია მეექვსე ბიტის განულება. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ქვედა რეგისტრის სიმბოლოების გარდაქმნა ზედა რეგისტრის სიმბოლოებად
char simbolo;

for ( int cvladi = 0; cvladi < 10; cvladi++ )
{
    simbolo = (char) ( 'a' + cvladi );
    cout << simbolo << " - ";
    simbolo = (char) ( simbolo & 65503 );
    cout << simbolo << endl;
}

system("pause");
return 0;
}
```

რიცხვი 65503 ორბაიტანია და ორობით სისტემაში ასე გამოისახება 1111 1111 1101 1111. & ოპერაცია უცვლელად ტოვებს simbolo ცვლადის ყველა ბიტს მეექვსე ბიტის გარდა, რომელიც განულებდა. & ოპერაციის გამოყენება შეგვიძლია მაშინ, როდესაც საჭიროა განისაზღვროს ბიტი დაყენებულია (1) თუ ჩამოგდებული (0). მაგალითად, ქვემოთ მოცემული if ოპერატორით მოწმდება status ცვლადის მეოთხე ბიტის მნიშვნელობა:

```
if ( status & 8 == true ) cout << L"მეოთხე ბიტი დაყენებულია";
```

რიცხვი 8 გამოიყენება იმიტომ, რომ მის ორობით წარმოდგენაში დაყენებულია მხოლოდ მეოთხე ბიტი (0000 1000). შედეგად, if ოპერატორი true შედეგს მხოლოდ მაშინ გასცემს, როდესაც status ცვლადის მნიშვნელობაში მეოთხე ბიტი იქნება დაყენებული. ქვემოთ მოცემული პროგრამით ასეთი მიდგომა გამოიყენება ეკრანზე int ტიპის მნიშვნელობის ორობითი წარმოდგენის გამოსატანად.

```
// ათობითი რიცხვის გარდაქმნა ორობით რიცხვად
```

```

#include "stdafx.h"
#include "iostream"
#include "ctime"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
int cvladi, ricxvi;

cin >> ricxvi;
for ( cvladi = 128; cvladi > 0; cvladi /= 2 )
{
    if ( ( ricxvi & cvladi ) != 0 ) cout << "1";
        else cout << "0";
}
cout << endl;

system("pause");
return 0;
}

```

for ციკლში & ოპერატორის გამოყენებით მოწმდება ricxvi ცვლადის თითოეული ბიტი. თუ ის დაყენებულია, მაშინ ეკრანზე გამოჩნდება 1, თუ ჩამოგდებულია - 0.

& ოპერატორის საწინააღმდეგოდ | ოპერატორი შეგვიძლია გამოვიყენოთ ბიტების დასაყენებლად. თუ ერთ-ერთ ოპერანდში ბიტი დაყენებულია, მაშინ შედეგში შესაბამისი ბიტი იქნება დაყენებული. მაგალითად,

```

11010011
  |
10101010
11111011

```

ზედა რეგისტრის სიმბოლოების გარდასაქმნელად ქვედა რეგისტრის სიმბოლოებად გამოვიყენოთ | ოპერატორი, როგორც ეს ქვემოთაა ნაჩვენები:

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ზედა რეგისტრის სიმბოლოების გარდაქმნა ქვედა რეგისტრის სიმბოლოებად
char simbolo;

for ( int cvladi = 0; cvladi < 10; cvladi++ )
{
    simbolo = (char) ( 'A' + cvladi );
    cout << simbolo << " - ";
}
// ამ ოპერატორის შესრულება აყენებს მეექვსე ბიტს

```

```
// შედეგად simbolo ცვლადში ჩაიწერება ქვედა რეგისტრის სიმბოლო
simbolo = (char) ( simbolo | 32 );
cout << simbolo << endl;
}

system("pause");
return 0;
}
```

| ოპერაცია გამოიყენება თითოეული სიმბოლოს მნიშვნელობისა და 32 მნიშვნელობის მიმართ, რომელიც ორობით სისტემაში ასე გამოისახება 0000 0000 0010 0000. ამ ორობით წარმოდგენაში დაყენებულია მხოლოდ მეექვსე ბიტი. თუ | ოპერაციის ერთი ოპერანდია რიცხვი 32, მეორე ოპერანდი კი - ნებისმიერი რიცხვი, მაშინ შედეგად მივიღებთ მნიშვნელობას, რომელშიც დაყენებული იქნება მეექვსე ბიტი. დანარჩენი ბიტები დარჩება უცვლელი.

^ ოპერაციის შესრულების შედეგად ბიტის დაყენება მოხდება მხოლოდ მაშინ, როდესაც ორივე ბიტი განსხვავებულია. მაგალითად,

```
11010011
  ^
10101010
11111011
```

^ ოპერატორს საკმაოდ საინტერესო თვისება აქვს, რომელიც საშუალებას გვაძლევს ის გამოვიყენოთ შეტყობინების ან რაიმე ინფორმაციის დასაშიფრად. თუ ამ ოპერატორს გამოვიყენებთ x და y ცვლადების მიმართ და შემდეგ მიღებული შედეგისა და y ცვლადის მიმართ, მივიღებთ x მნიშვნელობას, ე.ი.

```
R1 = x ^ y;
R2 = R1 ^ y;
```

ოპერატორების შესრულების შედეგად R2 ცვლადს მიენიჭება x ცვლადის მნიშვნელობა. უნარული ~ ოპერაცია ცვლის ოპერანდის ყველა ბიტის მნიშვნელობას საწინააღმდეგოთი. მაგალითად, თუ A = 1001 0011, მაშინ ~A ოპერაციის შედეგი იქნება 0110 1100.

ქვემოთ მოცემულ პროგრამაში ხდება ~ ოპერატორის გამოყენების დემონსტრირება. ეკრანზე ჯერ ჩნდება რიცხვი 35-ის (0010 0011) ორობითი წარმოდგენა, შემდეგ კი ამ რიცხვის მიმართ ~ ოპერაციის გამოყენების შედეგი (1101 1100).

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ~ ოპერატორის მუშაობის დემონსტრირება
int ricxvi;

cin >> ricxvi;
cout << "atobiti ricxvi - " << ricxvi << endl;
// ricxvi ცვლადის ორობითი წარმოდგენის ეკრანზე გამოტანა
cout << "orobiTi ricxvi:" << endl;
```

```

for ( int cvladi = 128; cvladi > 0; cvladi /= 2 )
    if ( ( ricxvi & cvladi ) != 0 ) cout << "1 ";
    else cout << "0 ";
cout << endl;
// ricxvi ცვლადის ორობითი წარმოდგენის ინვერსია და ეკრანზე გამოტანა
cout << "orobiti ricxvis inversia:" << endl;
ricxvi = ~ricxvi;
for ( int cvladi = 128; cvladi > 0; cvladi /= 2 )
    if ( ( ricxvi & cvladi ) != 0 ) cout << "1 ";
    else cout << "0 ";
cout << endl;
//
    system("pause");
    return 0;
}
ყველა ორობითი ბიტობრივი ოპერატორი შეგვიძლია ჩავწეროთ შედგენილი ბიტობრივი
ოპერატორების სახით. მაგალითად,
x ^= 127;
y &= 63;
z |= 31;

```

ძვრის ოპერატორები

არსებობს ოპერანდების მარცხნივ და მარჯვნივ ძვრის ოპერატორები. მათი სინტაქსია:

ცვლადის_სახელი << ბიტების_რაოდენობა

ცვლადის_სახელი >> ბიტების_რაოდენობა

მარცხნივ ძვრა (<<) იწვევს ყველა ბიტის გადაადგილებას ერთი ბიტით მარცხნივ, ამასთან, თავისუფლდება უმცროსი ბიტები, რომლებიც ნულებით ივსება, ხოლო უფროსი ბიტების შესაბამისი რაოდენობა იკარგება. მარჯვნივ ძვრა (>>) იწვევს ყველა ბიტის ერთი ბიტით მარჯვნივ გადაადგილებას, ამასთან, თუ მარჯვნივ იძვრის უნიშნო მნიშვნელობის ბიტები, მაშინ უფროსი თანრიგები ნულებით შეივსება და უმცროსი თანრიგების შესაბამისი რაოდენობა იკარგება. ნიშნის მნიშვნელობის მარჯვნივ ძვრისას ხდება ნიშნის ბიტის შენახვა. ნიშნის დადებითი რიცხვისთვის უფროსი ბიტი არის 0, უარყოფითი რიცხვისთვის - 1.

ქვემოთ მოცემული პროგრამით ნაჩვენებია ძვრის ოპერაციების გამოყენების მაგალითი.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ძვრის ოპერატორებთან მუშაობის დემონსტრირება
int ricxvi = 1;

// ricxvi ცვლადის ორობითი წარმოდგენის ეკრანზე გამოტანა
for ( int i = 0; i < 8; i++ )
{
    for ( int j = 128; j > 0; j /= 2 )
        if ( ( ricxvi & j ) != 0 ) cout << "1";

```

```

        else cout << "0";
    cout << endl;
    ricxvi = ricxvi << 1;           // მვრა მარცხნივ ერთი ბიტით
}
ricxvi = 128;
// ricxvi ცვლადის ორობითი წარმოდგენის ეკრანზე გამოტანა
for ( int i = 0; i < 8; i++ )
{
    for ( int j = 128; j > 0; j /= 2 )
        if ( ( ricxvi & j ) != 0 ) cout << "1";
        else cout << "0";
    cout << endl;
    ricxvi = ricxvi >> 1;         // მვრა მარჯვნივ ერთი ბიტით
}

system("pause");
return 0;
}

```

asm საკვანძო სიტყვა

C++ ენაზე პროგრამის შედგენისას მისი ფრაგმენტი შეგვიძლია ასემბლერ ენაზე დაწეროთ. ამისთვის გამოიყენება asm საკვანძო სიტყვა. მისი სინტაქსია:

```

__asm ასემბლერის_ბრძანება [ ; ]
__asm { ასემბლერის_ბრძანების_სია } [ ; ]

```

C++ ენის კოდში შეგვიძლია მოვათავსოთ ასემბლერ ენაზე დაწერილი პროგრამა. მოყვანილი პროგრამით ხდება ამის დემონსტრირება:

```

#include "stdafx.h"
#include <iostream>
using namespace std;

int _tmain(int argc, _TCHAR* argv[ ])
{
    short ricxvi = 5;
    // ასემბლერ ენაზე შედგენილი კოდი
    __asm
    {
        mov ax, ricxvi
        mov bx, 10
        add ax, bx
        mov ricxvi, ax
    }
    cout << "jami = " << ricxvi;           // შედეგის გამოტანა ეკრანზე

    system("pause");
}

```



```
return 0;
}
```

C++ პროგრამაში შეგვიძლია ჩავრთოთ ასემბლერ ენის ერთი ბრძანება:

```
#include "stdafx.h"
#include <iostream>
#include <fstream>
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[ ])
{
    short ricxvi;
    // ასემბლერ ენის ბრძანება
    __asm mov ricxvi, 55
    cout << ricxvi;

    system("pause");
    return 0;
}
```

თავი 5. სტრიქონები

სტრიქონი

C++ ენაში სტრიქონი გამოიყენება სიმბოლოების მიმდევრობის შესანახად. ერთი სიმბოლო 1 ბაიტს ანუ 8 ბიტს იკავებს. ის string სიტყვის საშუალებით აღიწერება. მოყვანილ პროგრამაში ხდება ორი სტრიქონის გამოცხადება: s1 და s2. s1 სტრიქონს კლავიატურიდან ვანიჭებთ მნიშვნელობას, s2 სტრიქონს კი უშუალოდ ვანიჭებთ "C++" მნიშვნელობას.

```
#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    string s1, s2;

    cin >> s1;
    s2 = "C++";
    cout << s1 << " " << s2 << endl;

    system("pause");
    return 0;
}
```

ისევე როგორც მასივში, აქაც პირველი ელემენტის (სიმბოლოს) ინდექსია 0. ინდექსი გამოიყენება მხოლოდ სიმბოლოს მისაღებად:

```
string striqoni = "computer";
cout << striqoni[2]<< endl;
```

ორივე ოპერატორის შესრულების შედეგად ეკრანზე გამოჩნდება ასო "m".

სტრიქონი შეიძლება შეიცავდეს მმართველ სიმბოლოებს:

```
string striqoni = "C++\nLanguage";
cout << striqoni;
```

ამ სტრიქონების შესრულების შედეგად ეკრანზე პირველ სტრიქონში გამოჩნდება "C++", მეორე სტრიქონში კი - "Language".

როგორც ვიცით, endl საკვანძო სიტყვა კურსორს ათავსებს მომდევნო სტრიქონის დასაწყისში. იგივე შეგვიძლია გავაკეთოთ "\n" მმართველი სიმბოლოების გამოყენებით. ქვემოთ მოყვანილი კოდი ერთნაირ შედეგს გასცემს:

```
cout << "C++" << "\n" << "Language" << endl;
cout << "C++" << endl << "Language" << endl;
```

ცხრილი 5.1. სტრიქონებთან სამუშაო ფუნქციები

ფუნქცია	აღწერა
int find(string სტრიქონი)	გასცემს ნაპოვნი სტრიქონის ინდექსს. თუ სტრიქონი ვერ მოიძებნა, მაშინ გაიცემა -1.
int compare(string სტრიქონი)	გამომდახებელ სტრიქონს ადარებს მითითებულ სტრიქონს. გაიცემა 0, თუ სტრიქონები ტოლია; 1 - თუ პირველი სტრიქონი მეორეზე მეტია; -1 - თუ პირველი სტრიქონი მეორეზე ნაკლებია.
string append(string სტრიქონი)	გამომდახებელ სტრიქონს ბოლოში დაუმატებს მითითებულ სტრიქონს
string insert(int საწყისი_ინდექსი, string სტრიქონი)	გამომდახებელ სტრიქონში ინდექსი პოზიციიდან დაწყებული ჩასვამს მითითებულ სტრიქონს
string erase(int საწყისი_ინდექსი, int რაოდენობა)	გამომდახებელ სტრიქონში ინდექსი პოზიციიდან დაწყებული წაშლის მითითებული რაოდენობა-ის სიმბოლოს
string replace(int საწყისი_ინდექსი, int რაოდენობა, string სტრიქონი)	გამომდახებელ სტრიქონში საწყისი_ინდექსი პოზიციიდან დაწყებული მითითებული რაოდენობა-ის სიმბოლოს შეცვლის სტრიქონით
string substr(int საწყისი_ინდექსი, int რაოდენობა)	გამომდახებელი სტრიქონის საწყისი_ინდექსი პოზიციიდან გასცემს მითითებული რაოდენობა-ის სიმბოლოს

სტრიქონებთან სამუშაო ფუნქციები

განვიხილოთ სტრიქონებთან სამუშაო ზოგიერთი ფუნქცია (ცხრილი 5.1).

ქვემოთ მოცემული პროგრამით ხდება **length** ფუნქციის გამოყენების დემონსტრირება.

```
#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//    length ფუნქციებთან მუშაობის დემონსტრირება
    string str1 = "C++ is a language";
    cout << "სტრიქონის სიგრძე = " << str1.length() << endl;           // 17

    system("pause");
    return 0;
}
```

პროგრამის str1.length() გამოსახულებაში str1 არის გამომდახებელი სტრიქონის სახელი. მის მარჯვნივ წერტილის დასმის შემდეგ გაიხსნება სტრიქონებთან სამუშაო ფუნქციების სია. ვირჩევთ length() ფუნქციას. მისი შესრულების შედეგად გაიცემა str1 სტრიქონში სიმბოლოების რაოდენობა.

ქვემოთ მოცემული პროგრამით ხდება **insert** ფუნქციის გამოყენების დემონსტრირება.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//    insert ფუნქციებთან მუშაობის დემონსტრირება
    string str1 = "C++ is a language";
    string str2 = " good";
    string shedegi;

    shedegi = str1.insert(8, str2);
    cout << shedegi << endl;           // "C++ is a good language"

    system("pause");
    return 0;
}

```

ამ პროგრამაში insert(8, str2) ფუნქცია str1 სტრიქონს მე-9 პოზიციის წინ (რომლის ინდექსია 8) str2 სტრიქონს ჩაუმატებს. მიღებული ახალი სტრიქონი მიენიჭება shedegi ცვლადს. ქვემოთ მოცემული პროგრამით ხდება **replace** ფუნქციის გამოყენების დემონსტრირება.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//    replace ფუნქციებთან მუშაობის დემონსტრირება
    string str1 = "Java is a language";
    string str2 = "C++";
    string shedegi;

    shedegi = str1.replace(0, 4, str2);
    cout << shedegi << endl;           // "C++ is a language"

    system("pause");
    return 0;
}

```

აქ replace(0, 4, str2) ფუნქცია str1 სტრიქონში დაწყებული პირველი პოზიციიდან (რომლის ინდექსია 0) 4 სიმბოლოს str2 სტრიქონით შეცვლის. მიღებული ახალი სტრიქონი მიენიჭება shedegi ცვლადს.

ქვემოთ მოცემული პროგრამით ხდება **erase** ფუნქციის გამოყენების დემონსტრირება.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
//   erase ფუნქციებთან მუშაობის დემონსტრირება
string str1("C++ is a good language");
string shedegi;

shedegi = str1.erase(8, 5);
cout << shedegi << endl;           // "C++ is a language"

system("pause");
return 0;
}

```

ამ პროგრამაში `erase(8, 5)` ფუნქცია `str1` სტრიქონის მე-9 პოზიციიდან (რომლის ინდექსია 8) 5 სიმბოლოს წაშლის. მიღებული ახალი სტრიქონი მიენიჭება `shedegi` ცვლადს.

ქვემოთ მოცემული პროგრამით ხდება **substr** ფუნქციის გამოყენების დემონსტრირება.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//   substr ფუნქციებთან მუშაობის დემონსტრირება
string str1 = "C++ is a good language";
string shedegi;

shedegi = str1.substr(3, 10);
cout << shedegi << endl;           // "is a good"

system("pause");
return 0;
}

```

აქ `substr(3, 10)` ფუნქცია `str1` სტრიქონის მე-4 პოზიციიდან (რომლის ინდექსია 3) 10 სიმბოლოს გასცემს. მიღებული ახალი სტრიქონი მიენიჭება `shedegi` ცვლადს.

ქვემოთ მოცემული პროგრამით ხდება **append** ფუნქციის გამოყენების დემონსტრირება.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//   append ფუნქციებთან მუშაობის დემონსტრირება
string str1 = "C++ is a good";
string str2 = " language";
string shedegi;

shedegi = str1.append(str2);

```

```

cout << shedegi << endl;                                // "C++ is a good language"

system("pause");
return 0;
}

```

აქ `append(str2)` ფუნქცია `str1` სტრიქონს ბოლოში `str2` სტრიქონს დაუმატებს. მიღებული ახალი სტრიქონი მიენიჭება `shedegi` ცვლადს.

ქვემოთ მოცემული პროგრამით ხდება **find** ფუნქციის გამოყენების დემონსტრირება.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//   find ფუნქციებთან მუშაობის დემონსტრირება
string str1 = "C++ is a good language";
string str2 = "good";
int index;

index = str1.find(str2);                                // 9
cout << str1 << "\n" << str2 << " " << index << endl;

system("pause");
return 0;
}

```

აქ `find(str2)` ფუნქცია `str1` სტრიქონში შეასრულებს `str2` სტრიქონის ძებნას. `index` ცვლადში ჩაიწერება იმ პოზიციის ინდექსი, საიდანაც ნაპოვნი სტრიქონი იწყება.

ქვემოთ მოცემული პროგრამით ხდება **compare** ფუნქციის გამოყენების დემონსტრირება.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
//   compare ფუნქციებთან მუშაობის დემონსტრირება
string str1 = "C++ is a good language";
string str2 = "good";
int shedegi;

shedegi = str1.compare(str2);
cout << str1 << "\n" << str2 << endl;
switch ( shedegi )
{
    case -1: cout << "pirveli striqoni meoreze metia" << endl; break;
    case 0: cout << "striqonebi tolia" << endl; break;
    case 1: cout << "pirveli striqoni meoreze naklebia" << endl; break;
}
}

```

```

}

system("pause");
return 0;
}

```

აქ compare(str2) ფუნქცია str1 სტრიქონში შეასრულებს str2 სტრიქონის ძებნას. შედარების შედეგი shedegi ცვლადს მიენიჭება. შემდეგ ხდება ამ ცვლადის მნიშვნელობის ანალიზი switch ოპერატორის გამოყენებით.

ქვემოთ მოცემული პროგრამით ხდება სტრიქონების მინიჭების დემონსტრირება.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// სტრიქონების მინიჭების დემონსტრირება
string str1 = "C++ is a good";
string str2 = "language";

str1 = str2;
cout << "str1 = " << str1 << "\nstr2 = " << str2 << endl;           // "language language"

system("pause");
return 0;
}

```

ორი სტრიქონის შესადარებლად შეგვიძლია ==, >, <, >=, <=, != ოპერატორების გამოყენება.

მაგალითი.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ==, >, <, >=, <=, !=
string str1 = "C++ is a good";
string str2 = "language";

cout << "str1 = " << str1 << "\nstr2 = " << str2 << endl;
if (str1 == str2) cout << "striqonebi tolia" << endl;
else cout << "striqonebi toli ar aris" << endl;

system("pause");
return 0;
}

```

სტრიქონების მასივი

ქვემოთ მოცემული პროგრამით ხდება სტრიქონების ორი მასივის გამოცხადება. სრულდება s1 სტრიქონის ინიციალიზაცია. s2 სტრიქონს მნიშვნელობები cin ფუნქციის გამოყენებით ენიჭება.

```
#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    string s1[3] = { "C++", "C#", "Python" };
    string s2[3];
    int indexi;

    for ( indexi = 0; indexi < 3; indexi++ )
        cin >> s2[indexi];
    for ( indexi = 0; indexi < 3; indexi++ )
        cout << s2[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}
```


თავი 6. ინფორმაციის შეტანა-გამოტანა

შეტანა-გამოტანის ნაკადების კლასები

C++ პროგრამები მონაცემების შეტანა-გამოტანას ნაკადების საშუალებით ახდენენ. ნაკადში იგულისხმება მონაცემების მიმდევრობა. ნაკადი ფიზიკურ მოწყობილობას შეტანა-გამოტანის სისტემის საშუალებით უკავშირდება.

C++ ენაში შეტანა-გამოტანის ოპერაციები ყველაზე დაბალ დონეზე - ბაიტებზე ოპერირებენ, რადგან შეტანა-გამოტანის მოწყობილობების უმრავლესობა არის ბაიტ-ორიენტირებული.

არსებობს სიმბოლოების ASCII და Unicode ნაკრები. ASCII სიმბოლოს ზომაა 1 ბაიტი, ხოლო Unicode სიმბოლოს ზომა კი - 2 ბაიტი. ჩვენ განვიხილავთ ASCII სიმბოლოებს.

C++ ენაში არსებობს ფაილებთან სამუშაო ofstream და ifstream ტიპები. ofstream ტიპი გამოიყენება ფაილში მონაცემების ჩაწერის დროს, ifstream ტიპი კი - წაკითხვის დროს. ისინი განსაზღვრულია "fstream" ბიბლიოთეკაში. თუ ვაპირებთ ფაილებთან მუშაობას, მაშინ პროგრამის ფაილის დასაწყისში უნდა დავუმატოთ #include "fstream" დირექტივა.

ფაილში სხვადასხვა ტიპის მონაცემების ჩაწერა და წაკითხვა

ofstream ტიპის საშუალებით ხდება ცვლადის გამოცხადება, რომელიც ფაილს ჩაწერის რეჟიმში უკავშირდება. მაგალითად,

```
ofstream out_file("file_1.dat");
```

სტრიქონში ხდება out_file ცვლადის გამოცხადება, რომელიც file_1.dat ფაილს უკავშირდება. ამ დროს, თუ ფაილი არ არსებობს, მაშინ ის შეიქმნება. თუ ფაილი არსებობს, მაშინ ის წაიშლება და ხელახლა შეიქმნება. ფაილის შექმნა სრულდება მიმდინარე კატალოგში. მიმდინარე კატალოგი, რომელიც შეიცავს პროექტის სახელის მქონე ფაილს, რომლის გაფართოებაცაა .cpp .

შეგვიძლია მივუთითოთ გზა ფაილისკენ, მაგალითად

```
ofstream out_file("D:\\data\\file_1.dat");
```

ამ შემთხვევაში file_1.dat ფაილი შეიქმნება D:\data კატალოგში.

ifstream ტიპის საშუალებით ხდება ცვლადის გამოცხადება, რომელიც ფაილს უკავშირდება წაკითხვის რეჟიმში. მაგალითად,

```
ifstream in_file("file_1.dat");
```

სტრიქონში ხდება in_file ცვლადის გამოცხადება, რომელიც file_1.dat ფაილს უკავშირდება. ამ დროს, თუ ფაილი არ არსებობს, მაშინ გაიცემა შეტყობინება შეცდომის შესახებ. თუ ფაილი არსებობს, მაშინ ის გაიხსნება. ფაილის ძებნა სრულდება მიმდინარე კატალოგში. შეგვიძლია მივუთითოთ გზა ფაილისკენ, მაგალითად

```
ifstream in_file("D:\\data\\file_1.dat");
```

ამ შემთხვევაში file_1.dat ფაილი უნდა არსებობდეს D:\data კატალოგში.

მოყვანილი პროგრამით ჯერ ხდება ფაილში მთელი რიცხვების ჩაწერა და შემდეგ წაკითხვა.

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
#include "fstream"
```

```
using namespace std;
```

```

int _tmain(int argc, _TCHAR* argv[])
{
// ფაილში მთელი რიცხვების ჩაწერა და წაკითხვა
int ricxvi1 = 10, ricxvi2 = 25, ricxvi3, ricxvi4;
// ფაილის შექმნა რიცხვების ჩასაწერად
ofstream out_file("file_1.dat");
out_file << ricxvi1 << " " << ricxvi2;
out_file.close();
// ფაილის გახსნა რიცხვების წასაკითხად
ifstream in_file("file_1.dat");
in_file >> ricxvi3 >> ricxvi4;
in_file.close();
cout << "ricxvi3 = " << ricxvi3 << " ricxvi4 = " << ricxvi4 << endl;

system("pause");
return 0;
}

```

მოყვანილი პროგრამით ჯერ ხდება ფაილში წილადი რიცხვების ჩაწერა და შემდეგ წაკითხვა.

```

#include "stdafx.h"
#include "iostream"
#include "fstream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
// ფაილში წილადი რიცხვების ჩაწერა და წაკითხვა
float ricxvi1 = 10.98, ricxvi2 = 25.64, ricxvi3, ricxvi4;
// ფაილის შექმნა რიცხვების ჩასაწერად
ofstream out_file("file_1.dat");
out_file << ricxvi1 << " " << ricxvi2;
out_file.close();
// ფაილის გახსნა რიცხვების წასაკითხად
ifstream in_file("file_1.dat");
in_file >> ricxvi3 >> ricxvi4;
in_file.close();
cout << "ricxvi3 = " << ricxvi3 << " ricxvi4 = " << ricxvi4 << endl;

system("pause");
return 0;
}

```

მოყვანილი პროგრამით ჯერ ხდება ფაილში სიმბოლოების ჩაწერა და შემდეგ წაკითხვა.

```

#include "stdafx.h"
#include "iostream"
#include "fstream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])

```

```

{
// ფაილში სიმბოლოების ჩაწერა და წაკითხვა
char simbolo1 = 'A', simbolo2 = 'd', simbolo3, simbolo4;
// ფაილის შექმნა რიცხვების ჩასაწერად
ofstream out_file("file_1.dat");
out_file << simbolo1 << " " << simbolo2;
out_file.close();
// ფაილის გახსნა რიცხვების წასაკითხად
ifstream in_file("file_1.dat");
in_file >> simbolo3 >> simbolo4;
in_file.close();
cout << "simbolo3 = " << simbolo3 << " simbolo4 = " << simbolo4 << endl;

system("pause");
return 0;
}

```

მოყვანილი პროგრამით ჯერ ხდება ფაილში სტრიქონების ჩაწერა და შემდეგ წაკითხვა.

```

#include "stdafx.h"
#include "iostream"
#include "string"
#include "fstream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])

```

```

{
// ფაილში სტრიქონების ჩაწერა და წაკითხვა
string striqoni1 = "Roman", striqoni2 = "Samkharadze", striqoni3, striqoni4;
// ფაილის შექმნა რიცხვების ჩასაწერად
ofstream out_file("file_1.dat");
out_file << striqoni1 << " " << striqoni2;
out_file.close();
// ფაილის გახსნა რიცხვების წასაკითხად
ifstream in_file("file_1.dat");
in_file >> striqoni3 >> striqoni4;
in_file.close();
cout << "striqoni3 = " << striqoni3 << " striqoni4 = " << striqoni4 << endl;

system("pause");
return 0;
}

```

მოყვანილი პროგრამით ჯერ ხდება ფაილში ლოგიკური მონაცემების ჩაწერა და შემდეგ წაკითხვა.

```

#include "stdafx.h"
#include "iostream"
#include "fstream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])

```

```

{
// ფაილში ლოგიკური მონაცემების ჩაწერა და წაკითხვა
bool logikuri1 = true, logikuri2 = false, logikuri3, logikuri4;
// ფაილის შექმნა რიცხვების ჩასაწერად
ofstream out_file("file_1.dat");
out_file << logikuri1 << " " << logikuri2;
out_file.close();
// ფაილის გახსნა რიცხვების წასაკითხად
ifstream in_file("file_1.dat");
in_file >> logikuri3 >> logikuri4;
in_file.close();
cout << "logikuri3 = " << logikuri3 << " logikuri4 = " << logikuri4 << endl;

system("pause");
return 0;
}

```

ეკრანზე true მნიშვნელობის ნაცვლად გამოჩნდება 1, false მნიშვნელობის ნაცვლად კი - 0. მოყვანილი პროგრამით ჯერ ხდება ფაილში სხვადასხვა ტიპის მონაცემების ჩაწერა, შემდეგ კი - წაკითხვა.

```

#include "stdafx.h"
#include "iostream"
#include "string"
#include "fstream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
// ფაილში სხვადასხვა ტიპის მონაცემების ჩაწერა და წაკითხვა
int i1 = 10, i2;
double d1 = 20.25, d2;
char c1 = 'R', c2;
string s1 = "Samkharadze", s2;

ofstream out_file("file_1.dat");
out_file << i1 << " " << d1 << " " << c1 << " " << s1;
out_file.close();
//
ifstream in_file("file_1.dat");
in_file >> i2 >> d2 >> c2 >> s2;
in_file.close();

cout << "i2 = " << i2 << " f2 = " << d2
<< " c2 = " << c2 << " s2 = " << s2 << endl;

system("pause");
return 0;
}

```

ფაილში სხვადასხვა ტიპის მასივების ჩაწერა და წაკითხვა

მოყვანილი პროგრამით ჯერ ხდება ფაილში მთელი რიცხვების ერთგანზომილებიანი მასივის ელემენტების ჩაწერა, შემდეგ კი - წაკითხვა.

```
#include "stdafx.h"
#include "iostream"
#include "fstream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ფაილში მთელი რიცხვების მასივის ჩაწერა და წაკითხვა
int masivi1[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }, masivi2[10];
int indexi;
// ფაილში masivi1 მასივის ელემენტების ჩაწერა
ofstream out_file("file_1.dat");
for ( indexi = 0; indexi < 10; indexi++ )
    out_file << masivi1[indexi] << " ";
out_file.close();
// ფაილიდან რიცხვების წაკითხვა და masivi2 მასივში ჩაწერა
ifstream in_file("file_1.dat");
for ( indexi = 0; indexi < 10; indexi++ )
    in_file >> masivi2[indexi];
in_file.close();
// masivi2 მასივის ელემენტების ეკრანზე გამოტანა
for ( indexi = 0; indexi < 10; indexi++ )
    cout << masivi2[indexi] << " ";
cout << endl;

system("pause");
return 0;
}
```

ანალოგიურად სრულდება ფაილში მთელი რიცხვების ორგანზომილებიანი მასივის ელემენტების ჩაწერა, შემდეგ კი - წაკითხვა.

```
#include "stdafx.h"
#include "iostream"
#include "fstream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ფაილში მთელი რიცხვების მასივის ჩაწერა და წაკითხვა
int masivi1[3][3] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } }, masivi2[3][3];
int str, sv;
// ფაილში masivi1 მასივის ელემენტების ჩაწერა
ofstream out_file("file_1.dat");
for ( str = 0; str < 3; str++ )
```

```

        for ( sv = 0; sv < 3; sv++ )
            out_file << masivi1[str][sv];
    out_file.close();
// ფაილიდან რიცხვების წაკითხვა და masivi2 მასივში ჩაწერა
ifstream in_file("file_1.dat");
for ( str = 0; str < 3; str++ )
    for ( sv = 0; sv < 3; sv++ )
        in_file >> masivi2[str][sv];
in_file.close();
// masivi2 მასივის ელემენტების ეკრანზე გამოტანა
for ( str = 0; str < 3; str++ )
    {
        for ( sv = 0; sv < 3; sv++ )
            cout << masivi2[str][sv] << " ";
        cout << endl;
    }

    system("pause");
    return 0;
}

```

მოყვანილი პროგრამით ჯერ ხდება ფაილში წილადი რიცხვების ერთგანზომილებიანი მასივის ჩაწერა, შემდეგ კი - წაკითხვა.

```

#include "stdafx.h"
#include "iostream"
#include "fstream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ფაილში წილადი რიცხვების ერთგანზომილებიანი მასივის ჩაწერა და წაკითხვა
double masivi1[10] = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9, 10.11 }, masivi2[10];
int indexi;
// ფაილში masivi1 მასივის ელემენტების ჩაწერა
ofstream out_file("file_1.dat");
for ( indexi = 0; indexi < 10; indexi++ )
    out_file << masivi1[indexi] << " ";
out_file.close();
// ფაილიდან რიცხვების წაკითხვა და masivi2 მასივში ჩაწერა
ifstream in_file("file_1.dat");
for ( indexi = 0; indexi < 10; indexi++ )
    in_file >> masivi2[indexi];
in_file.close();
// masivi2 მასივის ელემენტების ეკრანზე გამოტანა
for ( indexi = 0; indexi < 10; indexi++ )
    cout << masivi2[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}

```

```
}
```

ანალოგიურად სრულდება ფაილში მთელი რიცხვების ორგანზომილებიანი მასივის ელემენტების ჩაწერა, შემდეგ კი - წაკითხვა.

```
#include "stdafx.h"
#include "iostream"
#include "fstream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ფაილში წილადი რიცხვების ორგანზომილებიანი მასივის ჩაწერა და წაკითხვა
double masivi1[3][3] = { { 1.1, 2.2, 3.3 }, { 4.4, 5.5, 6.6 }, { 7.7, 8.8, 9.9 } }, masivi2[3][3];
int str, sv;
// ფაილში masivi1 მასივის ელემენტების ჩაწერა
ofstream out_file("file_1.dat");
for ( str = 0; str < 3; str++ )
    for ( sv = 0; sv < 3; sv++ )
        out_file << masivi1[str][sv] << " ";
out_file.close();
// ფაილიდან რიცხვების წაკითხვა და masivi2 მასივში ჩაწერა
ifstream in_file("file_1.dat");
for ( str = 0; str < 3; str++ )
    for ( sv = 0; sv < 3; sv++ )
        in_file >> masivi2[str][sv];
in_file.close();
// masivi2 მასივის ელემენტების ეკრანზე გამოტანა
for ( str = 0; str < 3; str++ )
    {
    for ( sv = 0; sv < 3; sv++ )
        cout << masivi2[str][sv] << " ";
    cout << endl;
    }

    system("pause");
    return 0;
}
```

მოყვანილი პროგრამით ჯერ ხდება ფაილში სიმბოლოების ერთგანზომილებიანი მასივის ელემენტების ჩაწერა, შემდეგ კი - წაკითხვა.

```
#include "stdafx.h"
#include "iostream"
#include "fstream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ფაილში სიმბოლოების ერთგანზომილებიანი მასივის ჩაწერა და წაკითხვა
char masivi1[10] = { 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j' }, masivi2[10];
int indexi;
// ფაილში masivi1 მასივის ელემენტების ჩაწერა
```

```

ofstream out_file("file_1.dat");
for ( indexi = 0; indexi < 10; indexi++ )
    out_file << masivi1[indexi] << " ";
out_file.close();
// ფაილიდან რიცხვების წაკითხვა და masivi2 მასივში ჩაწერა
ifstream in_file("file_1.dat");
for ( indexi = 0; indexi < 10; indexi++ )
    in_file >> masivi2[indexi];
in_file.close();
// masivi2 მასივის ელემენტების ეკრანზე გამოტანა
for ( indexi = 0; indexi < 10; indexi++ )
    cout << masivi2[indexi] << " ";
cout << endl;

system("pause");
return 0;
}

```

ანალოგიურად სრულდება ფაილში სიმბოლოების ორგანზომილებიანი მასივის ელემენტების ჩაწერა, შემდეგ კი - წაკითხვა.

```

#include "stdafx.h"
#include "iostream"
#include "fstream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ფაილში სიმბოლოების ორგანზომილებიანი მასივის ჩაწერა და წაკითხვა
int masivi1[3][3] = { { 'a', 'b', 'c' }, { 'd', 'e', 'f' }, { 'g', 'h', 'i' } }, masivi2[3][3];
int str, sv;
// ფაილში masivi1 მასივის ელემენტების ჩაწერა
ofstream out_file("file_1.dat");
for ( str = 0; str < 3; str++ )
    for ( sv = 0; sv < 3; sv++ )
        out_file << masivi1[str][sv] << ' ';
out_file.close();
// ფაილიდან რიცხვების წაკითხვა და masivi2 მასივში ჩაწერა
ifstream in_file("file_1.dat");
for ( str = 0; str < 3; str++ )
    for ( sv = 0; sv < 3; sv++ )
        in_file >> masivi2[str][sv];
in_file.close();
// masivi2 მასივის ელემენტების ეკრანზე გამოტანა
for ( str = 0; str < 3; str++ )
    {
    for ( sv = 0; sv < 3; sv++ )
        cout << ( char ) masivi2[str][sv] << " ";
    cout << endl;
    }
}

```



```

    system("pause");
    return 0;
}

```

მოყვანილი პროგრამით ჯერ ხდება ფაილში სტრიქონების ერთგანზომილებიანი მასივის ელემენტების ჩაწერა, შემდეგ კი - წაკითხვა.

```

#include "stdafx.h"
#include "iostream"
#include "string"
#include "fstream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ფაილში სტრიქონების ერთგანზომილებიანი მასივის ჩაწერა და წაკითხვა
string masivi1[5] = { "saba", "ana", "lika", "nata", "romani" }, masivi2[5];
int indexi;
// ფაილში masivi1 მასივის ელემენტების ჩაწერა
ofstream out_file("file_1.dat");
for ( indexi = 0; indexi < 5; indexi++ )
    out_file << masivi1[indexi] << " ";
out_file.close();
// ფაილიდან რიცხვების წაკითხვა და masivi2 მასივში ჩაწერა
ifstream in_file("file_1.dat");
for ( indexi = 0; indexi < 5; indexi++ )
    in_file >> masivi2[indexi];
in_file.close();
// masivi2 მასივის ელემენტების ეკრანზე გამოტანა
for ( indexi = 0; indexi < 5; indexi++ )
    cout << masivi2[indexi] << " ";
cout << endl;

    system("pause");
    return 0;
}

```

ანალოგიურად სრულდება ფაილში სტრიქონების ორგანზომილებიანი მასივის ელემენტების ჩაწერა, შემდეგ კი - წაკითხვა.

```

#include "stdafx.h"
#include "iostream"
#include "string"
#include "fstream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
// ფაილში სტრიქონების ორგანზომილებიანი მასივის ჩაწერა და წაკითხვა
string masivi1[3][3] = { { "ana", "saba", "lika" }, { "romani", "nata", "levani" },
                        { "achiko", "giorgi", "beqa" } }, masivi2[3][3];
int str, sv;
// ფაილში masivi1 მასივის ელემენტების ჩაწერა

```

```

ofstream out_file("file_1.dat");
for ( str = 0; str < 3; str++ )
    for ( sv = 0; sv < 3; sv++ )
        out_file << masivi1[str][sv] << " ";
out_file.close();
// ფაილიდან რიცხვების წაკითხვა და masivi2 მასივში ჩაწერა
ifstream in_file("file_1.dat");
for ( str = 0; str < 3; str++ )
    for ( sv = 0; sv < 3; sv++ )
        in_file >> masivi2[str][sv];
in_file.close();
// masivi2 მასივის ელემენტების ეკრანზე გამოტანა
for ( str = 0; str < 3; str++ )
    {
    for ( sv = 0; sv < 3; sv++ )
        cout << masivi2[str][sv] << " ";
    cout << endl;
    }

system("pause");
return 0;
}

```

ფაილისთვის მონაცემების დამატება

ფაილს შეგვიძლია დავუმატოთ ნებისმიერი ტიპის მონაცემი. მოყვანილი პროგრამით ჯერ ფაილში ჩაიწერება int, double და char ტიპის მონაცემები. შემდეგ ამ მონაცემებს დაემატება string ტიპის მონაცემი. ბოლოს, ფაილიდან სრულდება ყველა მონაცემის წაკითხვა.

```

#include "stdafx.h"
#include "iostream"
#include "fstream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int i1 = 5, i2;
    double d1 = 5.5, d2;
    char c1 = '+', c2;
    string s1 = "C+", s2;
    // i1, d1 და c1 ცვლადების მნიშვნელობების ჩაწერა ფაილში.
    ofstream out_file_W("file_1.dat");
    out_file_W << i1 << ' ' << d1 << ' ' << c1;
    out_file_W.close();
    // ფაილს დაემატა s1 სტრიქონი.
    ofstream app_file_A("file_1.dat", ios::app);
    app_file_A << s1;
}

```

```
app_file_A.close();
//      ფაილიდან წაკითხული მონაცემები ენიჭება i2, d2, c2 და s2 ცვლადებს.
ifstream in_file_R("file_1.dat");
in_file_R >> i2 >> d2 >> c2 >> s2;
in_file_R.close();
//      წაკითხული მონაცემების ეკრანზე გამოტანა.
cout << i2 << " " << d2 << " " << c2 << " " << s2 << endl;

system("pause");
return 0;
}
```

თავი 7. ფუნქციები

ფუნქცია

ფუნქცია არის კოდის თვითსაკმარისი ნაწილი, რომელიც გარკვეულ მოქმედებებს ასრულებს. მას აქვს სახელი და შეიძლება პროგრამული კოდის ერთ ან მეტ სტრიქონს შეიცავდეს. ფუნქციის სახელად შეგვიძლია ნებისმიერი იდენტიფიკატორის გამოყენება. საკვანძო სიტყვის გამოყენება ფუნქციის სახელად არ შეიძლება. Main() სახელი დარეზერვებულია იმ ფუნქციისთვის, საიდანაც პროგრამის შესრულება იწყება. ძირითად ფუნქციაში ფუნქციის გამოძახება ბევრჯერ შეიძლება.

ფუნქცია შედგება სათაურისა და კოდისგან (ტანისგან). ფუნქციის სათაური შედგება ფუნქციის მიერ გაცემული შედეგის ტიპის, ფუნქციის სახელისა და პარამეტრების სიისაგან.

ფუნქციის გამოცხადების სინტაქსია:

შედეგის_ტიპი ფუნქციის_სახელი ([პარამეტრების_სია])

{

ფუნქციის კოდი

}

აქ, **შედეგის_ტიპი** არის ფუნქციის მიერ გასაცემი მნიშვნელობის ტიპი. თუ ფუნქცია მნიშვნელობას არ გასცემს, მაშინ უნდა მივუთითოთ void ტიპი. **ფუნქციის_სახელი** არის ნებისმიერი იდენტიფიკატორი. **პარამეტრების_სია** შედგება პარამეტრების სახელებისაგან, რომლებიც ერთმანეთისგან მძიმეებით გამოიყოფა. თითოეული პარამეტრის სახელის წინ უნდა ეწეროს შესაბამისი ტიპი. პარამეტრი ესაა ცვლადი, რომელიც იღებს არგუმენტის მნიშვნელობას, რომელიც ფუნქციას გადაეცემა მისი გამოძახებისას. კვადრატული ფრჩხილები იმაზე მიუთითებენ, რომ პარამეტრები შეიძლება გამოვტოვოთ.

არ შეიძლება ერთი ფუნქციის შიგნით მეორე ფუნქციის განსაზღვრა. არ არის აუცილებელი გლობალური ცვლადების ჩართვა პარამეტრების სიაში, რადგან ისინი ყოველთვის მისაწვდომია ფუნქციისთვის.

ფუნქცია უნდა მოვათავსოთ

```
int _tmain(int argc, _TCHAR* argv[])
```

სტრიქონის ზემოთ (წინ)

```
using namespace std;
```

სტრიქონის შემდეგ.

შევადგინოთ ფუნქცია, რომელიც ეკრანზე რაიმე ტექსტს გამოიტანს:

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
void Function_1()
```

```
{
```

```
    cout << "C++ is a programming language" << endl;
```

```
}
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
    Function_1();
```

```

    system("pause");
    return 0;
}

```

ამ შემთხვევაში, Function_1() ფუნქცია არაფერს არ გვიბრუნებს, ამიტომ მისი სახელის მარცხნივ მითითებულია void სიტყვა. ამ ფუნქციას პარამეტრების არ აქვს. მის გამოსაძახებლად ძირითად პროგრამაში ვუთითებთ მხოლოდ ფუნქციის სახელს.

ფუნქციის გამოცხადება შეიძლება **პროტოტიპის** გამოყენებითაც. პროტოტიპი არის ფუნქციის სათაური, რომელიც წერტილ-მძიმით მთავრდება. პროტოტიპი უნდა მიეთითოს int _tmain(int argc, _TCHAR* argv[]) სტრიქონის ზემოთ (წინ), ფუნქციის გამოძახებამდე. თვით ფუნქცია სათაურთან და კოდთან ერთად ეთითება ძირითადი პროგრამის შემდეგ. ზემოთ მოყვანილი ფუნქცია ჩავწეროთ პროტოტიპის გამოყენებით.

```

#include "stdafx.h"
#include "iostream"
using namespace std;
// ფუნქციის პროტოტიპი
void Function_1();

int _tmain(int argc, _TCHAR* argv[])
{
    Function_1();

    system("pause");
    return 0;
}
void Function_1()
{
    cout << "C++ is a programming language" << endl;
}

```

პარამეტრის გამოყენება

გამოძახებისას ფუნქციას შეგვიძლია გადავცეთ ერთი ან მეტი მნიშვნელობა. ფუნქციისთვის გადასაცემ მნიშვნელობას არგუმენტი ეწოდება. ფუნქციის პარამეტრების სიაში გამოცხადებულ ცვლადს, რომელიც იღებს არგუმენტის მნიშვნელობას, პარამეტრი ეწოდება. პარამეტრების გამოცხადება ხდება მრგვალი ფრჩხილების შიგნით, რომლებიც ფუნქციის სახელს მოსდევს. პარამეტრის გამოცხადების სინტაქსი ცვლადების გამოცხადების სინტაქსის ანალოგიურია. პარამეტრი იმყოფება თავისი ფუნქციის ხილვადობის უბანში, ანუ ხილულია მხოლოდ ამ ფუნქციის შიგნით და მოქმედებს როგორც ლოკალური ცვლადი. თუ ფუნქციას რამდენიმე პარამეტრი აქვს, მაშინ ისინი ერთმანეთისგან მძიმეებით გამოიყოფა.

ახლა შევადგინოთ ფუნქცია, რომელსაც ერთი მთელი პარამეტრი აქვს. ფუნქცია ამოწმებს ეს პარამეტრი დადებითი თუ უარყოფითი და ეკრანზე გამოაქვს შესაბამისი შეტყობინება.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

void Function_1( int par1 )
{

```

```

        if ( par1 >= 0 ) cout << "ricxvi dadebitia" << endl;
            else cout << "ricxvi uaryofitia" << endl;
    }

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi;
    cin >> ricxvi;
    Function_1(ricxvi);

    system("pause");
    return 0;
}

```

Function_1() ფუნქცია ამ შემთხვევაშიც არაფერს არ გვიბრუნებს, მაგრამ მას გადავეცით ერთი არგუმენტი, კერძოდ მთელი ტიპის არგუმენტი. ძირითად პროგრამაში ამ ფუნქციის გამოძახებისას par1 პარამეტრს გადაეცემა ricxvi არგუმენტი. სხვა სიტყვებით, რომ ვთქვათ par1 პარამეტრის მნიშვნელობა ენიჭება ricxvi არგუმენტს.

Function_1() ფუნქცია ჩავწერთ პროტოტიპის გამოყენებით. ამ დროს პარამეტრის სახელის მითითება აუცილებელი არ არის. აუცილებელია პარამეტრის ტიპის მითითება.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

void Function_1( int ); // ფუნქციის პროტოტიპი

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi;
    cin >> ricxvi;
    Function_1(ricxvi);

    system("pause");
    return 0;
}

```

```

void Function_1( int par1 )
{
    if ( par1 >= 0 ) cout << "ricxvi dadebitia" << endl;
        else cout << "ricxvi uaryofitia" << endl;
}

```

ფუნქციიდან მართვის დაბრუნება

ფუნქციიდან მართვის დაბრუნება ანუ ფუნქციის შესრულების შეწყვეტა და გამომძახებელი პროგრამისთვის მართვის დაბრუნება, ორ შემთხვევაში ხდება. პირველი, როდესაც გვხვდება ფუნქციის დამხურავი ფიგურული ფრჩხილი და მეორე, როდესაც სრულდება return ოპერატორი. არსებობს return ოპერატორის ორი ფორმა. პირველი ფორმა გამოიყენება ფუნქციაში, რომელიც მნიშვნელობას არ აბრუნებს (ფუნქციას void ტიპი აქვს),

მეორე კი - ფუნქციაში, რომელიც მნიშვნელობას აბრუნებს.

ფუნქციაში, რომელიც მნიშვნელობას არ აბრუნებს, გამოიყენება შემდეგი სინტაქსის return ოპერატორი:

[return;]

კვადრატული ფრჩხილებია იმაზე მიუთითებენ, რომ ასეთ ფუნქციაში return ოპერატორი შეგვიძლია არ მივუთითოთ.

return ოპერატორის შესრულებისას მართვა გადაეცემა გამომძახებელი პროგრამის იმ ადგილს, საიდანაც ფუნქცია იყო გამოძახებული, ამასთან ფუნქციის კოდის დარჩენილი ნაწილი არ შესრულდება.

ფუნქციის ერთ-ერთი მნიშვნელოვანი თვისებაა მის მიერ მნიშვნელობის დაბრუნება (გაცემა). ფუნქციები მნიშვნელობას უბრუნებენ გამომძახებელ პროგრამას შემდეგი სინტაქსის მქონე return ოპერატორის გამოყენებით:

return მნიშვნელობა;

შევადგინოთ ფუნქცია, რომელიც გამოთვლის და დაგვიბრუნებს სამკუთხედის პერიმეტრს.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int Perimetri(int gverdi1, int gverdi2, int gverdi3)
{
    return gverdi1 + gverdi2 + gverdi3;
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    int gverdi11, gverdi12, gverdi13, perimetri1;
    int gverdi21, gverdi22, gverdi23, perimetri2;

    cin >> gverdi11 >> gverdi12 >> gverdi13;
    cin >> gverdi21 >> gverdi22 >> gverdi23;

    perimetri1 = Perimetri(gverdi11, gverdi12, gverdi13);
    perimetri2 = Perimetri(gverdi21, gverdi22, gverdi23);

    cout << "pirveli samkutkhedis perimetri = " << perimetri1 << endl;
    cout << "meore samkutkhedis perimetri = " << perimetri2 << endl;

    system("pause");
    return 0;
}
```

აქ გამოცხადებულია Perimetri() ფუნქცია, რომელიც გვიბრუნებს (გასცემს) მთელ რიცხვს და აქვს სამი მთელი ტიპის პარამეტრი. ძირითად პროგრამაში ეს ფუნქცია ორჯერ გამოიძახება

```
perimetri1 = Perimetri(gverdi11, gverdi12, gverdi13);
```

```
და
perimetri2 = Perimetri(gverdi21, gverdi22, gverdi23);
```

სტრიქონებში. პირველი გამოძახების დროს, ფუნქციის პარამეტრებს გადაეცემათ gverdi11, gverdi12 და gverdi13 არგუმენტები. კერძოდ, gverdi1 პარამეტრს მიენიჭება gverdi11 არგუმენტის,

gverdi2 პარამეტრს - gverdi12 არგუმენტის, ხოლო gverdi3 პარამეტრს - gverdi13 არგუმენტის მნიშვნელობა. ფუნქცია დაგვიბრუნებს შესაბამისი მნიშვნელობის მქონე პერიმეტრს, რომელიც მიენიჭება perimetri1 ცვლადს. მეორე გამოძახების შემთხვევაში, ფუნქციის პარამეტრებს გადაეცემათ gverdi21, gverdi22 და gverdi23 არგუმენტები. კერძოდ, gverdi1 პარამეტრს მიენიჭება gverdi21 არგუმენტის, gverdi2 პარამეტრს - gverdi22 არგუმენტის, ხოლო gverdi3 პარამეტრს - gverdi23 არგუმენტის მნიშვნელობა. ფუნქცია დაგვიბრუნებს შესაბამისი მნიშვნელობის მქონე პერიმეტრს, რომელიც მიენიჭება perimetri2 ცვლადს.

იგივე ფუნქცია ჩავწერთ პროტოტიპის გამოყენებით.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int Perimetri(int , int , int );           // ფუნქციის პროტოტიპი

int _tmain(int argc, _TCHAR* argv[])
{
    int gverdi11, gverdi12, gverdi13, perimetri1;
    int gverdi21, gverdi22, gverdi23, perimetri2;

    cin >> gverdi11 >> gverdi12 >> gverdi13;
    cin >> gverdi21 >> gverdi22 >> gverdi23;

    perimetri1 = Perimetri(gverdi11, gverdi12, gverdi13);
    perimetri2 = Perimetri(gverdi21, gverdi22, gverdi23);

    cout << "pirveli samkutkhedis perimetri = " << perimetri1<< endl;
    cout << "meore samkutkhedis perimetri = " << perimetri2<< endl;

    system("pause");
    return 0;
}
int Perimetri(int gverdi1, int gverdi2, int gverdi3)
{
    return gverdi1 + gverdi2 + gverdi3;
}
```

განვიხილოთ მეორე პროგრამა, რომელშიც ფუნქციისთვის მნიშვნელობის გადასაცემად პარამეტრი გამოიყენება. luwi_kenti(int par1) ფუნქციას ერთი მთელი ტიპის პარამეტრი აქვს და აბრუნებს (გასცემს) bool ტიპის შედეგს. თუ მისთვის გადაცემული მნიშვნელობა არის ლუწი, მაშინ ის გასცემს true მნიშვნელობას, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
// ფუნქციის განსაზღვრა, რომელიც ამოწმებს რიცხვი კენტია თუ ლუწი
bool Luwi_kenti( int par1)
{
    if ( (par1 % 2 ) == 0 ) return true;
    else return false;
}
```



```

int _tmain(int argc, _TCHAR* argv[])
{
int ricxvi;

cin >> ricxvi;
if ( Luwi_kenti(ricxvi) ) cout << "ricxvi " << ricxvi << " lucia" << endl;
    else cout << "ricxvi " << ricxvi << " kentia" << endl;

system("pause");
return 0;
}

```

Luwi_kenti() ფუნქციის გამოძახებისას ricxvi არგუმენტის მნიშვნელობა მიენიჭება par1 პარამეტრს.

ახლა შევადგინოთ arisJeradi() ფუნქცია, რომელიც განსაზღვრავს არის თუ არა პირველი პარამეტრი მეორე პარამეტრის ჯერადი.

```

#include "stdafx.h"
#include "iostream"
using namespace std;
// ფუნქცია ამოწმებს პირველი პარამეტრი უნაშთოდ იყოფა თუ არა მეორე პარამეტრზე
bool arisJeradi(int par1, int par2)
{
if ( ( par1 % par2 ) == 0 ) return true;
    else return false;
}

```

```

int _tmain(int argc, _TCHAR* argv[])
{
int ricxvi1, ricxvi2;

cin >> ricxvi1 >> ricxvi2;
// ფუნქციის გამოძახება
if ( arisJeradi(ricxvi1, ricxvi2) ) cout << ricxvi1 << " aris " << ricxvi2 << "-is jeradi" << endl;
    else cout << ricxvi1 << " ar aris " << ricxvi2 << "-is jeradi" << endl;

system("pause");
return 0;
}

```

arisJeradi() ფუნქციის გამოძახებისას ricxvi1 არგუმენტის მნიშვნელობა მიენიჭება par1 პარამეტრს, ricxvi2 არგუმენტის მნიშვნელობა კი - par2 პარამეტრს.

ფუნქციისთვის მასივების გადაცემა

ფუნქციას შეგვიძლია გადავცეთ მასივი. ამ დროს ფუნქციას გადაეცემა მასივის საწყისი მისამართი, ანუ მასივის პირველი ელემენტის მისამართი.

ქვემოთ მოცემულ პროგრამაში Minimumi() ფუნქციას გადაეცემა მთელი რიცხვების ერთგანზომილებიანი მასივი. ფუნქცია პოულობს და აბრუნებს მასივის მინიმალური

ელემენტის მნიშვნელობას.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
// ფუნქციისთვის ერთგანზომილებიანი მასივის გადაცემის დემონსტრირება
int Minimumi(int mas[])
{
    int min_ricxvi = mas[0];
    for ( int indexi =1; indexi < 5; indexi++ )
        if ( min_ricxvi > mas[indexi] ) min_ricxvi = mas[indexi];
    return min_ricxvi;
}

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[5] = { 5, -1, 8, -4, 2 };

    int shedegi = Minimumi(masivi); // Minimumi ფუნქციას masivi მასივი გადაეცემა
    cout << "minimaluri elementi = " << shedegi<< endl;
    for ( int indexi = 0; indexi < 5; indexi++ )
        cout << masivi[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}
```

აქ, პროგრამის `int shedegi = Minimumi(masivi);` სტრიქონში ხდება `Minimumi()` ფუნქციის გამოძახება. მას გადაეცემა `masivi` მასივის საწყისი მისამართი (პირველი ელემენტის მისამართი), რომელიც მიენიჭება `mas` პარამეტრს. შედეგად, `masivi` და `mas` მასივები მესხიერების ერთსა და იმავე უბანს მიმართავენ. ამიტომ, მასივთან მუშაობა შესრულდება იმ უბანში, რომელიც `masivi` მასივისთვის გამოიყო ძირითად პროგრამაში.

იგივე ფუნქცია შეგვიძლია პროტოტიპის გამოყენებით ჩავწეროთ.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
// ფუნქციისთვის ერთგანზომილებიანი მასივის გადაცემის დემონსტრირება
int Minimumi(int []); // ფუნქციის პროტოტიპი

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[5] = { 5, -1, 8, -4, 2 };

    int shedegi = Minimumi(masivi); // Minimumi ფუნქციას masivi მასივი გადაეცემა
    cout << "minimaluri elementi = " << shedegi<< endl;
    for ( int indexi = 0; indexi < 5; indexi++ )
        cout << masivi[indexi] << " ";
    cout << endl;
}
```

```

system("pause");
return 0;
}
int Minimumi(int mas[])
{
int min_ricxvi = mas[0];
for ( int indexi =1; indexi < 5; indexi++ )
    if ( min_ricxvi > mas[indexi] ) min_ricxvi = mas[indexi];
return min_ricxvi;
}

```

ქვემოთ მოცემულ პროგრამაში Minimumi() ფუნქციას გადაეცემა მთელი რიცხვების ორგანზომილებიანი მასივი. ფუნქცია პოულობს და აბრუნებს მასივის მინიმალური ელემენტის მნიშვნელობას. ამ შემთხვევაშიც, ფუნქციას გადაეცემა მისი პირველი ელემენტის მისამართი.

```

#include "stdafx.h"
#include "iostream"
using namespace std;
// ფუნქციისთვის ორგანზომილებიანი მასივის გადაცემის დემონსტრირება
int Minimumi(int mas[][3])
{
int min_ricxvi = mas[0][0];
for ( int striqoni =0; striqoni < 3; striqoni++ )
    for ( int sveti =0; sveti < 3; sveti++ )
if ( min_ricxvi > mas[striqoni][sveti] ) min_ricxvi = mas[striqoni][sveti];
return min_ricxvi;
}

```

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi[3][3] = { { 5, -1, 8}, {-4, 2, 3}, { 2, 7, -3} };

```

```

int shedegi = Minimumi(masivi); // Minimumi ფუნქციას masivi მასივი გადაეცემა
cout << "minimaluri elementi = " << shedegi << endl;
for ( int striqoni =0; striqoni < 3; striqoni++ )
{
for ( int sveti =0; sveti < 3; sveti++ )
    cout << masivi[striqoni][sveti] << " ";
cout << endl;
}

```

```

system("pause");
return 0;
}

```

აქ, ფუნქციის გამოცხადებისას უნდა მიეთითოს ან მხოლოდ სვეტების რაოდენობა:

```

int Minimumi(int mas[][3]),
ან სტრიქონებისა და სვეტების რაოდენობა:
int Minimumi(int mas[3][3]).

```

იგივე ფუნქცია შეგვიძლია პროტოტიპის გამოყენებით ჩავწეროთ.

```

#include "stdafx.h"
#include "iostream"
using namespace std;
// ფუნქციისთვის ორგანზომილებიანი მასივის გადაცემის დემონსტრირება
int Minimumi(int [][][3]);

int _tmain(int argc, _TCHAR* argv[])
{
int masivi[3][3] = { { 5, -1, 8}, {-4, 2, 3}, { 2, 7, -3} };

int shedegi = Minimumi(masivi); // Minimumi ფუნქციას masivi მასივი გადაეცემა
cout << "minimaluri elementi = " << shedegi << endl;
for ( int striqoni =0; striqoni < 3; striqoni++ )
{
for ( int sveti =0; sveti < 3; sveti++ )
cout << masivi[striqoni][sveti] << " ";
cout << endl;
}

system("pause");
return 0;
}
int Minimumi(int mas[][3])
{
int min_ricxvi = mas[0][0];
for ( int striqoni =0; striqoni < 3; striqoni++ )
for ( int sveti =0; sveti < 3; sveti++ )
if ( min_ricxvi > mas[striqoni][sveti] ) min_ricxvi = mas[striqoni][sveti];
return min_ricxvi;
}

```

ფუნქციას შეიძლება გადავცეთ წილადების, ლოგიკური მონაცემების, სიმბოლოებისა და სტრიქონების ერთ და ორგანზომილებიანი მასივები.

პარამეტრების გადაცემა მიმთითებლებისა და მიმართვების გამოყენებით

ფუნქციისთვის პარამეტრების გადაცემისას შეგვიძლია მიმთითებლებისა და მიმართვების გამოყენება. შევადგინოთ ფუნქცია, რომელსაც ორი პარამეტრი გადაეცემა. პარამეტრები წარმოადგენს მთელ რიცხვზე მიმთითებლებს. ფუნქცია ამ პარამეტრებს მნიშვნელობებს უცვლის.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

void Gacvla(int *par1, int *par2)
{

```

```

int temp;
temp = *par1;
*par1 = *par2;
*par2 = temp;
}

int _tmain(int argc, _TCHAR* argv[])
{
int ricxvi1, ricxvi2;

cin >> ricxvi1 >> ricxvi2;
cout << "pirveli ricxvi = " << ricxvi1 << "   meore ricxvi = " << ricxvi2 << endl;
Gacvla(&ricxvi1, &ricxvi2);
cout << "gacvlis shemdeg:" << endl;
cout << "pirveli ricxvi = " << ricxvi1 << "   meore ricxvi = " << ricxvi2 << endl;

system("pause");
return 0;
}

```

Gacvla ფუნქციის პარამეტრები მთელ რიცხვზე მიმთითებლებია. ამიტომ, ძირითადი პროგრამიდან ამ ფუნქციის გამოძახებისას მის par1 და par2 პარამეტრებს, შესაბამისად გადაეცემა ricxvi1 და ricxvi2 ცვლადების მისამართები. შედეგად, ამ პარამეტრების მნიშვნელობების შეცვლა გამოიწვევს ricxvi1 და ricxvi2 ცვლადების მნიშვნელობების შეცვლას. ეს ხდება იმიტომ, რომ მონაცემების დამუშავება შესრულდა იმ უბანში, რომელიც ricxvi1 და ricxvi2 ცვლადებს გამოეყო ძირითად პროგრამაში.

იგივე ფუნქცია პროტოტიპის გამოყენებით ჩავწერთ.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

void Gacvla(int *, int *);           //   ფუნქციის პროტოტიპი

int _tmain(int argc, _TCHAR* argv[])
{
int ricxvi1, ricxvi2;

cin >> ricxvi1 >> ricxvi2;
cout << "pirveli ricxvi = " << ricxvi1 << "   meore ricxvi = " << ricxvi2 << endl;
Gacvla(&ricxvi1, &ricxvi2);
cout << "gacvlis shemdeg:" << endl;
cout << "pirveli ricxvi = " << ricxvi1 << "   meore ricxvi = " << ricxvi2 << endl;

system("pause");
return 0;
}
void Gacvla(int *par1, int *par2)
{
int temp;
temp = *par1;

```

```
*par1 = *par2;
*par2 = temp;
}
```

იგივე ფუნქცია შევადგინოთ მიმართვების გამოყენებით.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

void Gacvla(int &par1, int &par2)
{
int temp;
temp = par1;
par1 = par2;
par2 = temp;
}
//
int _tmain(int argc, _TCHAR* argv[])
{
int ricxvi1, ricxvi2;

cin >> ricxvi1 >> ricxvi2;
cout << "pirveli ricxvi = " << ricxvi1 << "   meore ricxvi = " << ricxvi2 << endl;
Gacvla(ricxvi1, ricxvi2);
cout << "gacvlis shemdeg:" << endl;
cout << "pirveli ricxvi = " << ricxvi1 << "   meore ricxvi = " << ricxvi2 << endl;

system("pause");
return 0;
}
```

Gacvla ფუნქციის პარამეტრები მთელ რიცხვზე მიმართვებია ანუ მთელი რიცხვების ფსევდონიმებია. ამიტომ, ძირითადი პროგრამიდან ამ ფუნქციის გამოძახებისას ricxvi1 და ricxvi2 არგუმენტების ფსევდონიმები შესაბამისად ხდება par1 და par2 პარამეტრები. ე.ი., მეხსიერების უბნებს, რომელიც ricxvi1 და ricxvi2 ცვლადებისთვის გამოიყო, შესაბამისად დაერქვა par1 და par2 სახელები. ამიტომ, par1 და par2 ცვლადების მნიშვნელობების შეცვლა შეცვლის ricxvi1 და ricxvi2 ცვლადების მნიშვნელობებს, რადგან ცვლადების დამუშავება ერთსა და იმავე უბანში ხდება.

იგივე ფუნქცია პროტოტიპის გამოყენებით ჩავწეროთ.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

void Gacvla(int &, int &);           //   ფუნქციის პროტოტიპი

int _tmain(int argc, _TCHAR* argv[])
{
int ricxvi1, ricxvi2;

cin >> ricxvi1 >> ricxvi2;
cout << "pirveli ricxvi = " << ricxvi1 << "   meore ricxvi = " << ricxvi2 << endl;
```

```
Gacvla(ricxvi1, ricxvi2);
cout << "gacvlis shemdeg:" << endl;
cout << "pirveli ricxvi = " << ricxvi1 << "   meore ricxvi = " << ricxvi2 << endl;
```

```
system("pause");
return 0;
}
void Gacvla(int &par1, int &par2)
{
int temp;
temp = par1;
par1 = par2;
par2 = temp;
}
```

როგორც ვიცით, return ოპერატორი გასცემს ერთ შედეგს და ამთავრებს ფუნქციის შესრულებას. იმისათვის, რომ ფუნქციიდან მივიღოთ ერთზე მეტი მნიშვნელობა უნდა გამოვიყენოთ მიმთითებლები და მიმართვები. მოყვანილი ფუნქცია return ოპერატორის მეშვეობით გასცემს მართკუთხედის პერიმეტრის, ხოლო მიმთითებლის მეშვეობით კი - ფართობის მნიშვნელობას.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int FartPerim(int par1, int par2, int *pfartobi)
{
*pfartobi = par1 * par2;
return 2 * ( par1 + par2 );
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
int sigrdze, sigane, fartobi, perimetri;
```

```
cin >> sigrdze >> sigane;
// ფუნქციის გამოძახება
perimetri = FartPerim(sigrdze, sigane, &fartobi);
cout << "fartobi = " << fartobi << "   perimetri = " << perimetri << endl;
```

```
system("pause");
return 0;
}
```

ამ შემთხვევაში, pfartobi მიმთითებელს გადაეცემა fartobi ცვლადის მისამართი. ამიტომ, pfartobi მისამართით ჩაწერილი მნიშვნელობა მიენიჭება fartobi ცვლადს.

იგივე ფუნქცია პროტოტიპის გამოყენებით ჩავწეროთ.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
int FartPerim(int , int , int *);           // ფუნქციის პროტოტიპი
```

```

int _tmain(int argc, _TCHAR* argv[])
{
int sigrdze, sigane, fartobi, perimetri;

cin >> sigrdze >> sigane;
// ფუნქციის გამოძახება
perimetri = FartPerim(sigrdze, sigane, &fartobi);
cout << "fartobi = " << fartobi << " perimetri = " << perimetri << endl;

system("pause");
return 0;
}
int FartPerim(int par1, int par2, int *pfartobi)
{
*pfartobi = par1 * par2;
return 2 * ( par1 + par2 );
}

```

ახლა, შევადგინოთ ფუნქცია, რომელიც return ოპერატორის გამოყენებით მართკუთხედის პერიმეტრს დაგვიბრუნებს, ხოლო მიმართვის გამოყენებით კი - ფართობს.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int FartPerim1(int par1, int par2, int &fart)
{
fart = par1 * par2;
return 2 * (par1 + par2);
}

```

```

int _tmain(int argc, _TCHAR* argv[])
{
int sigrdze, sigane, fartobi, perimetri;

cin >> sigrdze >> sigane;
// ფუნქციის გამოძახება
perimetri = FartPerim1(sigrdze, sigane, fartobi);
cout << "fartobi = " << fartobi << " perimetri = " << perimetri << endl;

system("pause");
return 0;
}

```

ფუნქციას გამოძახებისას სამი არგუმენტი გადაეცემა. მესამეა fartobi ცვლადი, რომლის ფსევდონიმი ხდება fart პარამეტრი. ე.ი., მეხსიერების უბანს, რომელიც fartobi ცვლადისთვის გამოიყო, დაერქვა მეორე სახელი - fart. ამიტომ, fart ცვლადის მნიშვნელობის შეცვლა შეცვლის fartobi ცვლადის მნიშვნელობას, რადგან ცვლადების დამუშავება ერთსა და იმავე უბანში ხდება.

იგივე ფუნქცია პროტოტიპის გამოყენებით ჩავწეროთ.

```

#include "stdafx.h"
#include "iostream"

```



```

using namespace std;

int FartPerim1(int , int , int &);

int _tmain(int argc, _TCHAR* argv[])
{
int sigrdze, sigane, fartobi, perimetri;

cin >> sigrdze >> sigane;
// ფუნქციის გამოძახება
perimetri = FartPerim1(sigrdze, sigane, fartobi);
cout << "fartobi = " << fartobi << " perimetri = " << perimetri << endl;

system("pause");
return 0;
}
int FartPerim1(int par1, int par2, int &fart)
{
fart = par1 * par2;
return 2 * (par1 + par2);
}

```

ფუნქციის გადატვირთვა

C++ ენაში ორ ან მეტ ფუნქციას შეიძლება ერთნაირი სახელი ჰქონდეს იმ პირობით, რომ ან პარამეტრების რაოდენობა უნდა იყოს განსხვავებული ან თუ პარამეტრების რაოდენობა ერთნაირია, მაშინ მათ სხვადასხვა ტიპი უნდა ჰქონდეს. ასეთ ფუნქციებს **გადატვირთვადი** ეწოდებათ, ხოლო ერთნაირი სახელის მქონე ფუნქციების განსაზღვრის პროცესს - **ფუნქციის გადატვირთვა**. გადატვირთვადი ფუნქციის გამოძახებისას სრულდება ის ფუნქცია, რომლის პარამეტრებიც ემთხვევა არგუმენტებს. გადატვირთვადი ფუნქციები შეიძლება აბრუნებდეს სხვადასხვა ტიპის მონაცემებს. მაგალითი:

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

// ფუნქციის გადატვირთვის დემონსტრირება
// ფუნქციას პარამეტრები არ აქვს და აბრუნებს სტრიქონს
string Funqcia()
{
return "uparametrebo funqcia";
}
// ფუნქციას double ტიპის ერთი პარამეტრი აქვს და აბრუნებს მის კვადრატს
double Funqcia(double par1)
{
return pow(par1, 2);
}

```

```
// ფუნქციას int ტიპის ორი პარამეტრი აქვს და აბრუნებს მათ ჯამს
int Funqcia(int par1, int par2)
{
return par1 + par2;
}
// ფუნქციას double ტიპის ორი პარამეტრი აქვს და აბრუნებს მათ ჯამს
double Funqcia(double par1, double par2)
{
return par1 + par2;
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
int ricxvi1;
double wiladi1, wiladi2;
string striqoni;
```

```
// Funqcia ფუნქციის ყველა ვერსიის გამოძახება
striqoni = Funqcia();
cout << striqoni << endl;
wiladi2 = Funqcia(5.7);
cout << "ertparametriani funqcia. shedegi = " << wiladi2 << endl;
ricxvi1 = Funqcia(5, 10);
cout << "orparametriani funqcia. shedegi = " << ricxvi1 << endl;
wiladi1 = Funqcia(5.5, 10.10);
cout << "orparametriani funqcia. shedegi = " << wiladi1 << endl;
```

```
system("pause");
return 0;
}
```

პროგრამაში ოთხჯერ ხდება Funqcia() ფუნქციის გადატვირთვა. პირველ ვერსიას არ აქვს პარამეტრი. მეორე ვერსიას აქვს ერთ მთელი ტიპის პარამეტრი, მესამეს - ორი მთელი ტიპის პარამეტრი და მეოთხეს - ორი წილადი პარამეტრი.

თუ პარამეტრების რაოდენობა თანაბარია და არგუმენტებსა და პარამეტრებს განსხვავებული ტიპები აქვს, მაშინ სრულდება ტიპების ავტომატური გარდაქმნა. განვიხილოთ მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

```
// ტიპების ავტომატური გარდაქმნა ფუნქციის გადატვირთვისას
int Funqcia( int par1 )
{
return par1;
}
double Funqcia( double par1 )
{
```

```

        return par1;
    }

int _tmain(int argc, _TCHAR* argv[])
{
    int i1 = 15;
    double d1 = 15.15;
    short s1 = 20;
    float f1 = 5.5F;
    // აქ სრულდება Funqcia(int par1) ფუნქცია
    cout << Funqcia(i1) << endl;
    // აქ სრულდება Funqcia(double par1) ფუნქცია
    cout << Funqcia(d1) << endl;
    // აქ სრულდება Funqcia(int par1) ფუნქცია
    cout << Funqcia(s1) << endl;
    // აქ სრულდება Funqcia(double par1) ფუნქცია
    cout << Funqcia(f1) << endl;

    system("pause");
    return 0;
}

```

პროგრამაში Funqcia() ფუნქციის ერთ ვერსიას აქვს int ტიპის პარამეტრი, მეორეს კი - double ტიპის. მათი გამოძახება ხდება შესაბამისად int, double, short და float ტიპის მქონე არგუმენტებისათვის. ფუნქციისთვის short ტიპის მნიშვნელობის გადაცემისას ის ავტომატურად გარდაიქმნება int ტიპად, ხოლო float ტიპის მნიშვნელობის გადაცემისას კი - double ტიპად.

გადატვირთვა და არაცალსახობა

ფუნქციის გადატვირთვისას შეიძლება ადგილი ჰქონდეს *არაცალსახობას* (ambiguity) მისი გამოძახებისას. არაცალსახობის არსი იმაში მდგომარეობს, რომ კომპილატორს არ შეუძლია გაარკვიოს თუ რომელი გადატვირთული ფუნქცია უნდა გამოიძახოს. ფუნქციის გადატვირთვისას არაცალსახობა გამოწვეულია ტიპის გარდაქმნით, პარამეტრი-მიმართვებისა და ნაგულისხმევი არგუმენტების გამოყენებით. განვიხილოთ თითოეული შემთხვევა.

მოცემული პროგრამით ხდება არაცალსახობის დემონსტრირება, რომელიც გამოწვეულია ტიპის გარდაქმნით.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// არაცალსახობის დემონსტრირება, რომელიც გამოწვეულია ტიპის გარდაქმნით
double Funqcia(double par)
{
    return par / 2.0;
}
float Funqcia(float par)
{

```

```

        return (float)(par / 4.0);
    }
int _tmain(int argc, _TCHAR* argv[])
{
    float f1 = 3.5F;
    double d1;
    int i1;
    cin >> d1 >> i1;
    // არაცალსახობა არ არის. გამოიძახება Funqcia(float par) ფუნქცია
    cout << Funqcia(f1) << endl;
    // არაცალსახობა არ არის. გამოიძახება Funqcia(double par) ფუნქცია
    cout << Funqcia(d1) << endl;
    // ადგილი აქვს არაცალსახობას.
    cout << Funqcia(i1) << endl;

    system("pause");
    return 0;
}

```

აქ არაცალსახობას ადგილი აქვს Funqcia(i1) ფუნქციის გამოძახებისას. ამ დროს int ტიპი შეიძლება დაყვანილი იყოს როგორც float ტიპზე, ისე double ტიპზე. ამიტომ, შეიძლება გამოიძახებული იყოს როგორც Funqcia(f1), ისე Funqcia(d1) ფუნქცია. არაცალსახობის თავიდან ასაცილებლად უნდა გამოვიყენოთ მესამე ფუნქცია - int Funqcia(int par) ან i1 ცვლადი გამოვაცხადოთ როგორც float ან double.

მოცემული პროგრამით ხდება არაცალსახობის დემონსტრირება, რომელიც გამოწვეულია პარამეტრი-მიმართვის გამოყენებით.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// არაცალსახობის დემონსტრირება, რომელიც
// გამოწვეულია პარამეტრი-მიმართვის გამოყენებით
double Funqcia(double par1)
{
    return par1 * par1;
}
float Funqcia(double &par1)
{
    return par1 + par1;
}

int _tmain(int argc, _TCHAR* argv[])
{
    double ricxvi;
    cin >> ricxvi;
    // ადგილი აქვს არაცალსახობას.
    cout << Funqcia(ricxvi) << endl;
}

```

```

system("pause");
return 0;
}

```

აქ არაცალსახობას ადგილი აქვს Funqcia(wiladi) ფუნქციის გამოძახებისას. ამ შემთხვევაში შეიძლება გამოძახებული იყოს როგორც Funqcia(double par1), ისე Funqcia(double &par1) ფუნქცია. არაცალსახობის თავიდან ასაცილებლად უნდა წავშალოთ ერთ-ერთი ფუნქცია ან რომელიმე ფუნქციაში შევცვალოთ პარამეტრის ტიპი.

მოცემული C++ პროგრამით ხდება არაცალსახობის დემონსტრირება, რომელიც გამოწვეულია ნაგულისხმევი არგუმენტის გამოყენებით.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// არაცალსახობის დემონსტრირება, რომელიც
// გამოწვეულია ნაგულისხმევი არგუმენტის გამოყენებით
double Funqcia(double par)
{
    return par / 2.0;
}
float Funqcia(double par1, double par2 = 15)
{
    return par1 * par2;
}

int _tmain(int argc, _TCHAR* argv[])
{
    double wiladi1, wiladi2;

    cin >> wiladi1 >> wiladi2;
    // ადგილი არ აქვს არაცალსახობას.
    cout << Funqcia(wiladi1, wiladi2) << endl;
    // ადგილი აქვს არაცალსახობას.
    cout << Funqcia(wiladi1);

    system("pause");
    return 0;
}

```

აქ არაცალსახობას ადგილი აქვს Funqcia(wiladi1) ფუნქციის გამოძახებისას. ამ დროს შეიძლება გამოძახებული იყოს როგორც Funqcia(double par), ისე Funqcia(double par1, double par2 = 15) ფუნქცია. არაცალსახობის თავიდან ასაცილებლად რომელიმე ფუნქციაში უნდა შევცვალოთ პარამეტრის ტიპი.

ნაგულისხმევი არგუმენტი

ნაგულისხმევი არგუმენტი (default argument) გამოიყენება ფუნქციის გამოძახებისას პარამეტრისთვის წინასწარ განსაზღვრული მნიშვნელობის მისანიჭებლად იმ შემთხვევაში,

როდესაც შესაბამისი არგუმენტი მითითებული არ არის. ნაგულისხმევი არგუმენტის გამოყენება არის ფუნქციის ფარული გადატვირთვის ფორმა.

პარამეტრს რომ გადავცეთ ნაგულისხმევი არგუმენტი, ამისათვის ფუნქციის გამოცხადებისას უნდა შევასრულოთ პარამეტრის ინიციალიზება, ანუ საჭირო პარამეტრს უნდა მივანიჭოთ მნიშვნელობა. მაგალითად, გამოვაცხადოთ ორ პარამეტრიანი ფუნქცია და ორივე პარამეტრს მივანიჭოთ ჩვენთვის სასურველი მნიშვნელობები:

```
int Funqcia(int par1 = 5, int par2 = 20);
```

ეს ფუნქცია სამი გზით შეგვიძლია გამოვიძახოთ:

```
int shedegi1 = Funqcia();
```

```
int shedegi2 = Funqcia(7);
```

```
int shedegi3 = Funqcia(2, 8);
```

პირველ შემთხვევაში, არგუმენტები არ არის მითითებული და იგულისხმება ფუნქციის გამოცხადებისას განსაზღვრული მნიშვნელობები - 5 და 20. მეორე შემთხვევაში, მითითებულია ერთი არგუმენტი, რომელიც par1 პარამეტრს მიენიჭება, par2 პარამეტრს კი - მიენიჭება მნიშვნელობა 20. მესამე შემთხვევაში, par1 პარამეტრს მიენიჭება მნიშვნელობა 2, par2 პარამეტრს კი - მნიშვნელობა 8.

უნდა გვახსოვდეს, რომ თუ პირველი პარამეტრი მოიცემა გაჩუმებით, მაშინ ყველა დანარჩენი პარამეტრიც გაჩუმებით უნდა მოიცეს. მაგალითად, არასწორია ფუნქციის ასეთი გამოცხადება:

```
int Funqcia(int par1 = 5, int par2);
```

სწორია ფუნქციის შემდეგი გამოცხადება:

```
int Funqcia(int par1, int par2 = 9);
```

მოცემული პროგრამით ხდება ნაგულისხმევ პარამეტრებთან მუშობის დემონსტრირება.

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
// ნაგულისხმევ პარამეტრებთან მუშობის დემონსტრირება
```

```
int Funqcia(int par1 = 5, int par2 = 10)
```

```
{
```

```
    return par1 + par2;
```

```
}
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
int cvladi1, cvladi2;
```

```
int shedegi1, shedegi2, shedegi3;
```

```
cin << cvladi1 << cvladi2;
```

```
shedegi1 = Funqcia();
```

```
shedegi2 = Funqcia(cvladi1);
```

```
shedegi3 = Funqcia(cvladi1, cvladi2);
```

```
cout << shedegi1 << endl;
```

```
cout << shedegi2 << endl;
```

```
cout << shedegi3 << endl;
```

```
system("pause");
```

```
return 0;
```

}

თავი 8. კლასები, ობიექტები და მეთოდები

კლასის გამოცხადება

კლასი არის ობიექტზე ორიენტირებული დაპროგრამების ძირითადი სტრუქტურა. იგი მომხმარებლის მიერ შექმნილი ტიპია და წარმოადგენს ობიექტის შექმნის მექანიზმს. მის საფუძველზე იქმნება ობიექტი - ავტომობილი, თვითმფრინავი, მაღაზია, გეომეტრიული ფიგურა, სტუდენტი, ლექტორი, უნივერსიტეტი, ავადმყოფი და ა.შ. ობიექტი კლასის ეგზემპლარია. განსხვავება კლასსა და ობიექტს შორის ისაა, რომ კლასი არის ლოგიკური აბსტრაქცია, ობიექტი კი - ამ კლასის კონკრეტული რეალიზება კომპიუტერის მეხსიერებაში.

კლასში შემაჯალ ფუნქციას, ცვლადს, მასივს, მიმთითებელსა და კლასის სხვა ობიექტს კლასის წევრი ეწოდება. კლასი შეიძლება შეიცავდეს, აგრეთვე, სტატიკურ წევრებს, კონსტრუქტორებსა და დესტრუქტორებს. ამ ეტაპზე განვიხილავთ კლასის ძირითად წევრებს - ცვლადებს და ფუნქციებს.

კლასის გამოცხადების სინტაქსია:

```
მიმართვის_მოდიფიკატორი class კლასის_სახელი
{
// კლასის დახურული ცვლადები და ფუნქციები
public :
// კლასის ღია ცვლადები და ფუნქციები
} [ობიექტების_სია];
```

ობიექტების_სია შეიცავს მოცემული კლასის ტიპის მქონე ობიექტების სახელებს.

კორექტულად დაწერილ პროგრამაში კლასი განსაზღვრავს მხოლოდ ერთ ლოგიკურ ერთეულს. მაგალითად, კლასი, რომელიც შეიცავს ინფორმაციას ავტომანქანების შესახებ, არ უნდა შეიცავდეს სხვა ინფორმაციას, მაგალითად, შენობების ან ცხოველების შესახებ.

კლასი ორ ძირითად ფუნქციას ასრულებს, რომელიც მონაცემების ინკაფსულაციას უზრუნველყოფს. პირველი, ის მონაცემებს აკავშირებს კოდთან, რომელიც მათზე მანიპულირებს და მეორე, კლასი უზრუნველყოფს კლასის წევრებთან მიმართვის საშუალებებს. *ინკაფსულაცია* არის კლასის წევრებთან მიმართვის უფლებამოსილების გამიჯვნა, ანუ კლასის ზოგიერთ წევრს შეგვიძლია მივმართოთ კლასის გარეთ განსაზღვრული კოდიდან, ზოგიერთს კი - არა.

კლასის შიგნით გამოცხადებულ ცვლადებს და ფუნქციებს ამ კლასის *წევრები* (members) ეწოდება. ავტომატურად (ნაგულისხმევად), კლასში გამოცხადებული ცვლადები და ფუნქციები დახურულია (პრივატულია, private), ამიტომ მათი გამოცხადებისთვის არ არის აუცილებელი private სიტყვის გამოყენება. ღია (public) ცვლადებისა და ფუნქციების გამოცხადებისთვის public სიტყვა გამოიყენება.

კლასის ღია წევრებთან მიმართვა შეიძლება შესრულდეს ამ კლასის გარეთ განსაზღვრული კოდიდან. კლასის დახურულ წევრებთან მიმართვა შეუძლია მხოლოდ ამავე კლასის ფუნქციებს. ამ კლასის გარეთ განსაზღვრულ კოდს მხოლოდ ამავე კლასის ღია ფუნქციების გამოყენებით შეუძლია მიმართოს ამ კლასის დახურულ წევრებს.

კლასის წევრებთან მიმართვის შეზღუდვა არის ობიექტზე ორიენტირებული დაპროგრამების ერთ-ერთი ძირითადი პრინციპი. ის ობიექტებს იცავს არასწორი გამოყენებისგან. კლასის დახურულ წევრებთან მიმართვის უფლებას ვაძლევთ მხოლოდ კლასის შიგნით განსაზღვრულ ფუნქციებს, რითაც თავიდან ვიცილებთ მონაცემებისთვის არაკორექტული მნიშვნელობების მინიჭებას კლასის გარეთ განსაზღვრული კოდიდან.

შევქმნათ Samkutxedi კლასი და მასში მოვათავსოთ სამი ღია ცვლადი - სამკუთხედის გვერდები და ორი დახურული ცვლადი - პერიმეტრი და ფართობი:

```
class Samkutxedi
{
int perimetri;           // დახურული ცვლადები
double fartobi;
public:
int gverdi1;            // ღია ცვლადები
int gverdi2;
int gverdi3;
};
```

ეს კლასი ასეც შეგვიძლია ჩავწეროთ:

```
class Samkutxedi
{
private:
int perimetri;          // დახურული ცვლადები
double fartobi;
public:
int gverdi1;           // ღია ცვლადები
int gverdi2;
int gverdi3;
};
```

როგორც მაგალითიდან ჩანს, ჩვენ შეგვიძლია პირდაპირ მივმართოთ სამკუთხედის გვერდებს, რადგან ისინი ღია წევრებია. რადგან perimetri და fartobi დახურული ცვლადებია, ამიტომ მათთვის მნიშვნელობების მინიჭება შეუძლია მხოლოდ ამავე კლასში განსაზღვრულ ფუნქციებს. თუ perimetri-ს გამოვაცხადებთ ღია ცვლადად, მაშინ შეიძლება დავუშვათ შეცდომა. კერძოდ, თუ გვერდებს მივანიჭებთ მნიშვნელობები 1, 5 და 8, მაშინ perimetri ცვლადის მნიშვნელობა უნდა იყოს 14. თუ perimetri ღია ცვლადია, მაშინ ჩვენ შეგვიძლია მას ნებისმიერი მნიშვნელობა მივანიჭოთ, მაგალითად, 20. ამ შემთხვევაში მივიღებთ არასწორ სამკუთხედს. ამრიგად, პროგრამაში perimetri და fartobi ცვლადები იმდენად არის პრივატული, რომ დაცული იყოს ობიექტის მთლიანობა. თუ მათ გამოვაცხადებთ ღია ცვლადად, მაშინ იარსებებს იმის საფრთხე, რომ ამ ცვლადებს არასწორი მნიშვნელობა მიენიჭოთ და შედეგად, სამკუთხედის გვერდების ჯამი შეიძლება არ დაემთხვეს პერიმეტრის მნიშვნელობას. იგივე ეხება სამკუთხედის ფართობს.

class სიტყვის გამოყენებით იქმნება მონაცემების ახალი ტიპი, რომლის სახელია Samkutxedi. მას გამოვიყენებთ Samkutxedi ტიპის ობიექტის შესაქმნელად. უნდა გვახსოვდეს, რომ class სიტყვის გამოყენებით კლასის გამოცხადებისას ხდება მხოლოდ მისი აღწერა, მაგრამ ფიზიკურად ობიექტი ამ დროს არ იქმნება. Samkutxedi ტიპის ობიექტი იქმნება შემდეგნაირად: Samkutxedi Sam1;

ამ დროს მეხსიერებაში იქმნება Samkutxedi კლასის ეგზემპლარი (instance) - Sam1 ობიექტი. Sam1 ობიექტს ექნება Samkutxedi კლასის ყველა წევრის ასლი.

ამრიგად, ობიექტი შეიცავს კლასის თითოეული წევრის ასლს. ობიექტის წევრებთან მიმართვისთვის ვიყენებთ წერტილი (.) ოპერატორს. მისი სინტაქსია:

ობიექტის_სახელი.წევრის_სახელი

იმისათვის, რომ Sam1 ობიექტის gverdi2 ცვლადს მივანიჭოთ მნიშვნელობა 20, მინიჭების ოპერატორი შემდეგნაირად უნდა დავწეროთ:

```
Sam1.gverdi2 = 20;
```

კლასებთან მუშაობისას უნდა გვახსოვდეს, რომ კლასის აღწერა უმჯობესია ცალკე ფაილში მოვათავსოთ. თუმცა მისი მოთავსება შეიძლება იმ ფაილის დასაწყისში, რომელშიც ძირითადი პროგრამაა,

```
int _tmain(int argc, _TCHAR* argv[])
```

```
სტრიქონის წინ და
```

```
using namespace std;
```

სტრიქონის შემდეგ. შემდგომში, მთელ წიგნში, კლასის აღწერა ხან პროგრამის კოდთან ერთად იქნება მოთავსებული (ერთფაილიანი პროექტი), ხან ცალკე ფაილში (მრავალფაილიანი პროექტი).

ჯერ შევქმნათ ერთფაილიანი პროექტი. Samkutxedi კლასის აღწერა შევიტანოთ using namespace std; სტრიქონის შემდეგ:

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
class Samkutxedi
```

```
{
```

```
private:
```

```
    int perimetri;           // დახურული ცვლადები
```

```
    double fartobi;
```

```
public:
```

```
    int gverdi1;           // ღია ცვლადები
```

```
    int gverdi2;
```

```
    int gverdi3;
```

```
};
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
    system("pause");
```

```
    return 0;
```

```
}
```

შემდეგ, შევიტანოთ ძირითადი პროგრამა:

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
class Samkutxedi
```

```
{
```

```
private:
```

```
    int perimetri;           // დახურული ცვლადები
```

```
    double fartobi;
```

```
public:
```

```
    int gverdi1;           // ღია ცვლადები
```

```
    int gverdi2;
```

```
    int gverdi3;
```

```
};
```

```

int _tmain(int argc, _TCHAR* argv[])
{
    //      ობიექტის ცვლადებთან მუშაობის დემონსტრირება
    Samkutxedi Sam1;
    int perimetri;
    double fartobi;
    //      Sam1 ობიექტის ცვლადებს ენიჭებათ მნიშვნელობები
    cin >> Sam1.gverdi1 >> Sam1.gverdi2 >> Sam1.gverdi3;
    //      სამკუთხედის პერიმეტრის გამოთვლა
    perimetri = Sam1.gverdi1 + Sam1.gverdi2 + Sam1.gverdi3;
    fartobi = ( Sam1.gverdi1 * Sam1.gverdi2 ) / 2;
    cout << "samkutxedis perimetri = " << perimetri << endl;
    cout << "samkutxedis fartobi = " << fartobi << endl;

    system("pause");
    return 0;
}

```

პროგრამაში გამოცხადებულია perimetri და fartobi ცვლადები. კლასში გამოცხადებულია ამავე სახელის მქონე პრივატული perimetri და fartobi ცვლადები. მაგრამ, პროგრამაში გამოცხადებული ცვლადები და კლასში გამოცხადებული ცვლადები სხვადასხვა ცვლადებია და მეხსიერების სხვადასხვა უბანშია მოთავსებული. ჯერ-ჯერობით, კლასის პრივატულ ცვლადთან მიმართებას ვერ შევძლებთ. ამას მოვახერხებთ მეთოდების (ფუნქციების) შესწავლის შემდეგ.

როგორც აღვნიშნეთ, ობიექტების შექმნის ძირითადი პრინციპია ის, რომ თითოეულ ობიექტს აქვს კლასის ცვლადების საკუთარი ასლი. შედეგად, ერთი ობიექტის ცვლადები დამოუკიდებელია მეორე ობიექტის ცვლადებისაგან, ამიტომ მათი მნიშვნელობები შეიძლება ერთმანეთისაგან განსხვავდებოდეს. მაგალითად, თუ არსებობს Samkutxedi კლასის ორი ობიექტი, მაშინ თითოეულ მათგანს ექნება gverdi1, gverdi2, gverdi3, perimetri და fartobi ცვლადების საკუთარი ასლები, რომლებიც მეხსიერების სხვადასხვა უბანში იქნება მოთავსებული და შესაბამისად, მათი მნიშვნელობები შეიძლება ერთმანეთისგან განსხვავდებოდეს. ქვემოთ მოყვანილი პროგრამით ხდება ამ პრინციპის დემონსტრირება.

```

//      ორი ობიექტის ცვლადებთან მუშაობის დემონსტრირება
#include "stdafx.h"
#include "iostream"
using namespace std;

class Samkutxedi
{
    int perimetri;                //      დახურული ცვლადები
    double fartobi;
public : int gverdi1;            //      ღია ცვლადები
    int gverdi2;
    int gverdi3;
};

int _tmain(int argc, _TCHAR* argv[])
{
    Samkutxedi Sam1;                //      იქმნება Sam1 ობიექტი
}

```

```

Samkutxedi Sam2; // იქმნება Sam2 ობიექტი
int Sam1_perimetri, Sam2_perimetri;

// Sam1 ობიექტის ცვლადებს ენიჭება მნიშვნელობები
cin >> Sam1.gverdi1 >> Sam1.gverdi2 >> Sam1.gverdi3;

// Sam2 ობიექტის ცვლადებს ენიჭება მნიშვნელობები
cin >> Sam2.gverdi1 >> Sam2.gverdi2 >> Sam2.gverdi3;

// თითოეული სამკუთხედის პერიმეტრის გამოთვლა
Sam1_perimetri = Sam1.gverdi1 + Sam1.gverdi2 + Sam1.gverdi3;
Sam2_perimetri = Sam2.gverdi1 + Sam2.gverdi2 + Sam2.gverdi3;
cout << Sam1_perimetri << endl;
cout << Sam2_perimetri << endl;

system("pause");
return 0;
}

```

ობიექტები შეგვიძლია კლასის აღწერისას გამოვაცხადოთ. მოცემულ პროგრამაში კლასის აღწერის შემდეგ გამოცხადებულია Sam1 და Sam2 ობიექტები.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

class Samkutxedi
{
    int perimetri; // დახურული ცვლადები
    double fartobi;
public : int gverdi1; // ღია ცვლადები
        int gverdi2;
        int gverdi3;
} Sam1, Sam2;

int _tmain(int argc, _TCHAR* argv[])
{
int Sam1_perimetri, Sam2_perimetri;

// Sam1 ობიექტის ცვლადებს ენიჭება მნიშვნელობები
cin >> Sam1.gverdi1 >> Sam1.gverdi2 >> Sam1.gverdi3;

// Sam2 ობიექტის ცვლადებს ენიჭება მნიშვნელობები
cin >> Sam2.gverdi1 >> Sam2.gverdi2 >> Sam2.gverdi3;

// თითოეული სამკუთხედის პერიმეტრის გამოთვლა
Sam1_perimetri = Sam1.gverdi1 + Sam1.gverdi2 + Sam1.gverdi3;
Sam2_perimetri = Sam2.gverdi1 + Sam2.gverdi2 + Sam2.gverdi3;
cout << Sam1_perimetri<< endl;

```

```

cout << Sam2_perimetri<< endl;

system("pause");
return 0;
}

```

ობიექტზე მიმთითებელი

აქამდე, ობიექტის წევრებს მივმართავდით წერტილი (.) ოპერატორის საშუალებით. ობიექტის წევრებს შეგვიძლია მივმართოთ მიმთითებლის საშუალებითაც. ამ შემთხვევაში, გამოიყენება ისარი (->) ოპერატორი. ობიექტზე მიმთითებელი განისაზღვრება ისევე, როგორც ნებისმიერი ტიპის ცვლადზე მიმთითებელი. ამისათვის კლასის სახელის შემდეგ ვუთითებთ ობიექტის სახელს, რომლის წინ ვარსკვლავია (*) მოთავსებული. მაგალითად,

```

Samkutexdi *obj1;

ობიექტის მისამართის მისაღებად, მისი სახელის წინ უნდა დავწეროთ & ოპერატორი:
Samkutexdi obj1;
Samkutexdi *misamarti;
misamarti = &obj1;

```

თუ ობიექტზე მიმთითებლის მნიშვნელობას 1-ით გავზრდით, მაშინ ის მომდევნო ობიექტს მიმართავს.

მოცემული პროგრამით ხდება ობიექტზე მიმთითებელთან მუშაობის დემონსტრირება.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

// ობიექტზე მიმთითებელთან მუშაობის დემონსტრირება

```

class ChemiKlasi
{
public :
    int ricxvi1;
    int ricxvi2;
};

```

```

int _tmain(int argc, _TCHAR* argv[])

```

```

{
    ChemiKlasi obj1;           // ობიექტის შექმნა
    ChemiKlasi *mimtitebeli;  // მიმთითებლის გამოცხადება

```

```

    mimtitebeli = &obj1;      // მიმთითებლისთვის obj1 ობიექტის მისამართის მინიჭება
    cin >> mimtitebeli->ricxvi1;
    cin >> obj1.ricxvi2;
    cout << mimtitebeli->ricxvi1<< endl;
    cout << obj1.ricxvi2<< endl;

```

```

system("pause");
return 0;

```

```
}
```

პროგრამაში ChemiKlasi *mimtitebeli; გამოცხადება ქმნის ChemiKlasi ტიპის ობიექტზე მიმთითებელს. ამ დროს ობიექტი არ იქმნება. mimtitebeli მიმთითებელს obj1 ობიექტის მისამართს შემდეგნაირად ვანიჭებთ:

```
mimtitebeli = &obj1;
```

მიმთითებლის საშუალებით obj1 ობიექტის წევრთან მიმართვისთვის ისარი (->) ოპერატორი გამოიყენება:

```
cin >> mimtitebeli->ricxvi1;
```

ობიექტების მინიჭება

თუ ორ ობიექტს ერთნაირი ტიპი აქვს, მაშინ ერთი ობიექტი შეიძლება მეორეს მივანიჭოთ. ამ დროს, ობიექტს ენიჭება მეორე ობიექტის ყველა წევრის მნიშვნელობების ასლი. ეს ეხება მასივსაც. მიუხედავად იმისა, რომ ორივე ობიექტი ერთნაირი იქნება, ისინი მაინც სხვადასხვა ობიექტებად რჩება და მეხსიერების სხვადასხვა უბანს იკავებს. მაგალითი:

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
// ერთი ტიპის მქონე ობიექტების მინიჭება
```

```
class ChemiKlasi
```

```
{
```

```
public:
```

```
    int mteli;
```

```
    double wiladi;
```

```
    bool logikuri;
```

```
    int masivi[5];
```

```
};
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
int indexi;
```

```
ChemiKlasi obj1;
```

```
ChemiKlasi obj2;
```

```
// obj1 ობიექტის წევრებს ენიჭება მნიშვნელობები
```

```
cin >> obj1.mteli >> obj1.wiladi >> obj1.logikuri;
```

```
for (indexi = 0; indexi < 5; indexi++)
```

```
    obj1.masivi[indexi] = indexi + 10;
```

```
// obj2 ობიექტს ენიჭება obj1 ობიექტი
```

```
obj2 = obj1;
```

```
// obj2 ობიექტის წევრების ეკრანზე გამოტანა
```

```
cout << obj2.mteli << " " << obj2.wiladi << " " << obj2.logikuri << endl;
```

```
for (indexi = 0; indexi < 5; indexi++)
```

```
    cout << obj2.masivi[indexi] << " ";
```

```
cout << endl;
```

```

system("pause");
return 0;
}

```

მეთოდი

ფუნქცია შეიძლება იყოს ან არ იყოს კლასის წევრი. ფუნქცია, რომელიც არ არის კლასის წევრი, არის გლობალური. კლასის შიგნით გამოცხადებულ ფუნქციას **მეთოდი** ეწოდება. მას **ფუნქცია-წევრსაც** (*member function*) უწოდებენ. კლასის შიგნით შეიძლება გამოვაცხადოთ ფუნქციის პროტოტიპი.

ფუნქციის კოდი (ტანი) შეიძლება განსაზღვრული იყოს როგორც კლასის შიგნით, ისე კლასის გარეთ. კლასის შიგნით განსაზღვრული ფუნქციის სინტაქსია:

მიმართვის_მოდულიკატორი : შედეგის_ტიპი ფუნქციის_სახელი(პარამეტრების_სია)

```

{
ფუნქციის კოდი
}

```

აქ **მიმართვის_მოდულიკატორი** იღებს public მნიშვნელობას. თუ ის არ არის მითითებული, მაშინ ფუნქცია დახურულია და შესაბამისად, მისაწვდომი იქნება მხოლოდ იმ კლასის შიგნით, რომელშიც ის არის განსაზღვრული.

ზემოთ მოცემულ მაგალითებში სამკუთხედის პერიმეტრის გამოთვლა სრულდებოდა ძირითად პროგრამაში. პერიმეტრი უმჯობესია გამოითვალოს უშუალოდ კლასის შიგნით, რადგან მისი მნიშვნელობა დამოკიდებულია სამკუთხედის გვერდების ზომებზე. სამივე ეს სიდიდე კი ინკაფსულირებულია Samkutexedi კლასში. გარდა ამისა, Samkutexedi კლასისთვის იმ ფუნქციის დამატება, რომელშიც სრულდება პერიმეტრის გამოთვლა აუმჯობესებს ამ კლასის ობიექტზე ორიენტირებულ სტრუქტურას. მაგალითი:

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

// სამკუთხედის პერიმეტრის გამოთვლა კლასის შიგნით გამოცხადებულ ფუნქციაში

```
class Samkutexedi
```

```
{
    int perimetri;
    double fartobi;

```

```
public :
```

```
    int gverdi1;
    int gverdi2;
    int gverdi3;

```

// perimetri ფუნქცია განსაზღვრულია Samkutexedi კლასში

```
void Perimetri()
```

```
{
    perimetri = gverdi1 + gverdi2 + gverdi3;
    cout << "Perimetri = " << perimetri << endl;
}

```

```
};

```

```

int _tmain(int argc, _TCHAR* argv[])
{
Samkutxedi Sam1;
Samkutxedi Sam2;
//      Sam1 ობიექტის ცვლადებს ენიჭება მნიშვნელობები
cin >> Sam1.gverdi1 >> Sam1.gverdi2 >> Sam1.gverdi3;
//      Sam2 ობიექტის ცვლადებს ენიჭება მნიშვნელობები
cin >> Sam2.gverdi1 >> Sam2.gverdi2 >> Sam2.gverdi3;
//      პერიმეტრის გამოთვლა პირველი სამკუთხედისათვის
Sam1.Perimetri();
//      პერიმეტრის გამოთვლა მეორე სამკუთხედისათვის
Sam2.Perimetri();

system("pause");
return 0;
}

```

Perimetri() ფუნქცია განსაზღვრულია როგორც public, რაც იძლევა მისი გამოძახების შესაძლებლობას პროგრამის ნებისმიერი ადგილიდან. void სიტყვა მიუთითებს იმას, რომ Perimetri() ფუნქცია არ აბრუნებს (არ გასცემს) მნიშვნელობას. შემდეგ, ფიგურულ ფრჩხილებში მოთავსებულია ფუნქციის კოდი, რომელშიც გამოითვლება სამკუთხედის პერიმეტრი. მისი მნიშვნელობა ეკრანზე გამოაქვს cout ფუნქციას. რადგან, Samkutxedi ტიპის თითოეულ ობიექტს აქვს gverdi1, gverdi2 და gverdi3 ცვლადების საკუთარი ასლები, ამიტომ Perimetri() ფუნქციის გამოძახებისას პერიმეტრის გამოსათვლელად გამოყენებული იქნება გამომძახებელი ობიექტის ცვლადები.

Sam1.Perimetri(); სტრიქონში გამომძახებელია Sam1 ობიექტი. აქ ხდება Sam1 ობიექტის Perimetri() ფუნქციის გამოძახება, რის შემდეგ მართვა ამ ფუნქციას გადაეცემა. ის Sam1 ობიექტის ცვლადებთან იმუშავებს. როდესაც ფუნქცია მუშაობას დაამთავრებს, მართვა გადაეცემა პროგრამის იმ ადგილს, საიდანაც მისი გამოძახება მოხდა. პროგრამის შესრულება გაგრძელდება კოდის იმ სტრიქონიდან, რომელიც მოსდევს ფუნქციის გამომძახებელ სტრიქონს. ჩვენ შემთხვევაში, Sam1.Perimetri() ფუნქციის გამოძახება გამოიწვევს პერიმეტრის მნიშვნელობის გამოტანას Sam1 სამკუთხედისათვის, Sam2.Perimetri() ფუნქციის გამოძახება კი - პერიმეტრის მნიშვნელობის გამოტანას Sam2 სამკუთხედისათვის.

თუ ფუნქციის კოდი გვინდა განვსაზღვროთ კლასის გარეთ, მაშინ კლასის სახელის შემდეგ უნდა მოვათავსოთ "::" სიმბოლოები და ფუნქციის სახელი. "::" სიმბოლოებს **ხილვადობის უბნის გაფართოების ოპერატორი** ეწოდება. ფუნქციის განსაზღვრა შეგვიძლია მოვათავსოთ კლასის განსაზღვრის შემდეგ იმავე ფაილში:

```

#include "stdafx.h"
#include "iostream"
using namespace std;

//      კლასის გარეთ ფუნქციის გამოცხადების დემონსტრირება
class Samkutxedi
{
int perimetri;
double fartobi;
public :
int gverdi1;

```



```

int gverdi2;
int gverdi3;
// Perimetri ფუნქციის პროტოტიპის გამოცხადება
void Perimetri();
};
// Perimetri() ფუნქცია განსაზღვრულია Samkutxedi კლასის გარეთ
void Samkutxedi::Perimetri()
{
perimetri = gverdi1 + gverdi2 + gverdi3;
cout << "perimetri = " << perimetri << endl;
}

int _tmain(int argc, _TCHAR* argv[])
{
Samkutxedi Sam1;
Samkutxedi Sam2;
// Sam1 ობიექტის ცვლადებს ენიჭება მნიშვნელობები
cin >> Sam1.gverdi1 >> Sam1.gverdi2 >> Sam1.gverdi3;
// პერიმეტრის გამოთვლა პირველი სამკუთხედისათვის
Sam1.Perimetri();
// Sam2 ობიექტის ცვლადებს ენიჭება მნიშვნელობები
cin >> Sam2.gverdi1 >> Sam2.gverdi2 >> Sam2.gverdi3;
// პერიმეტრის გამოთვლა მეორე სამკუთხედისათვის
Sam2.Perimetri();

system("pause");
return 0;
}

```

კლასის შიგნით public void Perimetri(); სტრიქონში ხდება Perimetri() ფუნქციის პროტოტიპის გამოცხადება. რადგან, Perimetri() ფუნქცია განსაზღვრულია კლასის გარეთ, ამიტომ მისი სახელის წინ ვწერთ კლასის სახელს და „:“ ოპერატორს.

ფუნქციიდან მართვის დაბრუნება

ახლა Perimetri() ფუნქცია გადავაკეთოთ ისე, რომ მან გამოთვალოს პერიმეტრი და დაუბრუნოს ის გამომძახებელ პროგრამას. ასეთი მიდგომის ერთ-ერთი უპირატესობაა ის, რომ დაბრუნებული მნიშვნელობა შეგვიძლია გამოვიყენოთ გამოთვლებში. ქვემოთ მოცემულ პროგრამაში Perimetri() ფუნქციას ეკრანზე აღარ გამოაქვს პერიმეტრის მნიშვნელობა, არამედ ახდენს მის გამოთვლას და შედეგის დაბრუნებას.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// return ოპერატორის გამოყენების დემონსტრირება
class Samkutxedi
{

```

```

        int perimetri;
public : int gverdi1;
        int gverdi2;
        int gverdi3;
//      ფუნქციის განსაზღვრა
int Perimetri()
{
    perimetri = gverdi1 + gverdi2 + gverdi3;
    return perimetri;          //      მნიშვნელობის დაბრუნება
}
};

int _tmain(int argc, _TCHAR* argv[])
{
//      tolgerda და tolferda ობიექტების შექმნა
Samkutxedi tolgerda;
Samkutxedi tolferda;
int tolgerda_perimetri, tolferda_perimetri;

//      tolgerda ობიექტის ცვლადებს მნიშვნელობები ენიჭება
cin >> tolgerda.gverdi1 >> tolgerda.gverdi2 >> tolgerda.gverdi3;
//      tolferda ობიექტის ცვლადებს მნიშვნელობები ენიჭება
cin >> tolferda.gverdi1 >> tolferda.gverdi2 >> tolferda.gverdi3;
//      tolgerda და tolferda სამკუთხედებისათვის პერიმეტრი გამოითვლება
tolgerda_perimetri = tolgerda.Perimetri();
tolferda_perimetri = tolferda.Perimetri();
cout << tolgerda_perimetri<< endl;
cout << tolferda_perimetri<< endl;

system("pause");
return 0;
}

```

პროგრამაში Perimetri() ფუნქციის მიერ გაცემული მნიშვნელობები შესაბამისად მიენიჭება tolgerda_perimetri და tolferda_perimetri ცვლადებს.

პარამეტრის გამოყენება

როგორც ვნახეთ, გამოძახებისას ფუნქციას შეგვიძლია გადავცეთ ერთი ან მეტი პარამეტრი. ქვემოთ მოცემულ პროგრამაში ფუნქციისათვის მნიშვნელობის გადასაცემად პარამეტრი გამოიყენება. ArisLuwi კლასის შიგნით განსაზღვრული luwi_kenti(int par1) ფუნქცია აბრუნებს bool ტიპის შედეგს. თუ მისთვის გადაცემული მნიშვნელობა არის ლუწი, მაშინ ის გასცემს true მნიშვნელობას, წინააღმდეგ შემთხვევაში კი - false მნიშვნელობას.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

//      პროგრამა განსაზღვრავს რიცხვი კენტია თუ ლუწი
class ArisLuwi
{
// ფუნქციის განსაზღვრა, რომელიც ამოწმებს რიცხვი კენტია თუ ლუწი
public :
bool luwi_kenti( int par1)
{
if ( (par1 % 2 ) == 0 ) return true;
    else return false;
}
};

int _tmain(int argc, _TCHAR* argv[])
{
int ricxvi;
ArisLuwi obieqti;

cin >> ricxvi;
if ( obieqti.luwi_kenti(ricxvi) )
cout << "ricxvi " << ricxvi << " lucia";
    else cout << "ricxvi " << ricxvi << " kentia" << endl;

system("pause");
return 0;
}

```

luwi_kenti() ფუნქციის გამოძახებისას ricxvi არგუმენტის მნიშვნელობა მიენიჭება par1 პარამეტრს.

კიდევ ერთი მაგალითი. Gamyofi კლასი შეიცავს arisGamyofi() ფუნქციას, რომელიც განსაზღვრავს არის თუ არა პირველი პარამეტრი მეორე პარამეტრის გამყოფი.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

//      პროგრამა განსაზღვრავს ერთი რიცხვი არის თუ არა მეორის გამყოფი
class Gamyofi
{
//      ფუნქცია ამოწმებს პირველი პარამეტრი უნაშთოდ იყოფა თუ არა მეორე პარამეტრზე
public : bool arisGamyofi(int par1, int par2)
{
if ( ( par1 % par2 ) == 0 ) return true;
    else return false;
}
};

int _tmain(int argc, _TCHAR* argv[])
{
Gamyofi obieqti;

```

```

int ricxvi1, ricxvi2;

cin >> ricxvi1 >> ricxvi2;
if ( obieqti.arisGamyofi(ricxvi1, ricxvi2) )
    cout << ricxvi1 << " aris " << ricxvi2 << "-is gamyofi" << endl;
else    cout << ricxvi1 << " ar aris " << ricxvi2 << "-is gamyofi" << endl;

system("pause");
return 0;
}

```

arisGamyofi() ფუნქციის გამოძახებისას ricxvi1 არგუმენტის მნიშვნელობა მიენიჭება par1 პარამეტრს, ricxvi2 არგუმენტის მნიშვნელობა კი - par2 პარამეტრს.

კონსტრუქტორი

კონსტრუქტორი (constructor) არის ფუნქცია, რომლის დანიშნულებაცაა ობიექტის ცვლადების ინიციალიზება. მისი გამოცხადების სინტაქსი ჩვეულებრივი ფუნქციის სინტაქსის მსგავსია. კონსტრუქტორი ყოველთვის გამოიძახება ობიექტის შექმნისას. ჩვეულებრივი ფუნქციისგან განსხვავებით:

- კონსტრუქტორს ისეთივე სახელი აქვს, როგორც კლასს.
 - დასაბრუნებელი მნიშვნელობის ტიპი კონსტრუქტორში არ ეთითება.
- კონსტრუქტორის სინტაქსია:

კლასის_სახელი()

```

{
    კონსტრუქტორის_კოდი
}

```

როგორც წესი, კონსტრუქტორები გამოიყენება ობიექტის ცვლადებისთვის საწყისი მნიშვნელობების მისანიჭებლად, ან ინიციალიზების ნებისმიერი სხვა პროცედურის შესასრულებლად, რომლებიც აუცილებელია ობიექტის შესაქმნელად.

ყველა კლასს აქვს კონსტრუქტორი მიუხედავად იმისა, ის განსაზღვრულია თუ არა. C++ ენაში გათვალისწინებულია კონსტრუქტორის არსებობა, რომელიც ობიექტის ყველა ცვლადს ანიჭებს ნულოვან (ეს ეხება ჩვეულებრივი ტიპის ცვლადებს) ან null მნიშვნელობას (ეს ეხება მიმართვითი ტიპის ცვლადებს). ასეთ კონსტრუქტორს **ნაგულისხმევი კონსტრუქტორი (ავტომატურად განსაზღვრული კონსტრუქტორი)** ეწოდება. ის იმ შემთხვევაში გამოიძახება, როდესაც კლასში კონსტრუქტორი განსაზღვრული არ არის. თუ კონსტრუქტორი აშკარადაა განსაზღვრული კლასში, მაშინ სწორედ ის იქნება გამოძახებული.

გლობალური ობიექტებისთვის კონსტრუქტორი გამოიძახება მაშინ, როდესაც პროგრამა მუშობას იწყებს, ლოკალური ობიექტებისთვის კი - ცვლადის გამოცხადებისას.

პროფესიულ დონეზე შედგენილ პროგრამაში ობიექტის ცვლადების ინიციალიზება ყოველთვის კონსტრუქტორის მიერ სრულდება. მაგალითად, აქამდე განხილულ პროგრამებში ობიექტის ცვლადებს მნიშვნელობებს ჩვენ ვანიჭებდით. უმჯობესია, ეს კონსტრუქტორმა გააკეთოს. მაგალითი:

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

// კონსტრუქტორთან მუშაობის დემონსტრირება
class Samkutxedi
{
int perimetri;
public : int gverdi1;
        int gverdi2;
        int gverdi3;
// კონსტრუქტორი პარამეტრებით
Samkutxedi(int par1, int par2, int par3)
{
gverdi1 = par1;
gverdi2 = par2;
gverdi3 = par3;
perimetri = gverdi1 + gverdi2 + gverdi3;
}
// ფუნქცია გასცემს პერიმეტრს
int Perimetris_Gacema()
{
return perimetri;
}
};

int _tmain(int argc, _TCHAR* argv[])
{
int gverdi1, gverdi2, gverdi3, gverdi4, gverdi5, gverdi6;

cin >> gverdi1 >> gverdi2 >> gverdi3 >> gverdi4 >> gverdi5 >> gverdi6;
Samkutxedi Sam1(gverdi1, gverdi2, gverdi3);
Samkutxedi Sam2(gverdi4, gverdi5, gverdi6);

int SamkutxedisPerimetri1 = Sam1.Perimetris_Gacema();
int SamkutxedisPerimetri2 = Sam2.Perimetris_Gacema();
cout << "pirveli samkutxedis perimetri = " << SamkutxedisPerimetri1 << endl;
cout << "meore samkutxedis perimetri = " << SamkutxedisPerimetri2 << endl;

system("pause");
return 0;
}

```

პროგრამაში Samkutxedi Sam1(gverdi1, gverdi2, gverdi3); სტრიქონში სრულდება Samkutxedi() კონსტრუქტორის გამოძახება. მას სამი პარამეტრი აქვს, რომლებიც გამოიყენება სამკუთხედის გვერდებისათვის მნიშვნელობების მისანიჭებლად. gverdi1, gverdi2 და gverdi3 არგუმენტების მნიშვნელობები, შესაბამისად par1, par2 და par3 პარამეტრებს მიენიჭება. ისინი თავის მხრივ gverdi1, gverdi2 და gverdi3 ცვლადებს ენიჭება.

მოცემული პროგრამით ხდება ობიექტების გამოცხადება კლასის აღწერისას.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

//      ობიექტების გამოცხადება კლასის განსაზღვრისას
class Samkutxedi
{
    int gverdi1;
    int gverdi2;
    int gverdi3;
    int perimetri;
public :
Samkutxedi(int par1, int par2, int par3)
{
gverdi1 = par1;
gverdi2 = par2;
gverdi3 = par3;
}
int Perimetris_Gacema()
{
perimetri = gverdi1 + gverdi2 + gverdi3;
return perimetri;
}
} Sam1(1, 2, 3), Sam2(11, 12, 13);

int _tmain(int argc, _TCHAR* argv[])
{
int gverdi1, gverdi2, gverdi3, gverdi4, gverdi5, gverdi6;

cin >> gverdi1 >> gverdi2 >> gverdi3 >> gverdi4 >> gverdi5 >> gverdi6;
Samkutxedi Sam1(gverdi1, gverdi2, gverdi3);
Samkutxedi Sam2(gverdi4, gverdi5, gverdi6);

int SamkutxedisPerimetri1 = Sam1.Perimetris_Gacema();
int SamkutxedisPerimetri2 = Sam2.Perimetris_Gacema();

cout << "pirveli samkutxedis perimetri = " << SamkutxedisPerimetri1 << endl;
cout << "meore samkutxedis perimetri = " << SamkutxedisPerimetri2 << endl;

system("pause");
return 0;
}

```

კონსტრუქტორმა კლასში გამოცხადებული ცვლადების ინიციალიზებისთვის შეიძლება **ინიციალიზების სია** გამოიყენოს. ეს სია ეთითება ":" სიმბოლოს შემდეგ, რომელიც კონსტრუქტორის პარამეტრების გამოცხადებას მოსდევს. მოცემულ პროგრამაში gverdi1 ცვლადს par1 პარამეტრის მნიშვნელობა ენიჭება, gverdi2 ცვლადს - par2 პარამეტრის მნიშვნელობა, gverdi3 ცვლადს კი - par3 პარამეტრის მნიშვნელობა.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

// ინიციალიზების სიის გამოყენების დემონსტრირება
class Samkutxedi
{
    int gverdi1;
    int gverdi2;
    int gverdi3;
    int perimetri;
public :
Samkutxedi(int par1, int par2, int par3) : gverdi1(par1), gverdi2(par2), gverdi3(par3)
{
perimetri = gverdi1 + gverdi2 + gverdi3;
}
int Perimetris_Gacema()
{
return perimetri;
}
};

int _tmain(int argc, _TCHAR* argv[])
{
int gverdi1, gverdi2, gverdi3;

cin >> gverdi1 >> gverdi2 >> gverdi3;
Samkutxedi obj(gverdi1, gverdi2, gverdi3);
cout << obj.Perimetris_Gacema() << endl;

system("pause");
return 0;
}

```

როდესაც კონსტრუქტორს ერთი არგუმენტი აქვს, მაშინ შეგვიძლია გამოვიყენოთ ინიციალიზების ორი ფორმა:

```
// პირველი ფორმა
```

```

{
...
ChemiKlasi obj(5);
...
}

```

და

```
// მეორე ფორმა
```

```

{
...
ChemiKlasi obj = 5;
...
}

```

ინიციალიზების მეორე ფორმის აზრი ის არის, რომ ერთარგუმენტანი კონსტრუქტორისთვის ის იძლევა ამ არგუმენტის ტიპის არაცხადი გარდაქმნის ორგანიზების

საშუალებას იმ კლასის ტიპად, რომელსაც კონსტრუქტორი ეკუთვნის. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

// კონსტრუქტორის მიერ ცვლადების ინიციალიზების
// ორივე ფორმის დემონსტრირება
class ChemiKlasi
{
    int cvladi;
public :
    ChemiKlasi(int par)
    {
        cvladi = par;
    }
    int Naxva()
    {
        return cvladi;
    }
};

int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi1, ricxvi2;
    char simbolo;

    cin >> ricxvi1 >> ricxvi2 >> simbolo;
    ChemiKlasi obj1(ricxvi1);           // ინიციალიზების პირველი ფორმა
    ChemiKlasi obj2 = ricxvi2;         // ინიციალიზების მეორე ფორმა

    cout << obj1.Naxva() << endl;
    cout << obj2.Naxva() << endl;
    // აქ სრულდება ტიპის ავტომატური გარდაქმნა
    obj1 = simbolo;
    cout << obj1.Naxva() << endl;

    system("pause");
    return 0;
}
```

კონსტრუქტორის გადატვირთვა

ფუნქციის მსგავსად შესაძლებელია კონსტრუქტორის გადატვირთვა, დესტრუქტორის გადატვირთვა კი - არ შეიძლება. კონსტრუქტორის გადატვირთვას შემდეგი უპირატესობები აქვს: მოქნილობის უზრუნველყოფა, ობიექტების მასივების ინიციალიზება და ა.შ.

კონსტრუქტორების გადატვირთვა საშუალებას გვაძლევს ავირჩიოთ ობიექტის ინიციალიზების საშუალება. მოცემული პროგრამით ხდება გადატვირთვად კონსტრუქტორთან

მუშაობის დემონსტრირება.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

// გადატვირთვადი კონსტრუქტორის დემონსტრირება
class ChemiKlasi1
{
public :
int x;
ChemiKlasi1()
{
x = 0;
}
ChemiKlasi1(int par1)
{
x = par1 * par1;
}
ChemiKlasi1(double par2)
{
x = (int) par2 + 10;
}
ChemiKlasi1(int par3, int par4)
{
x = par3 * par4;
}
};

int _tmain(int argc, _TCHAR* argv[])
{
int ricxvi;
ChemiKlasi1 obieqti1;

cin >> ricxvi;
cout << obieqti1.x << endl;
ChemiKlasi1 obieqti2(ricxvi);
cout << obieqti2.x << endl;
ChemiKlasi1 obieqti3(28.89);
cout << obieqti3.x << endl;
ChemiKlasi1 obieqti4(3, 5);
cout << obieqti4.x << endl;

system("pause");
return 0;
}
```

პროგრამაში მოცემულია ChemiKlasi() ფუნქციის ოთხი ვერსია. საჭირო მათგანის არჩევა ხდება არგუმენტების მიხედვით.

კონსტრუქტორი ნაგულისხმევი პარამეტრებით

ნაგულისხმევი პარამეტრები შეგვიძლია კონსტრუქტორებსაც გადავცეთ. შედეგად, შეგვიძლია თავიდან ავიცილოთ კონსტრუქტორის გადატვირთვა ინიციალიზებული და არაინიციალიზებული ობიექტების შექმნის მიზნით. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

// კონსტრუქტორი ნაგულისხმევი პარამეტრით
class Klasi3
{
int ricxvi;
public :
Klasi3(int par = 0)
{
    ricxvi = par;
}
int Gacema()
{
    return ricxvi;
}
};

int _tmain(int argc, _TCHAR* argv[])
{
Klasi3 obj1(5);
Klasi3 obj2;

cout << obj1.Gacema() << endl;
cout << obj2.Gacema() << endl;

system("pause");
return 0;
}
```

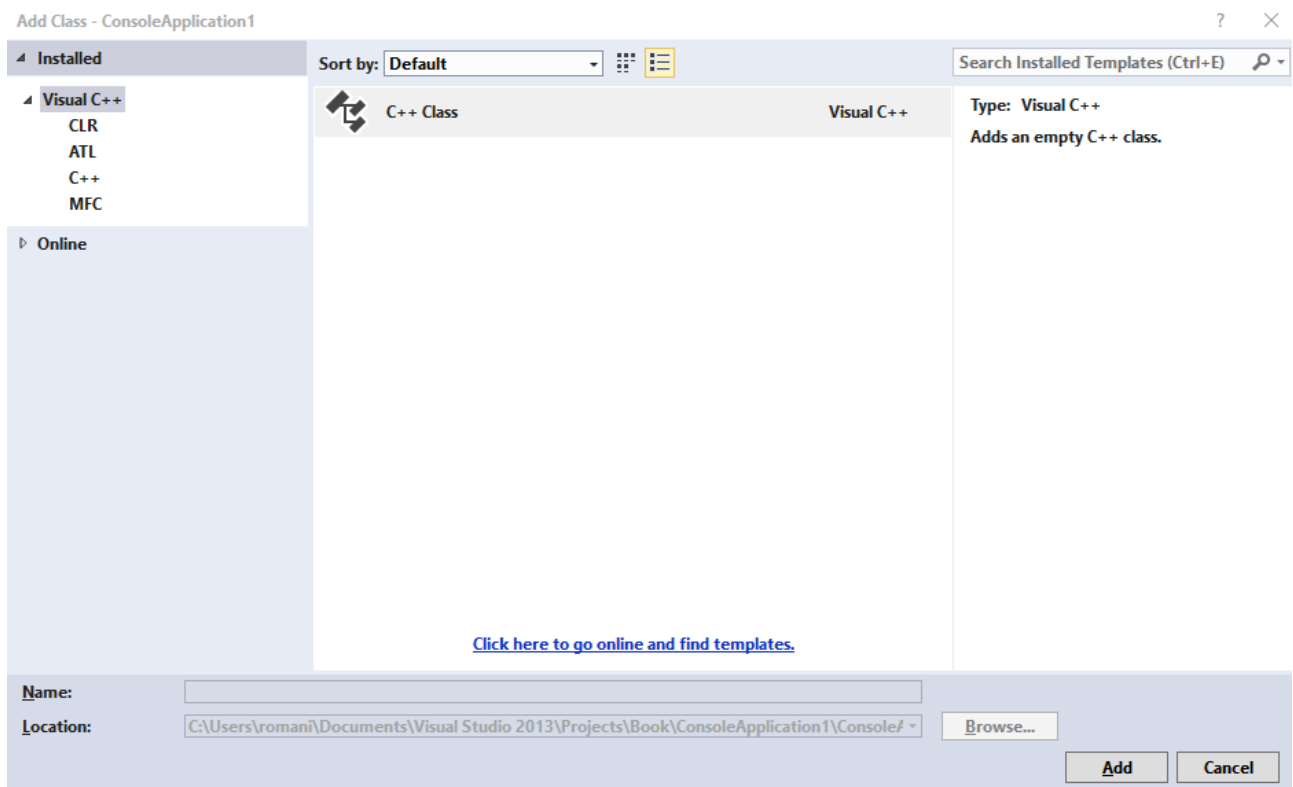
მრავალფაილიანი პროექტის შექმნა

ახლა შევქმნათ მრავალფაილიანი პროექტი. ამისათვის, ჯერ შევქმნათ ახალი პროექტი. შემდეგ, შევასრულოთ Project მენიუს Add Class ბრძანება. გახსნილ ფანჯარაში (ნახ. 8.1) ვაჭერთ Add კლავიშს. გაიხსნება Generic C++ Class Wizard ფანჯარა (ნახ. 8.2). Class name ველში შეგვაქვს კლასის სახელი - Martkutxedi. ეს სახელი ავტომატურად აისახება .h file და .cpp file ველებში. ეს იმას ნიშნავს, რომ შეიქმნება Martkutxedi.h და Martkutxedi.cpp ფაილები. ვაჭერთ Finish კლავიშს.

Martkutxedi.h ფაილს ექნება ნახ. 8.3-ზე ნაჩვენები სახე, Martkutxedi.cpp ფაილს კი - ნახ. 8.4-ზე ნაჩვენები სახე.

Martkutxedi კლასს დავუმატოთ ცვლადები, კონსტრუქტორი და მეთოდები. ამისათვის ვასრულებთ VIEW მენიუს Class View ბრძანებას. მთავარი ფანჯრის მარჯვენა ნაწილში გაიხსნება Class View ფანჯარა (ნახ. 8.5). Martkutxedi კლასის სახელზე ვაჭერთ თავის მარჯვენა კლავიშით, ვხსნით Add ქვემენიუს და ვასრულებთ Add Variable ბრძანებას. გახსნილი ფანჯრის Variable name ველში შეგვაქვს ცვლადის სახელი - sigrdze (ნახ. 8.6). პარამეტრის ტიპს ვირჩევს Variable type ჩამოშლადი სიიდან. ბოლოს ვაჭერთ Finish კლავიშს. ასეთივე გზით ვქმნით sigane, perimetri და fartobi ცვლადებს.

ახლა Martkutxedi კლასს დავუმატოთ კონსტრუქტორი და ორი ფუნქცია: Perimetri() და Fartobi(). ამისათვის, Class View ფანჯარაში Martkutxedi კლასის სახელზე ვაჭერთ თავის მარჯვენა კლავიშით, ვხსნით Add ქვემენიუს და ვასრულებთ Add Function ბრძანებას. გახსნილი ფანჯრის (ნახ. 8.7) Function name ველში შეგვაქვს ფუნქციის, ჩვენს შემთხვევაში კონსტრუქტორის, სახელი. Return type ველში ვშლით ტიპს, რადგან კონსტრუქტორი არანაირი ტიპის შედეგს არ გასცემს. Parameter name ველში შეგვაქვს პირველი პარამეტრის სახელი - par1. Parameter type ველში შეგვაქვს პარამეტრის ტიპი - int. ვაჭერთ Add კლავიშს და ეს პარამეტრი დაემატება Parameter list სიას (ნახ. 8.8). ასეთივე გზით კონსტრუქტორს დავუმატებთ par2 პარამეტრს. ბოლოს ვაჭერთ Finish კლავიშს. შედეგად, Martkutxedi.h ფაილს დაემატება კონსტრუქტორის პროტოტიპი (ნახ. 8.9), Martkutxedi.cpp ფაილს კი - თვითონ კონსტრუქტორი, სადაც შეგვაქვს მისი კოდი (ნახ. 8.10). ანალოგიურად კლასს დავუმატოთ Perimetri() და Fartobi() ფუნქციები. შესაბამისად, Martkutxedi.h და Martkutxedi.cpp ფაილები მიიღებენ ნახ. 8.11 და ნახ. 8.12-ზე ნაჩვენებ სახეს.



ნახ. 8.1. Add Class ფანჯარა.



Welcome to the Generic C++ Class Wizard

Class name: <input type="text" value="Martkutxedi"/>	.h file: <input type="text" value="Martkutxedi.h"/>cpp file: <input type="text" value="Martkutxedi.cpp"/> ...
Base class: <input type="text"/>	Access: <input type="text" value="public"/> ▼	<input type="checkbox"/> Virtual destructor <input type="checkbox"/> Inline <input type="checkbox"/> Managed

ნახ. 8.2. Generic C++ Class Wizard ფანჯარა.

```
Martkutxedi.h*  X Martkutxedi.cpp*  ConsoleApplication1.cpp  Start Page
Martkutxedi
#pragma once
class Martkutxedi
{
public:
    Martkutxedi();
    ~Martkutxedi();
};
```

ნახ. 8.3. Martkutxedi.h ფაილი.

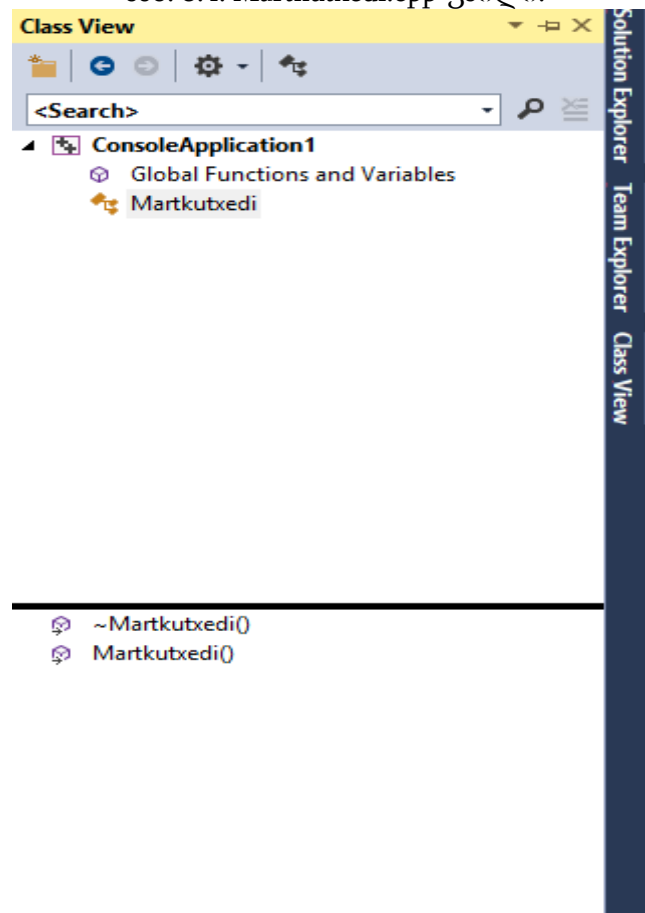
```
Martkutxedi.h*  Martkutxedi.cpp*  ConsoleApplication1.cpp  Start Page
→ Martkutxedi  Martkutxedi()

#include "stdafx.h"
#include "Martkutxedi.h"

Martkutxedi::Martkutxedi()
{
}

Martkutxedi::~Martkutxedi()
{
}
```

ნახ. 8.4. Martkutxedi.cpp ფაილი.



ნახ. 8.5. Class View ფანჯარა.

Welcome to the Add Member Variable Wizard



Access: public	<input type="checkbox"/> Control variable	
Variable type: int	Control ID: 	Category: Control
Variable name: sigrdze	Control type: 	Max chars:
	Min value: 	Max value:
	.h file: 	.cpp file:
Comment (// notation not required): 		
		Finish Cancel

ნახ. 8.6. პარამეტრის ტიპის არჩევა.

Welcome to the Add Member Function Wizard



Return type: 	Function name: Martkutxedi	
Parameter type: int	Parameter name: par1	Parameter list:
	<input type="button" value="Add"/> <input type="button" value="Remove"/>	
Access: public	<input type="checkbox"/> Static <input type="checkbox"/> Virtual <input type="checkbox"/> Pure	.cpp file: martkutxedi.cpp
	<input type="checkbox"/> Inline	
Comment (// notation not required): 		
Function signature: Martkutxedi()		
		Finish Cancel

ნახ. 8.7. ფუნქციის სახელის შეტანა.

Welcome to the Add Member Function Wizard



Return type: <input type="text" value=""/>	Function name: <input type="text" value="Martkutxedi"/>	
Parameter type: <input type="text" value="int"/>	Parameter name: <input type="text" value=""/>	Parameter list: <input type="text" value="int par 1"/>
Access: <input type="text" value="public"/>	<input type="checkbox"/> Static <input type="checkbox"/> Virtual <input type="checkbox"/> Pure <input type="checkbox"/> Inline	.cpp file: <input type="text" value="martkutxedi.cpp"/>
Comment (// notation not required): <input type="text" value=""/>		
Function signature: <input type="text" value="Martkutxedi(int par1)"/>		
		<input type="button" value="Finish"/> <input type="button" value="Cancel"/>

ნახ. 8.8. Parameter list სიაში პარამეტრის დამატება.

```

Martkutxedi.h  X  Martkutxedi.cpp*  ConsoleApplication1.cpp
Martkutxedi  Martkutxedi(int par1, int par2)
#pragma once
class Martkutxedi
{
public:
    Martkutxedi();
    ~Martkutxedi();

    int sigrdze;
    int sigane;
    int fartobi;
    int perimetri;
    Martkutxedi(int par1, int par2);
};
  
```

ნახ. 8.9. Martkutxedi.h ფაილს დაემატა კონსტრუქტორის პროტოტიპი.

```

Martkutxedi.h  Martkutxedi.cpp*  ConsoleApplication1.cpp
(Global Scope)
#include "stdafx.h"
#include "Martkutxedi.h"

Martkutxedi::Martkutxedi()
: sigrdze(0)
, sigane(0)
, fartobi(0)
, perimetri(0)
{
}

Martkutxedi::~Martkutxedi()
{
}

Martkutxedi::Martkutxedi(int par1, int par2)
{
    sigrdze = par1;
    sigane = par2;
}

```

ნახ. 8.10. Martkutxedi.cpp ფაილს დაემატა კონსტრუქტორი.

```

Martkutxedi.h*  Martkutxedi.cpp*  ConsoleApplication1.cpp
Martkutxedi  Martkutxedi(int par1, int par2)
#pragma once
class Martkutxedi
{
public:
    Martkutxedi();
    ~Martkutxedi();

    int sigrdze;
    int sigane;
    int fartobi;
    int perimetri;
    Martkutxedi(int par1, int par2);
    int Fartobi();
    int Perimetri();
};

```

ნახ. 8.11. Martkutxedi.h ფაილისთვის Perimetri() და Fartobi() ფუნქციების პროტორიპის დამატება.


```

Martkutxedi.h*   Martkutxedi.cpp*   ConsoleApplication1.cpp
→ Martkutxedi   Martkutxedi()
#include "stdafx.h"
#include "Martkutxedi.h"
Martkutxedi::Martkutxedi()
: sigrdze(0)
, sigane(0)
, fartobi(0)
, perimetri(0)
{ }
Martkutxedi::~~Martkutxedi()
{ }
Martkutxedi::Martkutxedi(int par1, int par2)
{
    sigrdze = par1;
    sigane = par2;
}
int Martkutxedi::Fartobi()
{
    fartobi = sigrdze * sigane;
    return fartobi;
}
int Martkutxedi::Perimetri()
{
    perimetri = 2 * ( sigrdze + sigane );
    return perimetri;
}

```

ნახ. 8.12. Martkutxedi.cpp ფაილისთვის Perimetri() და Fartobi() ფუნქციების კოდის დამატება.

```

Martkutxedi.h   Martkutxedi.cpp   ConsoleApplication1.cpp
(Global Scope)   _tmain(int argc, _TCHAR* argv[])
// ConsoleApplication1.cpp : Defines the entry point for the console application.
//
#include "stdafx.h"
#include "iostream"
#include "Martkutxedi.h"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int sigrdze, sigane, fartobi, perimetri;
    cin >> sigrdze >> sigane;
    Martkutxedi obj_1(sigrdze, sigane);
    fartobi = obj_1.Fartobi();
    perimetri = obj_1.Perimetri();
    cout << "fartobi = " << fartobi << "   perimetri = " << perimetri << endl;

    system("pause");
    return 0;
}

```

ნახ. 8.13. ძირითადი პროგრამა.

```
C:\Users\romani\Documents\Visual Studio 2013\Projects\Book\ConsoleA
2 3
fartobi = 6   perimetri = 10
Press any key to continue . . .
```

ნახ. 8.14. პროგრამის მუშაობის შედეგი.

იმისათვის, რომ შევძლოთ კლასის გამოყენება, ძირითადი პროგრამის ფაილს (ConsoleApplication.cpp) დასაწყისში, #include "iostream" დირექტივის შემდეგ, უნდა დავუმატოთ #include "Martkutxedi.h" დირექტივა. შევიტანოთ შესაბამისი კოდი (ნახ. 8.13). პროგრამის შესასრულებლად დავაჭიროთ კლავიატურის F5 კლავიშს ან ინსტრუმენტების პანელის **Local Windows Debugger** კლავიშს. შეგვაქვს მართკუთხედის სიგრძე და სიგანე და ვაჭერთ Enter კლავიშს. ეკრანზე გამოჩნდება მართკუთხედის ფართობისა და პერიმეტრის მნიშვნელობები (ნახ. 8.14).

კლასების შექმნა სხვა გზითაც შეიძლება. ცვლადებისა და ფუნქციების გამოცხადებისთვის არ არის აუცილებელი ნახ. 8.6, ნახ. 8.7 და ნახ. 8.8-ზე ნაჩვენები ფანჯრების გამოყენება. ვხსნით ნახ. 8.2-ზე ნაჩვენებ ფანჯარას. შეგვაქვს კლასის სახელი და ვაჭერთ Finish კლავიშს. გაიხსნება ნახ. 8.3 და ნახ. 8.4-ზე ნაჩვენები ფანჯრები. ნახ. 8.3-ზე ნაჩვენებ ფანჯარაში ხელით შეგვაქვს საჭირო ცვლადები და ფუნქციები, როგორც ეს ნახ. 8.11-ზეა ნაჩვენები. ნახ. 8.4-ზე ნაჩვენებ ფანჯარაში ხელით შეგვაქვს ფუნქციები ისე, როგორც ეს ნახ. 8.12-ზეა ნაჩვენები.

თავი 9. ფუნქციები უფრო დაწვრილებით.

ჩასადგმელი ფუნქცია

C++ ენაში შეგვიძლია გამოვაცხადოთ ფუნქცია, რომელიც არ გამოიძახება. პროგრამაში მისი კოდის მოთავსება ხდება გამოძახების ადგილში. *ჩასადგმელი ფუნქციის* (in-line) უპირატესობა იმაში მადგომარეობს, რომ ის დაკავშირებული არ არის ფუნქციების გამოძახებისა და მნიშვნელობის დაბრუნების მექანიზმთან. ამიტომ, ჩასადგმელი ფუნქციები სწრაფად სრულდება. ასეთი ფუნქციების ნაკლია ის, რომ თუ ისინი დიდი ზომისაა და ხშირად გვხვდება პროგრამაში, მაშინ პროგრამის ზომა მკვეთრად იზრდება. ამის გამო, ჩასადგმელი ფუნქციები ხშირად გამოიყენება, მაშინ როდესაც ფუნქციის ზომა მცირეა. ჩასადგმელი ფუნქციის გამოცხადებისათვის ფუნქციის სახელის წინ უნდა მივუთითოთ **inline** სპეციფიკატორი.

ჩასადგმელი ფუნქცია შეიძლება იყოს ან არ იყოს კლასის წევრი. მოცემული პროგრამით ხდება ჩასადგმელ ფუნქციასთან მუშაობის დემონსტრირება.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

//      inline ფუნქციასთან მუშაობა
inline int Jami(int par1, int par2)                //      Jami() ფუნქცია არის ჩასადგმელი
{
    return par1 + par2;
}

int _tmain(int argc, _TCHAR* argv[])
{
    int cvladi1, cvladi2, jami;

    cin >> cvladi1 >> cvladi2;
    jami = Jami(cvladi1, cvladi2);
    cout << "jami = " << jami << endl;

    system("pause");
    return 0;
}
```

ამ პროგრამაში Jami() ფუნქცია გამოცხადებულია როგორც ჩასადგმელი. ეს იმას ნიშნავს, რომ `int jami = obj.Jami(cvladi1, cvladi2);` სტრიქონში ფუნქციის გამოძახება არ მოხდება, რადგან მინიჭების ოპერატორის მარჯვენა ნაწილში მოთავსებული იქნება Jami() ფუნქციის კოდი.

ჩასადგმელი ფუნქცია გამოცხადებული უნდა იყოს მის პირველ გამოყენებამდე. წინააღმდეგ შემთხვევაში, კომპილატორს არ ეცოდინება, თუ რა კოდი უნდა ჩასვას პროგრამაში.

inline სპეციფიკატორი კომპილატორისთვის წარმოადგენს *მოთხოვნას* და არა ბრძანებას. თუ კომპილატორმა ვერ შეძლო მოთხოვნის შესრულება, მაშინ **inline** მოთხოვნა უარიყოფა და ფუნქცია კომპილირდება, როგორც ჩვეულებრივი ფუნქცია. ზოგიერთი კომპილატორი ფუნქციას არ აღიქვამს როგორც ჩასადგმელს, თუ ფუნქცია შეიცავს რეკურსიას, ან სტატიკურ წევრს და ა.შ.

თუ ფუნქცია და მისი კოდი გამოცხადებულია კლასის შიგნით, მაშინ ეს ფუნქცია ავტომატურად ხდება ჩასადგმელი ფუნქცია და **inline** სპეციფიკატორის მითითება აუცილებელი

არ არის. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

class Klasi
{
int ricxvi1, ricxvi2;
public :
Klasi(int par1, int par2)
{
    ricxvi1 = par1;
    ricxvi2 = par2;
}
int Jami()
{
    return ricxvi1 + ricxvi2; // Jami() ფუნქცია არის ჩასადგმელი
}
};

int _tmain(int argc, _TCHAR* argv[])
{
int cvladi1, cvladi2;

cin >> cvladi1 >> cvladi2;
Klasi obj(cvladi1, cvladi2);
int jami = obj.Jami();
cout << "jami = " << jami << endl;

system("pause");
return 0;
}
```

ბოლოს აღვნიშნოთ, რომ ჩასადგმელი ფუნქციები შეიძლება იყოს გადატვირთული.

მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

// ჩასადგმელი ფუნქციების გადატვირთვის დემონსტრირება
class Klasi
{
int ricxvi1, ricxvi2;
public :
int Jami(int par1, int par2)
{
    return par1 + par2;
}
double Jami(double par1, double par2)
```

```

{
    return par1 + par2;
}
};

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    Klasi obj;
    int cvladi1, cvladi2;
    double cvladi3, cvladi4;

    cin >> cvladi1 >> cvladi2 >> cvladi3 >> cvladi4;
    int jami_int = obj.Jami(cvladi1, cvladi2);
    double jami_double = obj.Jami(cvladi3, cvladi4);
    cout << jami_int << endl;
    cout << jami_double << endl;

    system("pause");
    return 0;
}

```

თუ კლასში გამოცხადებულია ფუნქციის პროტოტიპი, ხოლო კოდი კლასის გარეთ, მაშინ კოდის გამოცხადებისას ფუნქციის სახელის წინ უნდა მიეთითოს `inline` საკვანძო სიტყვა. ამის დემონსტრირება მოცემული პროგრამით ხდება.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// inline ფუნქციასთან მუშაობა
class Klasi
{
    int ricxvi1, ricxvi2;
public :
    Klasi(int par1, int par2);
    int Jami(); // ჩადგმა სრულდება ამ გამოცხადებაში
};
Klasi::Klasi(int par1, int par2)
{
    ricxvi1 = par1;
    ricxvi2 = par2;
}
inline int Klasi::Jami() // Jami() ფუნქცია არის ჩასადგმელი
{
    return ricxvi1 + ricxvi2;
}

int _tmain(int argc, _TCHAR* argv[])
{

```

```

int cvladi1, cvladi2;

cin >> cvladi1 >> cvladi2;
Klasi obj(cvladi1, cvladi2);
int jami = obj.Jami();
cout << "jami = " << jami << endl;

system("pause");
return 0;
}

```

ამ პროგრამაში Jami() ფუნქცია გამოცხადებულია როგორც ჩასადგმელი. ეს იმას ნიშნავს, რომ `int jami = obj.Jami();` სტრიქონში ფუნქციის გამოძახება არ მოხდება, რადგან აქ მოთავსებული იქნება Jami() ფუნქციის კოდი.

რეკურსია

რეკურსია არის პროცესი, როდესაც ფუნქცია თავის თავს იძახებს. ფუნქციას, რომელიც თავის თავს იძახებს - *რეკურსიული ფუნქცია* ეწოდება. რეკურსიული ფუნქციის საკვანძო კომპონენტია ოპერატორი, რომელიც ამავე ფუნქციას იძახებს.

რეკურსიის კლასიკური მაგალითია ფაქტორიალის გამოთვლა. N რიცხვის ფაქტორიალი აღინიშნება N!-ით და არის რიცხვების ნამრავლი 1-დან N-მდე. მაგალითად, 4-ის ფაქტორიალი არის $1 \times 2 \times 3 \times 4 = 24$. ქვემოთ მოცემული პროგრამით წარმოდგენილი რეკურსიული ფუნქცია განკუთვნილია რიცხვის ფაქტორიალის გამოთვლისათვის.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// რეკურსიული ფუნქცია
int factor(int ricxvi)
{
    int shedegi;
    if ( ricxvi == 1 ) return 1;
    shedegi = factor(ricxvi -1) * ricxvi;
    return shedegi;
}

int _tmain(int argc, _TCHAR* argv[])
{
    int number, factoriali;

    cin >> number;
    factoriali = factor(number);
    cout << factoriali << endl;

    system("pause");
    return 0;
}

```

}

რეკურსიული ფუნქციის შედგენისას სწორად უნდა განისაზღვროს რეკურსიის დამთავრების პირობა. ამ დროს უნდა შეწყდეს ფუნქციის გამოძახება და მან უნდა დააბრუნოს მნიშვნელობა. ჩვენს შემთხვევაში, რეკურსიული გამოძახება მთავრდება მაშინ, როდესაც არგუმენტი 1-ის ტოლი ხდება. ამ დროს ფუნქცია აბრუნებს 1-ს.

რეკურსიული ფუნქციების გამოყენებას აქვს როგორც დადებითი, ისე უარყოფითი მხარეები. უარყოფითი მხარეა ის, რომ რეკურსიული პროგრამები შედარებით ნელა სრულდება. გარდა ამისა, თუ რეკურსიული გამოძახებების რაოდენობა ძალიან დიდია, მაშინ შეიძლება აღიძრას განსაკუთრებული სიტუაცია (შეცდომა). ეს ძირითადად მაშინ ხდება, როდესაც არასწორად არის განსაზღვრული რეკურსიის დამთავრების პირობა: ფუნქციის გამოძახების ნაცვლად უნდა მოხდეს მნიშვნელობის დაბრუნება.

დადებითი მხარეა ის, რომ რიგი ალგორითმებისა რეკურსიულად შეიძლება რეალიზებული იყოს უფრო ეფექტურად და მარტივად. მაგალითად, სწრაფი დახარისხების, კატალოგებში ფაილების ძებნისა და ა.შ. რეკურსიული მიდგომა ხშირად გამოიყენება, აგრეთვე, ხელოვნური ინტელექტის სფეროში.

ფუნქციის შაბლონი

ფუნქციის შაბლონი (function template) შეიცავს კოდს, რომელიც გამოყენებული იქნება სხვადასხვა ტიპის მონაცემების მიმართ. ფუნქციის შაბლონს სხვანაირად **ფუნქცია-შაბლონს** უწოდებენ. ის ოპერირებს იმ ტიპის მონაცემებზე, რომელიც მას პარამეტრის სახით გადაეცემა. ფუნქციის შაბლონის გამოყენების აუცილებლობა გამოწვეულია იმით, რომ ბევრი ალგორითმი ლოგიკურად ერთნაირია და შეიძლება შესრულდეს სხვადასხვა ტიპის მონაცემებზე. მაგალითად, მასივში მაქსიმალური ელემენტის პოვნის ალგორითმი ერთნაირია როგორც მთელი რიცხვა მასივებისთვის, ისე წილადი რიცხვების მასივებისთვის. ფუნქციების შაბლონების გამოყენება საშუალებას გვაძლევს მონაცემების ტიპისგან დამოუკიდებლად განვსაზღვროთ ალგორითმი. ამის შემდეგ, კომპილატორი თვითონ აფორმირებს სწორ კოდს პარამეტრის ტიპის მიხედვით. ფაქტობრივად, ფუნქციის შაბლონი არის ფუნქცია, რომელიც ავტომატურად ასრულებს თვითგადატვირთვას.

ფუნქციის შაბლონის გამოცხადების სინტაქსია:

```
template < class ტიპი > დასაბრუნებელი_მნიშვნელობა ფუნქციის_სახელი ( პარამეტრების_სია )  
{  
  ფუნქციის კოდი  
}
```

template (შაბლონი) სიტყვა მიუთითებს, რომ უნდა შეიქმნას **ფუნქცია-შაბლონი**. **ტიპი** არის მონაცემების ტიპის ფიქტიური სახელი, რომელიც შეიცვლება მონაცემის რეალური ტიპის სახელით ფუნქციის კონკრეტული ვერსიის შექმნისას.

როდესაც კომპილატორი ქმნის ფუნქციის შაბლონის კონკრეტულ ვერსიას, მაშინ ის ქმნის **წარმოქმნილ ფუნქციას** (generated function). წარმოქმნილი ფუნქციის გენერირების პროცესს ფუნქციის **ეგზემპლარის შექმნა** (instantiating) ეწოდება. ანუ წარმოქმნილი ფუნქცია არის ფუნქცია-შაბლონის კონკრეტული ეგზემპლარი. მოცემული პროგრამით ხდება ფუნქციის შაბლონთან მუშაობის დემონსტრირება

```
#include "stdafx.h"  
#include "iostream"  
using namespace std;
```

```
// ფუნქციის შაბლონთან მუშაობის დემონსტრირება
// ფუნქცია-შაბლონის გამოცხადება
template < class Tipi > Tipi Funqcia(Tipi par1, Tipi par2)
{
    Tipi jami;

    jami = par1 + par2;
    return jami;
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    int mteli1, mteli2;
    double wiladi1, wiladi2;

    cin >> mteli1 >> mteli2 >> wiladi1 >> wiladi2;
    int shedegi1 = Funqcia(mteli1, mteli2);
    double shedegi2 = Funqcia(wiladi1, wiladi2);
    cout << shedegi1 << endl;
    cout << shedegi2 << endl;

    system("pause");
    return 0;
}
```

ამ პროგრამაში Tipi არის ზოგადი ტიპი, რომელიც გამოიყენება როგორც მონაცემების ტიპის ფიქტიური სახელი. როდესაც ხდება Funqcia(mteli1, mteli2) ფუნქციის გამოძახება, მაშინ Tipi იღებს int მნიშვნელობას, ხოლო Funqcia(wiladi1, wiladi2) ფუნქციის გამოძახებისას კი - double მნიშვნელობას.

class სიტყვის ნაცვლად შეგვიძლია გამოვიყენოთ typename სიტყვა:

```
#include "stdafx.h"
#include "iostream"
using namespace std;

template < typename Tipi > Tipi Funqcia_1(Tipi par1, Tipi par2)
{
    Tipi jami;

    jami = par1 + par2;
    return jami;
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    int mteli1, mteli2;
    double wiladi1, wiladi2;

    cin >> mteli1 >> mteli2 >> wiladi1 >> wiladi2;
```



```

int shedegi1 = Funqcia(mтели1, mтели2);
double shedegi2 = Funqcia(wiladi1, wiladi2);
cout << shedegi1 << endl;
cout << shedegi2 << endl;

system("pause");
return 0;
}

```

ფუნქცია-შაბლონის განსაზღვრისას შეგვიძლია მივუთითოთ ერთ ზოგად ტიპზე მეტი. მაგალითი:

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// ფუნქციის შაბლონში მითითებულია ორი ზოგადი ტიპი
template < class Tipi_1, class Tipi_2 > Tipi_1 Funqcia_2(Tipi_1 par1, Tipi_2 par2)
{
    Tipi_1 jami;

    jami = par1 + par2;
    return jami;
}

int _tmain(int argc, _TCHAR* argv[])
{
    int mтели1;
    double wiladi1;

    cin >> mтели1 >> wiladi1;
    double shedegi1 = Funqcia_2(wiladi1, mтели1);
    cout << shedegi1 << endl;

    system("pause");
    return 0;
}

```

Funqcia_2(wiladi1, mтели1) ფუნქციის გამოძახებისას Tipi_1 იღებს double მნიშვნელობას, Tipi_2 კი - int მნიშვნელობას.

განსხვავება ფუნქცია-შაბლონებსა და გადატვირთულ ფუნქციებს შორის იმაში მდგომარეობს, რომ ფუნქცია-შაბლონები ერთსა და იმავე მოქმედებებს ასრულებს სხვადასხვა ტიპის მონაცემებზე, ხოლო ფუნქციების გადატვირთვისას მათ შეიძლება განსხვავებული მოქმედებები შეასრულონ.

მართალია, ფუნქცია-შაბლონი თვითონ გადაიტვირთება, შესაძლებელია მისი აშკარად გადატვირთვაც. ამ შემთხვევაში, გადატვირთული ფუნქცია მალავს ფუნქცია-შაბლონს, თუ მოხდა პარამეტრების ტიპების დამთხვევა. მაგალითი:

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

template < class Tipi > Tipi Funqcia_3(Tipi par1, Tipi par2)
{
    Tipi jami;

    jami = par1 + par2;
    return jami;
}
// ფუნქცია-შაბლონის გადატვირთვა
double Funqcia_3(double par1, double par2)
{
    double namravli;

    namravli = par1 * par2;
    return namravli;
}

int _tmain(int argc, _TCHAR* argv[])
{
    int mteli1, mteli2;
    double wiladi1, wiladi2;

    cin >> mteli1 >> mteli2 >> wiladi1 >> wiladi2;
    int shedegi1 = Funqcia_3(mteli1, mteli2);
    double shedegi2 = Funqcia_3(wiladi1, wiladi2);
    cout << shedegi1<< endl;
    cout << shedegi2<< endl;

    system("pause");
    return 0;
}

```

Funqcia_3(wiladi1, wiladi2) ფუნქციის შესრულებისას გამოიძახება Funqcia_3(double par1, double par2) ფუნქცია. Funqcia_3(mteli1, mteli2) ფუნქციის შესრულებისას გამოიძახება Funqcia_3(Tipi par1, Tipi par2) ფუნქცია და Tipi ზოგადი ტიპი შეიცვლება int ტიპით.

კლასის შაბლონი

ფუნქციის შაბლონის გარდა, შესაძლებელია *კლასის შაბლონის (კლასი-შაბლონი)* განსაზღვრაც. კლასი-შაბლონში განსაზღვრულია ყველა ალგორითმი, ხოლო მონაცემების ფაქტიური ტიპები მოიცემა პარამეტრების სახით ამ კლასის ობიექტების შექმნისას.

კლასის შაბლონი სასარგებლოა მაშინ, როდესაც კლასი შეიცავს მუშაობის საერთო ლოგიკას. მისი საშუალებით შესაძლებელია ისეთი კლასის შექმნა, რომელიც ახდენს სტეკის, რიგის მუშაობის ორგანიზებას და ა.შ.

კლასი-შაბლონის სინტაქსია:

```

template < class ტიპი > class კლასის_სახელი {
    კლასის კოდი

```

```
}
```

კლასის შაბლონის შექმნის შემდეგ შეგვიძლია შევქმნათ ამ კლასის კონკრეტული ექსემპლარი შემდეგნაირად:

კლასის_სახელი < ტიპი > ობიექტი;

მოცემული პროგრამით ხდება კლასის შაბლონის შექმნის დემონსტრირება, რომელიც ახდენს სტეკის მუშაობის რეალიზებას მთელი რიცხვებისა და სიმბოლოებისთვის.

```
#include "stdafx.h"
#include "iostream"
using namespace std;
# define SIZE 10
// კლასის შაბლონის შექმნის დემონსტრირება
// კლასი-შაბლონი
template < class Steki_Tipi > class Steki
{
Steki_Tipi steki_masivi[SIZE]; // სტეკი
int ind; // სტეკის წვეროს ინდექსი
public :
void init()
{
ind = 0; // სტეკის ინიციალიზება
}
void push(Steki_Tipi ch);
Steki_Tipi pop();
};

template < class Steki_Tipi >
void Steki < Steki_Tipi >::push(Steki_Tipi ch)
{
if ( ind == SIZE )
{
cout << "steki savsea";
return;
}
steki_masivi[ind] = ch;
ind++;
}
//
template < class Steki_Tipi >
Steki_Tipi Steki < Steki_Tipi >::pop()
{
if ( ind == 0 )
{
cout << "steki carielia";
return 0;
}
ind--;
return steki_masivi[ind];
}
```

```

}

int _tmain(int argc, _TCHAR* argv[])
{
//      სიმბოლოების და წილადების სტეკის შექმნა
Steki <char> steki_1;
Steki <double> steki_2;
char c1 = 'a', c2 = 'b', c3 = 'a', c4 = 's';
double d1 = 10.5, d2 = 9.03, d3 = 5.17;
//      სიმბოლოების სტეკის შექმნა
steki_1.init();
steki_1.push(c1);
steki_1.push(c2);
steki_1.push(c3);
steki_1.push(c4);
cout << steki_1.pop() << endl;
cout << steki_1.pop() << endl;
cout << steki_1.pop() << endl;
cout << steki_1.pop() << endl;

//      წილადების სტეკის შექმნა
steki_2.init();
steki_2.push(d1);
steki_2.push(d2);
steki_2.push(d3);
cout << steki_2.pop() << endl;
cout << steki_2.pop() << endl;
cout << steki_2.pop() << endl;

system("pause");
return 0;
}

```

პროგრამაში steki_1 კლასის გამოცხადებისას Steki კლასი-შაბლონის კოდში ყველგან Steki_Tipi ზოგადი ტიპი შეიცვლება Char ტიპით. ანალოგიურად, steki_2 კლასის გამოცხადებისას Steki კლასი-შაბლონის კოდში ყველგან Steki_Tipi ზოგადი ტიპი შეიცვლება double ტიპით.

კლასის შაბლონის გამოცხადებისას შეიძლება მივუთითოთ რამდენიმე ზოგადი ტიპი. მაგალითი:

```

#include "stdafx.h"
#include "iostream"
using namespace std;

//      კლასების შაბლონში მითითებულია ორი ზოგადი ტიპი
template < class Tipi1, class Tipi2 > class ChemiKlasi
{
Tipi1 cvladi1;
Tipi2 cvladi2;

```

```

public :
ChemiKlasi(Tipi1 par1, Tipi2 par2)
{
    cvladi1 = par1;
    cvladi2 = par2;
}
void Naxva()
{
    cout << "pirveli cvladis mnishvneloba = " <<
    cvladi1 << "\n meore cvladis mnishvneloba = " << cvladi2 << endl;
}
};

int _tmain(int argc, _TCHAR* argv[])
{
    int mteli;
    double wiladi;

    cin >> mteli >> wiladi;
    ChemiKlasi <double, int> obj1(wiladi, mteli);
    obj1.Naxva();

    system("pause");
    return 0;
}

```

თავი 10. სტრუქტურები

სტრუქტურა

მონაცემთა ბაზებიდან და ფაილებიდან მიღებულ მონაცემებთან მუშაობისას ხშირად მოხერხებულია მონაცემების დიდი პორციების ერთიანად დამუშავება. ამისთვის მოხერხებულია სტრუქტურის გამოყენება. სტრუქტურა არის ტიპი, რომელსაც კლასის მსგავსად აქვს საკუთარი წევრები: ცვლადები და ფუნქციები. სტრუქტურასა და კლასს შორის ერთადერთი განსხვავება იმაში მდგომარეობს, რომ ავტომატურად (გაჩუმებით) კლასის წევრები დახურულია, სტრუქტურის წევრები კი - ღია. სტრუქტურის სინტაქსია:

```
struct ტიპის_სახელი
```

```
{
```

```
// ღია ცვლადები და ფუნქციები
```

```
private :
```

```
// დახურული ცვლადები და ფუნქციები
```

```
} ობიექტების_სია;
```

როგორც ვხედავთ, სტრუქტურის გამოცხადებისთვის გამოიყენება `struct` საკვანძო სიტყვა. მოცემული პროგრამით ხდება სტრუქტურასთან მუშაობის დემონსტრირება.

```
#include "stdafx.h"
```

```
#include "iostream"
```

```
using namespace std;
```

```
// სტრუქტურასთან მუშაობის დემონსტრირება
```

```
struct Samkutxedi
```

```
{
```

```
// ცვლადების გამოცხადება
```

```
int gverdi1;
```

```
int gverdi2;
```

```
int gverdi3;
```

```
// კონსტრუქტორის განსაზღვრა
```

```
Samkutxedi(int par1, int par2, int par3)
```

```
{
```

```
gverdi1 = par1;
```

```
gverdi2 = par2;
```

```
gverdi3 = par3;
```

```
perimetri = gverdi1 + gverdi2 + gverdi3;
```

```
}
```

```
// ფუნქციის განსაზღვრა
```

```
int Perimetri()
```

```
{
```

```
return perimetri;
```

```
}
```

```
private : int perimetri;
```

```
};
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```

int gverdi1, gverdi2, gverdi3;
cin >> gverdi1 >> gverdi2 >> gverdi3;
Samkutxedi struqtural(gverdi1, gverdi2, gverdi3);
int shedegi = struqtural.Perimetri();
cout << shedegi<< endl;

system("pause");
return 0;
}

```

შეიძლება ერთი სტრუქტურის მეორეში გადაწერა. მაგალითად,
Samkutxedi struqtural(gverdi1, gverdi2, gverdi3);
Samkutxedi struqtura2 = struqtural;

ჩამოთვლა

იმ შემთხვევებში, როდესაც გვჭირდება ისეთი ცვლადები, რომლებიც მნიშვნელობებს იღებენ დასაშვები მნიშვნელობების შეზღუდული ნაკრებიდან და რომლებზეც მიმართვა მოხერხებული იქნება რაიმე ჭდის გამოყენებით, გამოიყენება *ჩამოთვლები* (ენუმერაციები, enumerations). ჩამოთვლის გამოცხადების სინტაქსია:

enum name {enumerator-list} var;

ჩამოთვლის გამოცხადების მაგალითი:

```
enum KvirisDge { Orshabati, Samshabati, Otxshabati, Xutshabati, Paraskevi, Shabati, Kvira };
```

აქ განისაზღვრება ჩამოთვლის ტიპი - KvirisDge და ამ ტიპის ცვლადს შეიძლება მივანიჭოთ ჩამოთვლით განსაზღვრული სიდიდეებიდან ერთ-ერთი: Orshabati, Samshabati და ა.შ.

ჩამოთვლაში მივმართავთ მუდმივებს ჩამოთვლის ტიპის სახელის საშუალებით:

```
KvirisDge kd = KvirisDge::Samshabati;
```

ჩამოთვლაში განსაზღვრული მუდმივები წარმოადგენენ ობიექტებს. ავტომატურად ისინი არიან int ტიპის ობიექტები. მაგალითი:

```
#include "stdafx.h"
#include "iostream"
using namespace std;
```

// **ჩამოთვლასთან მუშაობის დემონსტრირება**

```
enum KvirisDge { Orshabati, Samshabati, Otxshabati, Xutshabati, Paraskevi, Shabati, Kvira };
```

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
    KvirisDge kd = KvirisDge::Samshabati;
    cout << kd << endl; // 1
    cout << KvirisDge::Kvira << endl; // 6

    system("pause");
    return 0;
}
```

ჩამოთვლა უნდა განისაზღვროს როგორც გლობალური, ფუნქციის გარეთ. ჩამოთვლაში

მუდმივებს შეიძლება ჰქონდეს 10.1 ცხრილში მოცემული ტიპები.

ცხრილი 10.1. ჩამოთვლის მუდმივების ტიპები

საბაზო ტიპი	ზომა ბაიტებში
bool	1
char	1
signed char	1
unsigned char	1
short	2
unsigned short	2
int	4
unsigned int	4
long	4
unsigned long	4
long long	8
unsigned long long	8

ჩამოთვლის მუდმივებისთვის შეგვიძლია ტიპის განსაზღვრა. მოცემულ მაგალითში მუდმივებს char ტიპი აქვს:

```
enum Tveebi : char { Ianvari, Tebervali, Marti, Aprili, Maisi, Ivnisi,
                    Ivlisi, Agvisto, Seqtemberi, Oqtomberi, Noemberi, Dekemberi };
```

პირველი მუდმივას სიდიდე ავტომატურად იქნება 0.

შეგვიძლია აშკარად მივუთითოთ სიდიდე რომელიმე მუდმივასთვის:

```
enum Tveebi : char { Ianvari = 2, Tebervali, Marti, Aprili, Maisi, Ivnisi,
                    Ivlisi, Agvisto, Seqtemberi, Oqtomberi, Noemberi, Dekemberi };
```

შედეგად, Tebervali მუდმივას სიდიდე იქნება 3 და ა.შ.

კიდევ ერთი მაგალითი:

```
enum Tveebi : char { Ianvari = 20, Tebervali = 10, Marti, Aprili, Maisi, Ivnisi,
                    Ivlisi, Agvisto, Seqtemberi, Oqtomberi, Noemberi, Dekemberi };
```

შედეგად, Marti მუდმივას სიდიდე იქნება 11 და ა.შ.

მუდმივებს შეიძლება ჰქონდეთ ერთნაირი მნიშვნელობები:

```
enum Tveebi : bool { Ianvari = true, Tebervali = true, Marti = true, Aprili = true, Maisi = true,
                    Ivnisi = true, Ivlisi = false, Agvisto = false, Seqtemberi = false,
                    Oqtomberi = false, Noemberi = false, Dekemberi = false };
```


დანართი 1. სავარჯიშოები თავების მიხედვით

თავი 2. C++ ენის საფუძვლები

არიტმეტიკული გამოსახულება

შეადგინეთ პროგრამა, რომელიც გამოთვლის ქვემოთ მოცემული არითმეტიკული გამოსახულებების მნიშვნელობებს:

2.1.1. $0,6n + 4,25$

2.1.2. $2b^2 + 3c - 1$

2.1.3. $-8m^3 + 4m^2 - 5m + 6$

2.1.4. $1,25ay^2 + 0,8(b - y)$

2.1.5. $9a^2 - 4ab + 5b^2$

2.1.6. $(5k + 2n)(k - 4,85n)$

2.1.7. $\frac{ab}{b+c} - \frac{a+d}{a+b}$

2.1.8. $\frac{(a+c)y}{x} + \frac{z}{b+x}$

2.1.9. $\frac{km}{5m-2k}$

2.1.10. $0,25x^2 + 4xy + 3,7y^2$

2.1.11. $\frac{a-0,184b}{ab}$

2.1.12. $y = 1 + x + \frac{x^2}{2}$

2.1.13. $y = \frac{x^2 + z^2}{1 - \frac{x^2 - z^2}{2}}$

შეადგინეთ პროგრამა, რომელიც გამოთვლის:

2.2.1. დენის ძალის მნიშვნელობას, $J = \frac{U}{R}$.

2.2.2. ხუთი რიცხვის საშუალო არითმეტიკულს.

2.2.3. ოთხი რიცხვის საშუალო გეომეტრიულს.

2.2.4. სამკუთხედის ფართობს, $S = \frac{ah}{2}$.

2.2.5. სამკუთხედის პერიმეტრს, $p = a + b + c$.

2.2.6. კვადრატის ფართობს.

2.2.7. მართკუთხედის პერიმეტრს.

2.2.8. წრეწირის სიგრძეს, $C = 2\pi r$.

2.2.9. წრეწირის ფართობს, $S = \pi r^2$.

2.2.10. ტრაპეციის ფართობს, $S = \frac{a+b}{2} h$.

2.2.11. ტრაპეციის პერიმეტრს.

ინკრემენტისა და დეკრემენტის ოპერატორები

შეადგინეთ პროგრამა, რომელიც გამოთვლის შემდეგი ფორმულების მნიშვნელობებს:

- 2.3.1. $y = a + --b + c++;$ $a = 5, b = 7, c = 12.$
 2.3.2. $y = ++a - b-- - ++c;$ $a = 5, b = 7, c = 12.$
 2.3.3. $y = ++a - b-- - c;$ $a = 5, b = 7, c = 12.$
 2.3.4. $y = --a - b + ++c;$ $a = 5, b = 7, c = 12.$
 2.3.5. $y = --a - ++b - ++c;$ $a = 5, b = 7, c = 12.$
 2.3.6. $y = a-- - ++b - --c;$ $a = 5, b = 7, c = 12.$
 2.3.7. $y = a-- - b++ - c--;$ $a = 5, b = 7, c = 12.$
 2.3.8. $y = --a * b++ + --c;$ $a = 5, b = 7, c = 12.$
 2.3.9. $y = a-- * ++b - c--;$ $a = 5, b = 7, c = 12.$
 2.3.10. $y = --a * b++ / --c;$ $a = 5, b = 7, c = 12.$
 2.3.11. $y = --a / b++ * --c;$ $a = 5, b = 7, c = 12.$
 2.3.12. $y = ++a * b-- - c;$ $a = 5, b = 7, c = 12.$
 2.3.13. $y = ++a / b-- - c;$ $a = 5, b = 7, c = 12.$
 2.3.14. $y = ++a + b-- * c;$ $a = 5, b = 7, c = 12.$
 2.3.15. $y = ++a - b-- / c;$ $a = 5, b = 7, c = 12.$

2.3.16. $y = \frac{a-- - b++ - c--}{--a - ++b - ++c}$ $a = 5, b = 7, c = 12.$

2.3.17. $y = \frac{a + --b + c++}{a + b++ + c--}$ $a = 5, b = 7, c = 12.$

2.3.18. $y = \frac{a++ - ++b * --c}{a-- - ++b * ++c}$ $a = 5, b = 7, c = 12.$

2.3.19. $y = \frac{--a + b-- * c++}{++a * b++ - --c}$ $a = 5, b = 7, c = 12.$

2.3.20. $y = \frac{++a + b++ * c++}{a * b++ - c++}$ $a = 5, b = 7, c = 12.$

2.3.21. $y = \frac{++a + b--}{c++};$ $a = 5, b = 7, c = 12.$

2.3.22. $y = \frac{a-- - b++}{--c};$ $a = 5, b = 7, c = 12.$

2.3.23. $y = \frac{++a}{b-- - c++};$ $a = 5, b = 7, c = 12.$

2.3.24. $y = \frac{a--}{b++ + --c};$ $a = 5, b = 7, c = 12.$

შედარებისა და ლოგიკის ოპერატორები

შეადგინეთ პროგრამა, რომელიც გამოთვლის ქვემოთ მოცემული ლოგიკური გამოსახულებების მნიშვნელობებს:

2.4.1. $Y = X1 \&\& X2 \parallel X3 \&\& !X4,$ სადაც $x1= true, x2=false, x3=true, x4=false$

- 2.4.2. $Y = X1 \parallel !X2 \&\& X3 \parallel X4$, სადაც $x1= \text{false}$, $x2=\text{false}$, $x3=\text{true}$, $x4=\text{true}$
 2.4.3. $Y = !X1 \&\& !X2 \&\& X3 \parallel X4$, სადაც $x1= \text{true}$, $x2=\text{true}$, $x3=\text{false}$, $x4=\text{false}$
 2.4.4. $Y = !X1 \parallel X2 \parallel !X3 \&\& X4$, სადაც $x1= \text{false}$, $x2=\text{true}$, $x3=\text{true}$, $x4=\text{false}$
 2.4.5. $Y = X1 \parallel X2 \parallel !X3 \parallel !X4$, სადაც $x1 = \text{false}$, $x2 = \text{true}$, $x3 = \text{true}$, $x4 = \text{false}$
 2.4.6. $Y = X1 \&\& X2 \parallel !X3 \&\& !X4$, სადაც $x1 = \text{false}$, $x2 = \text{true}$, $x3 = \text{true}$, $x4 = \text{false}$
 2.4.7. $Y = X1 \&\& !X2 \&\& !X3 \parallel !X4$, სადაც $x1 = \text{false}$, $x2 = \text{true}$, $x3 = \text{true}$, $x4 = \text{false}$
 2.4.8. $Y = !X1 \parallel !X2 \&\& !X3 \parallel !X4$, სადაც $x1 = \text{false}$, $x2 = \text{true}$, $x3 = \text{true}$, $x4 = \text{false}$
 შეადგინეთ პროგრამა, რომელიც ეკრანზე გამოიტანს შედარების შედეგს x რიცხვის სხვადასხვა მნიშვნელობისთვის:

- 2.5.1. $y = x \leq 10$.
 2.5.2. $y = x < 19$.
 2.5.3. $y = x = 30$.
 2.5.4. $y = x \neq 20$.
 2.5.5. $y = x > 15$.
 2.5.6. $y = x \geq 8$.
 2.5.7. $y = 10 < x \leq 20$.
 2.5.8. $y = 25 \leq x < 55$.
 2.5.9. $y = 30 > x \geq 5$.
 2.5.10. $y = 35 \geq x > 15$.
 2.5.11. $y = x < 5$ ან $x > 20$.
 2.5.12. $y = x = 15$ ან $x > 20$.

მათემატიკის ფუნქციები

შეადგინეთ პროგრამა, რომელიც გამოთვლის შემდეგი ფორმულების მნიშვნელობებს:

- 2.6.1. $y = \sqrt{1 + \sqrt{|x|}}$
 2.6.2. $y = (1 + x) \frac{x - \frac{\sqrt{x-1}}{4}}{e^x + \frac{1}{x^2 + 4}}$
 2.6.3. $y = \sin x^3 + \cos^3 x^4 - e^{\sqrt{x}}$
 2.6.4. $\frac{\sin x + \cos x}{\ln x}$
 2.6.5. $\frac{a+b-5,6}{\sqrt{a+b}}$
 2.6.6. $\frac{\sqrt{a^3+b}}{a+b^3}$

$$2.6.7. \quad y = \frac{\sqrt{|x-1|} - \sqrt{|x|}}{1 + \frac{x^2}{2} + \frac{x^3}{4}}$$

$$2.6.8. \quad y = 1 + |x^3 - x| + \frac{(x^2 - x)^2}{2} - \frac{(x+1)^3}{3}$$

$$2.6.9. \quad y = \frac{1 + \cos(x-2)^2}{\frac{x^4}{2} - \sin^2 x}$$

$$2.6.10. \quad y = \frac{2.2 \cos(x - e^x)}{\sqrt{\sin^2 x^3 - 2.2}}$$

$$2.6.11. \quad y = x^3 - \frac{e^{-x^2} + \cos x}{3.7 + \frac{x^2}{5.8 - \sqrt{x+1}}}$$

$$2.6.12. \quad y = \ln \left| x - \sqrt{|x^3 + 5.6|} \right| \left(x^2 - \frac{x-1}{e^{-x}} \right)$$

$$2.6.13. \quad y = x^3 - \frac{|e^{-x} - \cos x|}{\sqrt{x+1}}$$

$$2.6.14. \quad y = x^3 - \frac{\sqrt{4x^3 + 3x^2 + 2}}{3.7 + \ln x}$$

$$2.6.15. \quad y = \sqrt{e^x - 1} + \operatorname{tg} x + \frac{\ln x^3}{\sqrt{x^3}}$$

2.6.16.
$$y = \ln(x^2 - x + 1) \frac{\sin x^4}{\cos^4 x}$$

2.6.17.
$$y = \sqrt{\ln(x^3 + x^2)} - \operatorname{tg} x - e^x - 1$$

თავი 3. მმართველი ოპერატორი

if ოპერატორი

შეადგინეთ პროგრამა, რომელიც დაადგენს:

- 3.1.1. რიცხვი დადებითია თუ უარყოფითი. რიცხვი \sin ფუნქციის გამოყენებით შეიტანეთ. ეკრანზე გამოიტანეთ შესაბამისი შეტყობინება.
- 3.1.2. რიცხვი კენტია თუ ლუწი.
- 3.1.3. რიცხვი არის თუ არა 5-ის ჯერადი.
- 3.1.4. რიცხვი არის თუ არა 30-ის ტოლი.
- 3.1.5. რიცხვი მეტია თუ არა 10-ზე.
- 3.1.6. რიცხვი ნაკლებია თუ არა -2,01-ზე.
- 3.1.7. რიცხვი არის თუ არა მეტი ან ტოლი 15-ზე.
- 3.1.8. რიცხვი არის თუ არა ნაკლები ან ტოლი 23-ზე.
- 3.1.9. ორ რიცხვს შორის მაქსიმალურს.
- 3.1.10. ორ რიცხვს შორის მინიმალურს.
- 3.1.11. სამ რიცხვს შორის რამდენია კენტი.
- 3.1.12. სამ რიცხვს შორის რამდენია 5-ის ტოლი.
- 3.1.13. სამ რიცხვს შორის რამდენია 10-ზე მეტი.

ჩადგმული if ოპერატორი

შეადგინეთ პროგრამა, რომელიც დაადგენს:

- 3.2.1. სამ რიცხვს შორის მაქსიმალურს.
- 3.2.2. სამ რიცხვს შორის მინიმალურს.
- 3.2.3. ორ რიცხვს შორის არის თუ არა დადებითი.
- 3.2.4. ორ რიცხვს შორის არის თუ არა უარყოფითი.
- 3.2.5. ორ რიცხვს შორის არის თუ არა 7-ის ჯერადი.
- 3.2.6. სამ რიცხვს შორის არის თუ არა ლუწი.
- 3.2.7. სამ რიცხვს შორის არის თუ არა დადებითი.
- 3.2.8. სამ რიცხვს შორის არის თუ არა უარყოფითი.
- 3.2.9. ორ რიცხვს შორის რომელია 5-ის ჯერადი.
- 3.2.10. სამ რიცხვს შორის რომელია 50-ზე ნაკლები.

ლოგიკა

შეადგინეთ პროგრამა, რომელიც დაადგენს:

- 3.3.1. x რიცხვი არის თუ არა $0 < x \leq 17$ დიაპაზონში მოთავსებული.
- 3.3.2. x რიცხვი არის თუ არა $10 \leq x < 19$ დიაპაზონში მოთავსებული.
- 3.3.3. x რიცხვი არის თუ არა 25-ზე ნაკლები ან 100-ზე მეტი.
- 3.3.4. x რიცხვი არის თუ არა 30-ის ტოლი ან 5-ზე ნაკლები.
- 3.3.5. ორივე რიცხვი არის თუ არა დადებითი.

3.3.6. სამივე რიცხვი არის თუ არა 7-ის ჯერადი.

switch ოპერატორი

შეადგინეთ პროგრამა, რომელიც:

3.4.1. თვის ნომრის მიხედვით გასცემს შესაბამისი თვის დასახელებას.

3.4.2. ახდენს შუქნიშნის მუშაობის იმიტირებას. შეტანილი სიმბოლო მიხედვით გასცემს შესაბამის ფერს: Y – Yellow, G – Green, R – Red.

3.4.3. გაარკვევს კლავიატურიდან შეტანილი პატარა ასო ხმოვანია თუ თანხმოვანი.

3.4.4. სტუდენტის შეფასების მიხედვით (A, B, C, D, E) განსაზღვრავს ქულების დიაპაზონს.

3.4.5. სიმბოლო-ციფრის მიხედვით გაარკვევს ის კენტია თუ ლუწი.

3.4.6. შეტანილი სიმბოლოების მიხედვით მეტრი გადაყავს კილომეტრში და პირიქით: m - მეტრი კილომეტრში და k - კილომეტრი მეტრში.

for, while, do-while ოპერატორები

შეადგინეთ პროგრამა, რომელიც გამოთვლის შემდეგი გამოსახულებების მნიშვნელობებს:

3.5.1.
$$\ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5}$$

3.5.2.
$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9}$$

3.5.3.
$$\frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2}$$

3.5.4.
$$\frac{\pi^2}{8} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \frac{1}{9^2}$$

3.5.5.
$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5}$$

3.5.6.
$$\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \frac{x^4}{4} - \frac{x^5}{5}$$

3.5.7.
$$\arctg x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9}$$

3.5.8.
$$\frac{1}{1+x} = 1 - x + x^2 - x^3 + x^4 - x^5$$

3.5.9.
$$\frac{1}{1-x} = 1 + x + x^2 + x^3 + x^4 + x^5$$

3.5.10.
$$\sqrt{1+x} = 1 + \frac{1}{2}x - \frac{1}{8}x^2 + \frac{1}{16}x^3$$

3.5.11.
$$\frac{1}{\sqrt{1+x}} = 1 - \frac{1}{2}x + \frac{3}{8}x^2 - \frac{5}{16}x^3$$

$$3.5.12. \quad \arcsin x = x + \frac{1}{2 \cdot 3} x^3 + \frac{1 \cdot 3}{3 \cdot 4 \cdot 5} x^5$$

$$3.5.13. \quad \ln \frac{1+x}{1-x} = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7}\right)$$

$$3.5.14. \quad y = \frac{1 \cdot 2}{3 \cdot 4} + \frac{3 \cdot 4}{5 \cdot 6} + \frac{5 \cdot 6}{7 \cdot 8}$$

შეადგინეთ პროგრამა, რომელიც:

3.5.15. დათვლის ლუწი რიცხვების რაოდენობას 5-დან 27-მდე დიაპაზონში.

3.5.16. შეკრებს კენტ რიცხვებს 10-დან 20-მდე დიაპაზონში.

3.5.17. დათვლის 8-ის ჯერადი რიცხვების რაოდენობას 20-დან 50-მდე დიაპაზონში.

3.5.18. იპოვის 10-დან 70-მდე იმ რიცხვების ჯამს, რომლებიც არ არიან 8-ის ჯერადი.

3.5.19. შეადგინეთ პროგრამა, რომელიც შეკრებს 3-ის ჯერად რიცხვებს 14-დან 29-მდე დიაპაზონში.

continue, break ოპერატორები

შეადგინეთ პროგრამა, რომელიც:

3.6.1. შეკრებს 1-დან 20-მდე რიცხვებს მანამ, სანამ ჯამი არ გახდება 24-ზე მეტი.

3.6.2. დათვლის 4-ის ჯერადი რიცხვებს 6-დან 79-მდე დიაპაზონში მანამ, სანამ 4-ის ჯერადი რიცხვი არ გახდება 50-ზე მეტი.

3.6.3. შეკრებს 9-ის ჯერად რიცხვებს 2-დან 47-მდე დიაპაზონში მანამ, სანამ 9-ის ჯერადი რიცხვი იქნება 30-ზე ნაკლები.

3.6.4. გაამრავლებს 1-დან 20-მდე რიცხვებს მანამ, სანამ ნამრავლი არ გახდება 10 000-ზე მეტი.

თავი 4. მასივი, სტრიქონი და ბიტობრივი ოპერაცია

ერთგანზომილებიანი მასივი

შეადგინეთ პროგრამა, რომელიც:

4.1.1. იპოვის M_{10} მასივის ელემენტების ჯამს.

4.1.2. იპოვის M_{10} მასივის ელემენტების ნამრავლს.

4.1.3. იპოვის M_{10} მასივის ელემენტების კვადრატების ჯამს.

4.1.4. იპოვის M_{10} მასივის მინიმალურ ელემენტს.

4.1.5. იპოვის M_{10} მასივის მაქსიმალური ელემენტის ინდექსს.

4.1.6. იპოვის M_{10} მასივის ლუწი ელემენტების ჯამს.

4.1.7. იპოვის M_{10} მასივის კენტი ელემენტების რაოდენობას.

4.1.8. იპოვის M_{10} მასივის დადებითი ელემენტების ჯამს.

4.1.9. იპოვის M_{10} მასივის უარყოფითი ელემენტების რაოდენობას.

4.1.10. იპოვის M_{10} მასივის არანულოვანი ელემენტების რაოდენობას.

4.1.11. იპოვის M_{10} მასივის ნულოვანი ელემენტების რაოდენობას.

4.1.12. იპოვის M_{10} მასივის ლუწი ინდექსის მქონე ელემენტების ჯამს.

4.1.13. იპოვის M_{10} მასივის კენტი ინდექსის მქონე ელემენტების რაოდენობას.

4.1.14. იპოვის მაქსიმალურ სხვაობას M_{10} და N_{10} მასივების ელემენტებს შორის.

4.1.15. იპოვის M_{10} მასივის პირველ უარყოფით ელემენტს და მის ინდექსს.

4.1.16. იპოვის M_{10} მასივის პირველი დადებითი ელემენტის ინდექსს.

4.1.17. იპოვის M_{10} მასივის უარყოფითი ელემენტებიდან უდიდესის ინდექსს.

- 4.1.18. გამოთვლის M_{10} მასივის ელემენტების საშუალო არითმეტიკულს.
- 4.1.19. გამოთვლის M_{10} მასივის ელემენტების საშუალო გეომეტრიულს.
- 4.1.20. დათვლის M_{10} მასივის იმ ელემენტების რაოდენობას, რომლებიც მეტია x რიცხვზე.
- 4.1.21. იპოვის M_{10} მასივის იმ ელემენტების ჯამს, რომლებიც ნაკლებია x რიცხვზე.
- 4.1.22. განსაზღვრავს თუ რამდენი ელემენტია მოთავსებული M_{10} მასივის ორ მოცემულ ელემენტს შორის.
- 4.1.23. იპოვის M_{10} მასივის დადებითი ელემენტებიდან უმცირესს.
- 4.1.24. იპოვის M_{10} მასივის უარყოფითი ელემენტებიდან უდიდესს.
- 4.1.25. იპოვის იმ ელემენტების ჯამს, რომლებიც მოთავსებულია M_{10} მასივის ორ მოცემულ ელემენტს შორის.
- 4.1.26. იპოვის იმ ელემენტებს შორის მაქსიმალურს, რომლებიც მოთავსებულია M_{10} მასივის ორ მოცემულ ელემენტს შორის.
- 4.1.27. დათვლის M_{10} მასივის იმ ელემენტების რაოდენობას, რომელთა მნიშვნელობები 20-ზე მეტია.
- 4.1.28. M_{10} მასივიდან N_{10} მასივში გადაწერს იმ ელემენტებს, რომელთა მნიშვნელობები 20-ზე ნაკლებია.
- 4.1.29. M_{10} მასივში იპოვის იმ ელემენტების რაოდენობას, რომელთა მნიშვნელობები აღემატება საკუთარ ინდექსს.
- 4.1.30. M_{10} მასივის დადებით ელემენტებს გადაწერს N_{10} მასივში.
- 4.1.31. M_{10} მასივის 5-ის ჯერად ელემენტებს გადაწერს N_{10} მასივში.
- 4.1.32. M_{10} მასივის არანულოვან ელემენტებს გადაწერს N_{10} მასივში.
- 4.1.33. M_{10} მასივის კენტი ინდექსის მქონე ელემენტებს გადაწერს N_5 მასივში.
- 4.1.34. M_{10} მასივის ლუწი ინდექსის მქონე ელემენტებს გადაწერს N_5 მასივში.
- 4.1.35. M_{20} მასივის ლუწინდექსიან ელემენტებს გადაწერს A_{10} მასივში, ხოლო კენტინდექსიან ელემენტებს კი - B_{10} მასივში.
- 4.1.36. M_{10} მასივის დადებითი ელემენტების ინდექსებს გადაწერს N_{10} მასივში.
- 4.1.37. M_{10} მასივის ლუწ ელემენტებს გადაწერს N_{10} მასივში.
- 4.1.38. M_{10} მასივის კენტი ინდექსის მქონე ელემენტებს გადაწერს N_5 მასივში.
- 4.1.39. M_{10} მასივის 3-ის ჯერადი ელემენტების ინდექსებს გადაწერს N_{10} მასივში.
- 4.1.40. მოახდენს ზრდადობის მიხედვით დალაგებული A_{10} და B_{10} მასივების შერწყმას და შექმნის ახალ ზრდადობის მიხედვით დალაგებულ M_{20} მასივს.
- 4.1.41. M_{10} მასივში განსაზღვრავს გაორმაგებული კენტი რიცხვების რაოდენობას და ჩაწერს მათ N მასივში.
- 4.1.42. N მასივში გადაწერს M_{10} მასივის იმ ელემენტებს, რომლებიც 5-ზე გაყოფისას იძლევიან 1-დან 4-ის ტოლ ნაშთს.
- 4.1.43. M_{10} მასივის უარყოფით ელემენტებს შეცვლის 0-ებით.
- 4.1.44. M_{10} მასივის კენტ ელემენტებს შეცვლის მათი კვადრატებით.
- 4.1.45. M_{10} მასივის ელემენტებს უკუ (შებრუნებული) თანმიმდევრობით დაალაგებს.
- 4.1.46. M_{10} მასივის ელემენტებს დაალაგებს ისე, რომ დასაწყისში განლაგდეს ყველა უარყოფითი რიცხვი რიგითობის შენარჩუნებით, შემდეგ კი დადებითი რიცხვები რიგითობის შენარჩუნებით. დამხმარე მასივი არ გამოიყენოთ.
- 4.1.47. შექმნის A_{10} მასივს, რომელშიც ჯერ მოთავსებული იქნება M_{10} მასივის არანულოვანი ელემენტები, შემდეგ კი - ნულოვანი.
- 4.1.48. M_{10} მასივში ელემენტებს დაალაგებს ისე, რომ ჯერ მოთავსდეს არანულოვანი ელემენტები, შემდეგ კი - ნულოვანი.
- 4.1.49. M_{10} მასივის ელემენტებს ციკლურად დაძრავს მარჯვნივ n ელემენტით.
- 4.1.50. M_{10} მასივის ელემენტებს ციკლურად დაძრავს მარცხნივ n ელემენტით.

- 4.1.51. M_{10} მასივში ადგილებს შეუცვლის კენტი და ლუწი ინდექსის მქონე ელემენტებს.
 - 4.1.52. შეამოწმებს ემთხვევა თუ არა M_{10} მასივის პირველი ნახევარი მის მეორე ნახევარს.
 - 4.1.53. შეამოწმებს განსხვავდებიან თუ არა M_{10} და N_{10} მასივები.
 - 4.1.54. განსაზღვრავს შეიცავს თუ არა M_{10} მასივი N_{10} მასივს.
 - 4.1.55. განსაზღვრავს M_{10} მასივის ერთმანეთის მიყოლებით მდებარე ნულის ტოლი ელემენტების ყველაზე გრძელ მიმდევრობას.
 - 4.1.56. დათვლის M_{10} მასივის განსხვავებული ელემენტების რაოდენობას.
 - 4.1.57. დაადგენს არის თუ არა M_{10} მასივში ერთნაირი ელემენტები.
 - 4.1.58. M_{10} მასივში იპოვის ზრდადობით დალაგებულ ყველაზე გრძელ მიმდევრობას.
 - 4.1.59. M_{10} მასივში იპოვის კლებადობით დალაგებულ ყველაზე მოკლე მიმდევრობას.
 - 4.1.60. დათვლის M_{10} მასივის მეზობელ ელემენტებს შორის ნიშანცვლათა რაოდენობას.
 - 4.1.61. იპოვის M_{10} მასივის იმ ელემენტების ინდექსებს, რომელთა შორის სხვაობა უდიდესია.
 - 4.1.62. იპოვის M_{10} მასივის იმ ელემენტების ინდექსებს, რომელთა შორის სხვაობა უმცირესია.
 - 4.1.63. M_{10} მასივში იპოვის იმ 5-ელემენტიან მიმდევრობას, რომლის ელემენტების ჯამი მინიმალურია.
- ბ. კენტი ინდექსის მქონე რიცხვებს შორის უდიდესს;
- გ. ლუწი ინდექსის მქონე რიცხვებს შორის უმცირესს.

ორგანზომილებიანი მასივი

შეადგინეთ პროგრამა, რომელიც:

- 4.2.1. იპოვის $A_{5,5}$ მასივის მაქსიმალურ ელემენტს.
- 4.2.2. იპოვის $A_{5,5}$ მასივის მინიმალური ელემენტის ინდექსებს.
- 4.2.3. იპოვის $A_{5,5}$ მასივის ელემენტების ჯამს.
- 4.2.4. იპოვის $A_{5,5}$ მასივის ელემენტების ნამრავლს.
- 4.2.5. იპოვის $A_{5,5}$ მასივის დადებითი ელემენტების ჯამს.
- 4.2.6. იპოვის $A_{5,5}$ მასივის უარყოფითი ელემენტების რაოდენობას.
- 4.2.7. იპოვის $A_{5,5}$ მასივის კენტი ელემენტების ჯამს.
- 4.2.8. იპოვის $A_{5,5}$ მასივის ლუწი ელემენტების რაოდენობას.
- 4.2.9. იპოვის $A_{5,5}$ მასივის არანულოვანი ელემენტების ნამრავლს.
- 4.2.10. იპოვის $A_{5,5}$ მასივის ნულოვანი ელემენტების რაოდენობას.
- 4.2.11. იპოვის $A_{5,5}$ მასივის მთავარი დიაგონალის ელემენტების ჯამს.
- 4.2.12. იპოვის $A_{5,5}$ მასივის არამთავარი დიაგონალის ელემენტების ნამრავლს.
- 4.2.13. იპოვის $A_{5,5}$ მასივის მთავარი დიაგონალის მინიმალურ ელემენტს.
- 4.2.14. იპოვის $A_{5,5}$ მასივის არამთავარი დიაგონალის მაქსიმალური ელემენტის ინდექსებს.
- 4.2.15. იპოვის $A_{5,5}$ მასივის მთავარი დიაგონალის 6-ის ჯერადი ელემენტების რაოდენობას.
- 4.2.16. იპოვის $A_{5,5}$ მასივის არამთავარი დიაგონალის დადებითი ელემენტების ჯამს.
- 4.2.17. იპოვის $A_{5,5}$ მასივის 4-ის ჯერადი ელემენტების რაოდენობას.
- 4.2.18. დათვლის $A_{5,5}$ მასივის იმ ელემენტების რაოდენობას, რომელთა მნიშვნელობები 20-ზე მეტია.
- 4.2.19. $A_{5,5}$ მასივიდან M_{25} მასივში გადაწერს იმ ელემენტებს, რომელთა მნიშვნელობები 20-ზე ნაკლებია.
- 4.2.20. $A_{5,5}$ მასივის ლუწ ელემენტებს გადაწერს B_{25} მასივში.
- 4.2.21. $A_{5,5}$ მასივის არანულოვან ელემენტებს გადაწერს B_{25} მასივში.
- 4.2.22. $A_{5,5}$ მასივის თითოეული სვეტის ელემენტების ჯამს ჩაწერს B_5 მასივში.
- 4.2.23. $A_{5,5}$ მასივის თითოეული სტრიქონის ელემენტების ნამრავლს ჩაწერს B_5 მასივში.
- 4.2.24. დათვლის $A_{5,5}$ მასივის თითოეული სვეტის დადებითი ელემენტების რაოდენობას და ჩაწერს მათ B_5 მასივში.

- 4.2.25.** იპოვის $A_{5,5}$ მასივის თითოეული სტრიქონის კენტი ელემენტების ჯამს და ჩაწერს მათ B_5 მასივში.
- 4.2.26.** იპოვის $A_{5,5}$ მასივის თითოეული სვეტის მაქსიმალურ ელემენტს და ჩაწერს მათ B_5 მასივში.
- 4.2.27.** იპოვის $A_{5,5}$ მასივის თითოეული სტრიქონის მინიმალურ ელემენტს და ჩაწერს მათ B_5 მასივში.
- 4.2.28.** იპოვის $A_{5,5}$ მასივის მთავარი დიაგონალის მაქსიმალურ ელემენტს და ამ ელემენტის შემცველ სტრიქონს გადაწერს B_5 მასივში.
- 4.2.29.** იპოვის $A_{5,5}$ მასივის არამთავარი დიაგონალის მინიმალურ ელემენტს და ამ ელემენტის შემცველ სვეტს გადაწერს B_5 მასივში.
- 4.2.30.** იპოვის $A_{5,5}$ მასივის იმ სტრიქონს, რომელიც შეიცავს ლუწი ელემენტების მაქსიმალურ რაოდენობას და ამ სტრიქონს გადაწერს $B_{5,5}$ მასივში.
- 4.2.31.** იპოვის $A_{5,5}$ მასივის იმ სვეტს, რომელიც შეიცავს კენტი ელემენტების მინიმალურ რაოდენობას და ამ სვეტს გადაწერს B_5 მასივში.
- 4.2.32.** იპოვის $A_{5,5}$ მასივის მაქსიმალური ელემენტის შემცველ სტრიქონს და გადაწერს მას B_5 მასივში.
- 4.2.33.** იპოვის $A_{5,5}$ მასივის მინიმალური ელემენტის შემცველ სვეტს და გადაწერს მას B_5 მასივში.
- 4.2.34.** $A_{5,5}$ მასივის მთავარი დიაგონალის ელემენტებს B_5 მასივში გადაწერს.
- 4.2.35.** $A_{5,5}$ მასივის არამთავარი დიაგონალის ელემენტებს B_5 მასივში გადაწერს.
- 4.2.36.** იანგარიშებს $A_{5,5}$ მასივის ლუწი ნომრების მქონე სვეტების ელემენტების საშუალო არითმეტიკულს და ჩაწერს მათ B_5 მასივში.
- 4.2.37.** იანგარიშებს $A_{5,5}$ მასივის კენტი ნომრების მქონე სტრიქონების ელემენტების საშუალო გეომეტრიულს და ჩაწერს მათ B_2 მასივში.
- 4.2.38.** იპოვის $A_{5,5}$ მასივის იმ სტრიქონების ინდექსებს, რომელთა ყველა ელემენტი ნულის ტოლია. ინდექსები ჩაწერეთ B_5 მასივში.
- 4.2.39.** იპოვის $A_{5,5}$ მასივის იმ სვეტების ინდექსებს, რომელთა ყველა ელემენტი კენტია. ინდექსები ჩაწერეთ B_5 მასივში.
- 4.2.40.** იპოვის $A_{5,5}$ მასივის იმ სტრიქონების ინდექსებს, რომელთა ელემენტები ქმნიან მონოტონურად ზრდად მიმდევრობას. ინდექსები ჩაწერეთ B_5 მასივში.
- 4.2.41.** იპოვის $A_{5,5}$ მასივის იმ სვეტების ინდექსებს, რომელთა ელემენტები ქმნიან მონოტონურად კლებად მიმდევრობას. ინდექსები ჩაწერეთ B_5 მასივში.
- 4.2.42.** $A_{5,5}$ მასივიდან წაშლის იმ სვეტსა და სტრიქონს, რომელთა გადაკვეთაზე მდებარეობს მატრიცის მინიმალური ელემენტი.
- 4.2.43.** $A_{5,5}$ მასივიდან წაშლის ნულების შემცველ სტრიქონს.
- 4.2.44.** $A_{5,5}$ მასივიდან წაშლის ნულების შემცველ სვეტს.
- 4.2.45.** $A_{5,5}$ მასივის ნულოვან ელემენტებს შეცვლის 1-ებით.
- 4.2.46.** $A_{5,5}$ მასივში ადგილებს შეუცვლის მითითებული ნომრის მქონე სვეტსა და უკანასკნელ სვეტს.
- 4.2.47.** $A_{5,5}$ მასივში ადგილებს შეუცვლის მითითებული ნომრის მქონე სტრიქონსა და პირველ სტრიქონს.
- 4.2.48.** შეასრულებს $A_{5,5}$ მასივის სტრიქონების ძვრას ზევით ერთი სტრიქონით. ეს იმას ნიშნავს, რომ მეორე სტრიქონი დაიკავებს პირველის ადგილს, მესამე სტრიქონი დაიკავებს მეორის ადგილს და ა.შ. უკანასკნელი სტრიქონი შეივსება ნულებით.
- 4.2.49.** შეასრულებს $A_{5,5}$ მასივის სტრიქონების ძვრას ზევით ორი სტრიქონით.
- 4.2.50.** შეასრულებს $A_{5,5}$ მასივის სტრიქონების ძვრას ქვევით ერთი სტრიქონით. ეს იმას ნიშნავს, რომ პირველი სტრიქონი დაიკავებს მეორის ადგილს, მეორე სტრიქონი დაიკავებს მესამის ადგილს და ა.შ. პირველი სტრიქონი შეივსება ნულებით.

- 4.2.51.** შეასრულებს A_{5.5} მასივის სტრიქონების ძვრას ქვევით ორი სტრიქონით.
- 4.2.52.** შეასრულებს A_{5.5} მასივის სტრიქონების ციკლისებურ ძვრას ზევით ერთი სტრიქონით. ეს იმას ნიშნავს, რომ მეორე სტრიქონი დაიკავებს პირველის ადგილს, მესამე სტრიქონი დაიკავებს მეორის ადგილს და ა.შ. პირველი სტრიქონი დაიკავებს უკანასკნელის ადგილს.
- 4.2.53.** შეასრულებს A_{5.5} მასივის სტრიქონების ციკლისებურ ძვრას ზევით ორი სტრიქონით.
- 4.2.54.** შეასრულებს A_{5.5} მასივის სტრიქონების ციკლისებურ ძვრას ქვევით ერთი სტრიქონით. ეს იმას ნიშნავს, რომ პირველი სტრიქონი დაიკავებს მეორის ადგილს, მეორე სტრიქონი დაიკავებს მესამის ადგილს და ა.შ. უკანასკნელი სტრიქონი დაიკავებს პირველის ადგილს.
- 4.2.55.** შეასრულებს A_{5.5} მასივის სტრიქონების ციკლისებურ ძვრას ქვევით ორი სტრიქონით.
- 4.2.56.** შეასრულებს A_{5.5} მასივის სვეტების ძვრას მარჯვნივ ერთი სვეტით.
- 4.2.57.** შეასრულებს A_{5.5} მასივის სვეტების ძვრას მარჯვნივ ორი სვეტით.
- 4.2.58.** შეასრულებს A_{5.5} მასივის სვეტების ძვრას მარცხნივ ერთი სვეტით.
- 4.2.59.** შეასრულებს A_{5.5} მასივის სვეტების ძვრას მარცხნივ ორი სვეტით.
- 4.2.60.** შეასრულებს A_{5.5} მასივის სვეტების ციკლისებურ ძვრას მარცხნივ ერთი სვეტით.
- 4.2.61.** შეასრულებს A_{5.5} მასივის სვეტების ციკლისებურ ძვრას მარცხნივ ორი სვეტით.
- 4.2.62.** შეასრულებს A_{5.5} მასივის სვეტების ციკლისებურ ძვრას მარჯვნივ ერთი სვეტით.
- 4.2.63.** შეასრულებს A_{5.5} მასივის სვეტების ციკლისებურ ძვრას მარჯვნივ ორი სვეტით.
- 4.2.64.** იპოვის A_{5.5} მასივის იმ სტრიქონის ინდექსს, რომელიც შეიცავს 8-ის ჯერადი ელემენტების მაქსიმალურ რაოდენობას.
- 4.2.65.** იპოვის A_{5.5} მასივის იმ სვეტის ინდექსს, რომელიც შეიცავს დადებითი ელემენტების მინიმალურ რაოდენობას.
- 4.2.66.** იპოვის A_{5.5} მასივის იმ ელემენტების ჯამს, რომლებიც მთავარი დიაგონალის ზემოთაა მოთავსებული.
- 4.2.67.** იპოვის A_{5.5} მასივის იმ ელემენტების ნამრავლს, რომლებიც მთავარი დიაგონალის ქვემოთაა მოთავსებული.
- 4.2.68.** იპოვის A_{5.5} მასივის ორი ტოლი ელემენტის ინდექსებს.
- 4.2.69.** იპოვის A_{5.5} მასივის იმ ელემენტს, რომელიც ყველა სტრიქონში შედის.
- 4.2.70.** იპოვის A_{5.5} მასივის იმ ელემენტს, რომელიც ყველა სვეტში შედის.
- 4.2.71.** იპოვის A_{5.5} მასივის არამთავარი დიაგონალის ზემოთ მოთავსებული ელემენტების ჯამს.
- 4.2.72.** იპოვის A_{5.5} მასივის არამთავარი დიაგონალის ქვემოთ მოთავსებული ელემენტების ჯამს.
- 4.2.73.** იპოვის A_{5.5} მასივის იმ სტრიქონის ნომერს, რომლის ელემენტების ჯამი აღემატება დანარჩენი სტრიქონების ელემენტების ჯამს.
- 4.2.74.** იპოვის A_{5.5} მასივის იმ სვეტის ნომერს, რომლის ელემენტების ჯამი აღემატება დანარჩენი სვეტების ელემენტების ჯამს.
- 4.2.75.** განსაზღვრავს არის თუ არა A_{5.5} მასივში ერთნაირი სვეტები.
- 4.2.76.** განსაზღვრავს არის თუ არა A_{5.5} მასივში ერთნაირი სტრიქონები.
- 4.2.77.** A_{5.5} მასივიდან წაშლის იმ სვეტსა და სტრიქონს, რომელთა გადაკვეთაზე მდებარეობს მთავარი დიაგონალის მაქსიმალური ელემენტი.
- 4.2.78.** A_{5.5} მასივის რომელიმე სვეტსა და სტრიქონს შეავსებს ნულებით მათ გადაკვეთაზე მდებარე ელემენტის გარდა.
- 4.2.79.** იპოვის A_{5.5} მასივის უნაგირა წერტილს. უნაგირა წერტილი არის მასივის ის ელემენტი, რომელიც უმცირესია თავის სტრიქონში და უდიდესია თავის სვეტში.
- 4.2.80.** იპოვის A_{5.5} მასივის იმ ელემენტების ჯამს, რომლებიც არამთავარი დიაგონალის ქვემოთაა მოთავსებული.
- 4.2.81.** იპოვის A_{5.5} მასივის იმ ელემენტების ჯამს, რომლებიც არამთავარი დიაგონალის ზემოთაა მოთავსებული.

შემთხვევითი რიცხვების გენერატორი

შეადგინეთ პროგრამა, რომელიც:

- 4.3.1. მოახდენს [1,100] ინტერვალში მოთავსებული 20 შემთხვევითი რიცხვის გენერირებას.
- 4.3.2. დათვლის თუ რამდენჯერ გვხვდება q რიცხვი [1,1000] ინტერვალში 50 მცდელობის შემდეგ.
- 4.3.3. ჯერ მოახდენს 100 მთელი რიცხვის გენერირებას [1,100] ინტერვალში, შემდეგ კი დათვლის თუ რამდენჯერ გამოჩნდა თითოეული რიცხვი ამ ინტერვალში.
- 4.3.4. განსაზღვრავს თუ რამდენი მთელი რიცხვის გენერირება უნდა მოვახდინოთ [1,30] ინტერვალიდან, რომ მოცემული R რიცხვი მათ შორის 7-ჯერ შეგვხვდეს.
- 4.3.5. განსაზღვრავს თუ რამდენი ასანთი უნდა ამოვიღოთ ასანთის 5 ყუთიდან იმისათვის, რომ დაცარიელდეს ერთ-ერთი. თითო ყუთში 20 ასანთია.

თავი 5. სტრიქონები

სტრიქონი

შეადგინეთ პროგრამა, რომელიც:

- 5.1.1. სტრიქონში დათვლის სიმბოლოების რაოდენობას.
- 5.1.2. სტრიქონში დათვლის "a" სიმბოლოს რაოდენობას.
- 5.1.3. სტრიქონში დათვლის ციფრების რაოდენობას 0-დან 9-მდე.
- 5.1.4. სტრიქონში დათვლის სასვენი ნიშნების რაოდენობას (, ; ! ? :).
- 5.1.5. სტრიქონში დათვლის ხმოვნების რაოდენობას (a, i, o, u, e).
- 5.1.6. სტრიქონში იპოვის "a" სიმბოლოს პოზიციას.
- 5.1.7. სტრიქონში ყველა "i" სიმბოლოს "s" სიმბოლოთი შეცვლის.
- 5.1.8. სტრიქონში ყველა ციფრს "r" სიმბოლოთი შეცვლის.
- 5.1.9. სტრიქონში "This is a computer" ჩაუმატებს "good" სიტყვას დაწყებული მე-10 პოზიციიდან.
- 5.1.10. სტრიქონიდან "This is a computer" წაშლის "is" სიტყვას.
- 5.1.11. სტრიქონიდან "This is a computer" წაშლის 5 სიმბოლოს დაწყებული მე-5 პოზიციიდან.
- 5.1.12. სტრიქონში "This is C#" "C#" სიტყვას შეცვლის "C++" სიტყვით.
- 5.1.13. სტრიქონში "This is a test program" "This is" სიტყვებს შეცვლის "Now execute" სიტყვებით.

სხვადასხვა

შეადგინეთ პროგრამა, რომელიც:

- 5.2.1. სიმბოლოების M_{5,5} მასივში მოძებნის იმ სტრიქონს, რომელიც შეიცავს "a" სიმბოლოს მაქსიმალურ რაოდენობას.
- 5.2.2. სიმბოლოების M_{5,5} მასივში მოძებნის იმ სტრიქონს, რომელიც შეიცავს განსხვავებული სიმბოლოების მაქსიმალურ რაოდენობას.
- 5.2.3. სიმბოლოების M_{5,5} მასივში მოძებნის იმ სვეტს, რომელიც შეიცავს "a" სიმბოლოს მინიმალურ რაოდენობას.
- 5.2.4. სიმბოლოების M_{5,5} მასივში მოძებნის იმ სვეტს, რომელიც შეიცავს განსხვავებული სიმბოლოების მინიმალურ რაოდენობას.
- 5.2.5. სტრიქონის მე-10 პოზიციიდან 30-ე პოზიციამდე დათვლის "+" სიმბოლოს რაოდენობას.
- 5.2.6. სტრიქონში დათვლის "ro" სიმბოლოების რაოდენობას.
- 5.2.7. სტრიქონში დათვლის ინტერვალების რაოდენობას.
- 5.2.8. განსაზღვრავს შეიცავს თუ არა სტრიქონი "d" სიმბოლოს.
- 5.2.9. განსაზღვრავს შეიცავს თუ არა სტრიქონი ერთნაირი სიმბოლოების წყვილებს.
- 5.2.10. განსაზღვრავს შეიცავს თუ არა სტრიქონი 4 მეზობელ "k" სიმბოლოს.

- 5.2.11. სტრიქონში განსაზღვრავს მეზობელი ინტერვალების ყველაზე გრძელი მიმდევრობის ზომას.
- 5.2.12. სტრიქონიდან წაშლის "ten" სიტყვებს.
- 5.2.13. სტრიქონში "s" სიმბოლოს შემდეგ მოთავსებულ ყველა სიმბოლოს "t" სიმბოლოთი შეცვლის.
- 5.2.14. ინგლისური ანბანის ყველა ასოს გამოიტანს.
- 5.2.15. კლავიატურის ყველა სიმბოლოს გამოიტანს.
- 5.2.16. სტრიქონში დათვლის სიტყვების რაოდენობას.
- 5.2.17. სტრიქონის უკანასკნელ სიტყვაში "L" სიმბოლოს რაოდენობას დათვლის.
- 5.2.18. სტრიქონში იმ სიტყვების რაოდენობას დათვლის, რომლებიც "o" სიმბოლოთი იწყება.
- 5.2.19. სტრიქონში იმ სიტყვების რაოდენობას დათვლის, რომლებიც "shvili" სიმბოლოებით მთავრდება.
- 5.2.20. სტრიქონში იმ სიტყვების რაოდენობას დათვლის, რომელთა პირველი და უკანასკნელი ასოები ერთმანეთს ემთხვევა.
- 5.2.21. სტრიქონში ყველაზე გრძელ სიტყვას იპოვის.
- 5.2.22. სტრიქონში ყველაზე მოკლე სიტყვას იპოვის.
- 5.2.23. სტრიქონში პატარა ასოებს დიდი ასოებით შეცვლის.
- 5.2.24. სტრიქონში დიდ ასოებს პატარა ასოებით შეცვლის.
- 5.2.25. სტრიქონში წაშლის წერტილის წინ მოთავსებულ რიცხვებს.
- 5.2.26. სტრიქონში წაშლის წერტილის შემდეგ მოთავსებულ რიცხვებს.
- 5.2.27. სტრიქონის სიმბოლოებს უკუმიმდევრობით დაალაგებს.
- 5.2.28. ტექსტში მრგვალი ფრჩხილების ბალანსს შეამოწმებს. მრგვალი ფრჩხილები დაბალანსებულია თუ გამხსნელ ფრჩხილს მოსდევს შესაბამისი დამხურავი ფრჩხილი და მათი რაოდენობა ტოლია.
- 5.2.29. შეამოწმებს არის თუ არა ერთი წინადადება მეორის ნაწილი და პირიქით.
- 5.2.30. შეამოწმებს ერთნაირია თუ არა ორი წინადადება.
- 5.2.31. ორ წინადადებაში მოძებნის იმ სიტყვებს, რომლებიც ორივე წინადადებაში შედიან და ამ სიტყვებს ჩაწერს სტრიქონების მასივში.
- 5.2.32. მოცემულ რიცხვს (1-დან 10-მდე) ჩაწერს სიტყვების საშუალებით.
- 5.2.33. სტრიქონში მოძებნის იმ სიტყვებს, რომლებიც "dze" სიმბოლოებით მთავრდება და ამ სიტყვებს ჩაწერს სტრიქონების მასივში.
- 5.2.34. განსაზღვრავს ტექსტში რამდენი სიტყვა შეიცავს 1 ხმოვანს, 2 ხმოვანს, 3 ხმოვანს და ა.შ.
- 5.2.35. განსაზღვრავს ტექსტში სიტყვების რამდენი პროცენტი იწყება "F" სიმბოლოთი.
- 5.2.36. განსაზღვრავს შეიცავს თუ არა ტექსტი ასოებისაგან განსხვავებულ სიმბოლოებს.
- 5.2.37. ტექსტში იპოვის იმ სიტყვას, რომელიც შეიცავს ხმოვნების მაქსიმალურ რაოდენობას.
- 5.2.38. ტექსტში იპოვის იმ სიტყვას, რომელიც შეიცავს თანხმოვნების მინიმალურ რაოდენობას.
- 5.2.39. ტექსტში იპოვის იმ სიტყვას, რომელიც შეიცავს "g" სიმბოლოს მაქსიმალურ რაოდენობას.
- 5.2.40. ტექსტში იპოვის სიტყვას, რომელიც ყველაზე ხშირად გვხვდება.
- 5.2.41. ტექსტში იპოვის 10 სიტყვას, რომელიც ყველაზე ხშირად გვხვდება და დათვლის თითოეული მათგანის რაოდენობას.
- 5.2.42. ტექსტში განსაზღვრავს თუ რომელი ასოთი იწყება სიტყვების უმრავლესობა.
- 5.2.43. ტექსტში განსაზღვრავს თუ რამდენჯერ გვხვდება ანბანის თითოეული ასო.
- 5.2.44. ტექსტში განსაზღვრავს სიტყვების პირველი ასოების მინიმალურ რაოდენობას, რომელთა მიხედვით შეიძლება განვასხვავოთ სიტყვები.
- 5.2.45. ტექსტში იპოვის იმ სიტყვებს, რომლებიც შეიცავენ გაორმაგებულ ხმოვნებს.
- 5.2.46. ტექსტში იპოვის იმ სიტყვებს, რომლებიც შეიცავენ გაორმაგებულ თანხმოვნებს.
- 5.2.47. ტექსტიდან წაშლის ზედმეტ ინტერვალის ნიშნებს და სიტყვებს შორის დატოვებს

ინტერვალის თითო ნიშანს.

5.2.48. მოცემულია ორი სიტყვა. ისინი გააერთიანეთ ისე, რომ წინ მოთავსდეს მოკლე სიტყვა, შემდეგ კი - გრძელი.

5.2.49. მოცემულია ორი სიტყვა. მეორე სიტყვა მოათავსეთ პირველი სიტყვის შუაში.

5.2.50. დაადგინეთ რამდენი პალინდრომია მოცემულ ტექსტში. პალინდრომია, მაგალითად, "abccba", 123321 და ა.შ.

თავი 6. ინფორმაციის შეტანა-გამოტანა

შეადგინეთ პროგრამა, რომელიც:

6.1.1. ერთი ფაილიდან წაიკითხავს 5 მთელ რიცხვს, მოათავსებს მათ ერთგანზომილებიან მასივში და ამ მასივის ელემენტების ჯამს ჩაწერს მეორე ფაილში.

6.1.2. ერთი ფაილიდან წაიკითხავს 5 მთელ რიცხვს, მოათავსებს მათ ერთგანზომილებიან მასივში და ამ მასივის ელემენტებს და მათ ჯამს ჩაწერს მეორე ფაილში.

6.1.3. ერთი ფაილიდან მეორეში სათითაოდ გადაწერს 10 მთელ რიცხვს.

6.1.4. ერთი ფაილიდან წაიკითხავს 5 წილად რიცხვს, მოათავსებს მათ ერთგანზომილებიან მასივში და ამ მასივის ელემენტების ჯამს ჩაწერს მეორე ფაილში.

6.1.5. ერთი ფაილიდან წაიკითხავს 5 წილად რიცხვს, მოათავსებს მათ ერთგანზომილებიან მასივში და ამ მასივის ელემენტებს და მათ ჯამს ჩაწერს მეორე ფაილში.

6.1.6. ერთი ფაილიდან მეორეში სათითაოდ გადაწერს 10 წილად რიცხვს.

6.1.7. ერთი ფაილიდან წაიკითხავს 5 სიმბოლოს, მოათავსებს მათ სიმბოლოების მასივში და მასივის ელემენტებს ჩაწერს მეორე ფაილში.

6.1.8. ერთი ფაილიდან მეორეში სათითაოდ გადაწერს 10 სიმბოლოს.

6.1.9. ერთი ფაილიდან წაიკითხავს სტრიქონს და ჩაწერს მას მეორე ფაილში.

6.1.10. ერთი ფაილიდან წაიკითხავს სტრიქონებს და ჩაწერს მათ სტრიქონების მასივში. შემდეგ ამ მასივიდან სტრიქონებს ჩაწერს მეორე ფაილში.

6.1.11. ერთი ფაილიდან მეორეში სათითაოდ გადაწერს 10 წილად რიცხვს. შემდეგ, გადაწერილ რიცხვებს დაუმატებს მათ ჯამს.

6.1.12. ერთი ფაილიდან მეორეში სათითაოდ გადაწერს 10 მთელ რიცხვს. შემდეგ, გადაწერილ რიცხვებს დაუმატებს მათ ნამრავლს.

სხვადასხვა

შეადგინეთ პროგრამა, რომელიც:

6.2.1. ფაილიდან წაიკითხავს მთელ რიცხვებს და დათვლის მათ შორის უარყოფითი, ნულოვანი და დადებითი რიცხვების რაოდენობას.

6.2.2. ფაილიდან წაიკითხავს წილად რიცხვებს, დათვლის მათ რაოდენობას და გამოთვლის მათ ჯამს.

6.2.3. წაიკითხავს ფაილში ჩაწერილი წილადი რიცხვების შუაში მოთავსებულ რიცხვს. ფაილში ჩაწერილია კენტი რაოდენობის რიცხვი.

6.2.4. ფაილიდან წაიკითხავს მთელ რიცხვებს, დაალაგებს მათ ზრდადობის მიხედვით და ჩაწერს ამავე ფაილში.

6.2.5. ფაილიდან წაიკითხავს რიცხვებს და მათ შორის პირველსა და უკანასკნელს მეორე ფაილში გადაწერს.

6.2.6. ფაილიდან წაიკითხავს რიცხვებს და მეორე ფაილში გადაწერს:

ა. მათ შორის მინიმალურსა და მაქსიმალურს;

ბ. კენტი ინდექსის მქონე რიცხვებს შორის უდიდესს;

გ. ლუწი ინდექსის მქონე რიცხვებს შორის უმცირესს.

- დ. ლუწი და კენტი რიცხვების რაოდენობას;
- 6.2.7.** ფაილიდან ორ მონაცემს კითხულობს. თუ ეს მონაცემები ციფრებია, მაშინ მათი ჯამი, ნამრავლი და სხვაობა მეორე ფაილში ჩაწერეთ.
- 6.2.8.** ფაილის თითოეულ სტრიქონში დათვლის სიტყვების რაოდენობას.
- 6.2.9.** ფაილიდან წაიკითხავს რიცხვებს და მეორე ფაილში გადაწერს:
- ა. კენტ რიცხვებს;
 - ბ. ლუწ რიცხვებს;
 - გ. 7-ის ჯერად რიცხვებს;
 - დ. რიცხვებს, რომლებიც წარმოადგენენ კვადრატებს.
- 6.2.10.** განსაზღვრავს ორი ფაილი შეიცავს თუ არა ერთნაირ რიცხვებს.
- 6.2.11.** ერთი ფაილიდან მეორეში მონაცემებს ისე გადაწერს, რომ მიმდევრობით მოთავსებული რამდენიმე ინტერვალის შეიცვალოს ერთით.
- 6.2.12.** ფაილის ყოველი სტრიქონის წინ ინტერვალის ნიშანს ჩასვამს და გადაწერს მეორე ფაილში.
- 6.2.13.** ფაილის ყოველი სტრიქონის წინ და ბოლოში მოთავსებულ ინტერვალებს წაშლის და მიღებულ სტრიქონებს ამავე ფაილში ჩაწერს.
- 6.2.14.** ფაილში:
- ა. დათვლის 'a' სიმბოლოს რაოდენობას;
 - ბ. დათვლის 's', 'r' და 'e' სიმბოლოების რაოდენობას;
 - გ. დაადგენს შეიცავს თუ არა ფაილი "computer" სიმბოლოების მიმდევრობას;
 - დ. დათვლის "computer" სიტყვის რაოდენობას.
- 6.2.15.** ფაილიდან წაშლის ყველა სამსიმბოლოიან სიტყვას და შედეგს მეორე ფაილში გადაწერს.
- 6.2.16.** ფაილში:
- ა. იპოვის ყველაზე გრძელ სიტყვას;
 - ბ. იპოვის ყველაზე მოკლე სიტყვას;
 - გ. იპოვის ყველაზე გრძელ სიტყვას, რომლის მესამე სიმბოლოა 'h';
 - დ. დათვლის ორი, სამი და ოთხი სიმბოლოსაგან შემდგარი სიტყვების რაოდენობას.
- 6.2.17.** ერთ ფაილს მეორეში გადაწერს.
- 6.2.18.** ერთი ფაილიდან წაიკითხავს რამდენიმე სტრიქონს, პატარა ასოებს დიდ ასოებად გარდაქმნის და შედეგს მეორე ფაილში ჩაწერს.
- 6.2.19.** ერთი ფაილიდან მონაცემებს უკუ მიმდევრობით გადაწერს მეორე ფაილში.
- 6.2.20.** ფაილში მოძებნის 9 სიტყვას, რომლებიც ყველაზე ხშირად გვხვდება.
- 6.2.21.** ფაილში დათვლის სტრიქონების რაოდენობას.
- 6.2.22.** ფაილის თითოეულ სტრიქონში დათვლის სიმბოლოების რაოდენობას.
- 6.2.23.** ეკრანზე გამოიტანს ფაილის ყველაზე გრძელ და მოკლე სტრიქონებს.
- 6.2.24.** ფაილში დათვლის იმ სტრიქონების რაოდენობას, რომლებშიც სიმბოლოების რაოდენობა 60-ს აღემატება.
- 6.2.25.** ფაილში მოთავსებულ ტექსტს ეკრანზე გამოიტანს.
- 6.2.26.** ფაილში ჩაწერს 9 სტუდენტის გვარს, დაბადების წელსა და სიმაღლეს.
- 6.2.27.** მოცემულია ორი ფაილი, რომლებიც შეიცავენ ზრდადობით დალაგებულ რიცხვებს. შეადგინეთ პროგრამა, რომელიც მესამე ფაილს შექმნის და მასში ზრდადობის მიხედვით ჩაწერს ორივე ფაილში მოთავსებულ რიცხვებს.
- 6.2.28.** მოცემულია ორი ფაილი. შეადგინეთ პროგრამა, რომელიც მესამე ფაილში ჯერ ჩაწერს პირველ ფაილში მოთავსებულ ტექსტს, შემდეგ კი - მეორე ფაილში მოთავსებულ ტექსტს.
- 6.2.29.** ფაილი შეიცავს C++ ენაზე შედგენილ საწყის კოდს (პროგრამას). კოდის თითოეული ოპერატორი თითო სტრიქონს იკავებს. შეადგინეთ პროგრამა, რომელიც:
- ა. შეამოწმებს გამხსნელი და დამხურავი მრგვალი ფრჩხილების გამოყენების სისწორეს;

ბ. შეამოწმებს გამხსნელი და დამხურავი ფიგურული ფრჩხილების გამოყენების სისწორეს;

თავი 7. ფუნქციები

ფუნქციები

შეადგინეთ ფუნქცია, რომელიც გამოთვლის და აბრუნებს:

- 7.1.1. მართკუთხა სამკუთხედის ფართობს.
- 7.1.2. სამკუთხედის პერიმეტრს.
- 7.1.3. კვადრატის ფართობს.
- 7.1.4. კვადრატის პერიმეტრს.
- 7.1.5. მართკუთხედის ფართობს.
- 7.1.6. მართკუთხედის პერიმეტრს.
- 7.1.7. წრეწირის ფართობს.
- 7.1.8. წრეწირის სიგრძეს.
- 7.1.9. ტრაპეციის ფართობს.
- 7.1.10. ტრაპეციის პერიმეტრს.
- 7.1.11. გასცემს ორ რიცხვს შორის მაქსიმალურს.
- 7.1.12. გასცემს ორ რიცხვს შორის მინიმალურს.
- 7.1.13. დაადგენს ორ რიცხვს შორის არის თუ არა უარყოფითი და გასცემს შესაბამის შეტყობინებას.
- 7.1.14. დაადგენს ორ რიცხვს შორის არის თუ არა კენტი და გასცემს შესაბამის შეტყობინებას.
- 7.1.15. დაადგენს ორ რიცხვს შორის არის თუ არა 5-ის ჯერადი და გასცემს შესაბამის შეტყობინებას.
- 7.1.16. დაადგენს x რიცხვი არის თუ არა $0 < x \leq 17$ დიაპაზონში მოთავსებული და გასცემს შესაბამის შეტყობინებას.
- 7.1.17. დაადგენს x რიცხვი არის თუ არა 25-ზე ნაკლები ან 100-ზე მეტი და გასცემს შესაბამის შეტყობინებას.
- 7.1.18. დათვლის ლუწი რიცხვების რაოდენობას 5-დან 27-მდე დიაპაზონში.
- 7.1.19. შეკრებს კენტი რიცხვებს 10-დან 20-მდე დიაპაზონში.
- 7.1.20. შეკრებს 1-დან 20-მდე რიცხვებს მანამ, სანამ ჯამი არ გახდება 24-ზე მეტი.
- 7.1.21. შეკრებს 9-ის ჯერად რიცხვებს 2-დან 47-მდე დიაპაზონში მანამ, სანამ 9-ის ჯერადი რიცხვი იქნება 30-ზე ნაკლები.

ფუნქციისთვის მასივების გადაცემა

- 7.2.1. ფუნქციას გადაეცით ერთგანზომილებიანი მასივი. ფუნქცია გამოთვლის და აბრუნებს მასივის ელემენტების ჯამს.
- 7.2.2. ფუნქციას გადაეცით ერთგანზომილებიანი მასივი. ფუნქცია გამოთვლის და აბრუნებს მასივის ელემენტების ნამრავლს.
- 7.2.3. ფუნქციას გადაეცით ერთგანზომილებიანი მასივი. ფუნქცია გამოთვლის და აბრუნებს მასივის ლუწი ელემენტების ჯამს.
- 7.2.4. ფუნქციას გადაეცით ერთგანზომილებიანი მასივი. ფუნქცია გამოთვლის და აბრუნებს მასივის კენტი ელემენტების რაოდენობას.
- 7.2.5. ფუნქციას გადაეცით ერთგანზომილებიანი მასივი. ფუნქცია გამოთვლის და აბრუნებს მასივის დადებითი ელემენტების ჯამს.
- 7.2.6. ფუნქციას გადაეცით ერთგანზომილებიანი მასივი. ფუნქცია გამოთვლის და აბრუნებს მასივის უარყოფითი ელემენტების რაოდენობას.

7.3.10. შეადგინეთ ფუნქცია, რომელიც გამოთვლის წრის ფართობსა და პერიმეტრს. ფუნქცია პერიმეტრს გვიბრუნებს ჩვეულებრივი გზით, ფართობს კი - მიმართვის საშუალებით.

7.3.11. შეადგინეთ ფუნქცია, რომელიც ადგილს გაუცვლის ორ მთელ რიცხვს. პარამეტრები გადაეცით მიმართვის საშუალებით.

7.3.12. შეადგინეთ ფუნქცია, რომელიც ადგილს გაუცვლის ორ წილად რიცხვს. პარამეტრები გადაეცით მიმართვის საშუალებით.

ფუნქციის გადატვირთვა

7.4.1. გადატვირთეთ ფუნქცია, რომელსაც გადაეცემა ორი პარამეტრი და აბრუნებს მათ შორის მაქსიმალურს. ფუნქციის გამოძახება შესაძლებელი უნდა იყოს ორი მთელი და ორი წილადი რიცხვისთვის.

7.4.2. გადატვირთეთ ფუნქცია, რომელსაც გადაეცემა ორი პარამეტრი და აბრუნებს მათ შორის მინიმალურს. ფუნქციის გამოძახება შესაძლებელი უნდა იყოს ორი მთელი და ორი წილადი რიცხვისთვის.

7.4.3. გადატვირთეთ ფუნქცია, რომელსაც გადაეცემა ორი პარამეტრი და აბრუნებს მათ ჯამს. ფუნქციის გამოძახება შესაძლებელი უნდა იყოს ორი მთელი და ორი წილადი რიცხვისთვის.

7.4.4. გადატვირთეთ ფუნქცია, რომელსაც გადაეცემა ორი პარამეტრი და აბრუნებს მათ ნამრავლს. ფუნქციის გამოძახება შესაძლებელი უნდა იყოს ორი მთელი და ორი წილადი რიცხვისთვის.

7.4.5. გადატვირთეთ ფუნქცია, რომელსაც გადაეცემა სამი პარამეტრი და აბრუნებს მათ ჯამს. ფუნქციის გამოძახება შესაძლებელი უნდა იყოს სამი მთელი და სამი წილადი რიცხვისთვის.

7.4.6. გადატვირთეთ ფუნქცია, რომელსაც გადაეცემა სამი პარამეტრი და აბრუნებს მათ ნამრავლს. ფუნქციის გამოძახება შესაძლებელი უნდა იყოს სამი მთელი და სამი წილადი რიცხვისთვის.

7.4.7. გადატვირთეთ ფუნქცია, რომელსაც გადაეცემა ერთი პარამეტრი. ფუნქცია განსაზღვრავს არის თუ არა პარამეტრი დადებითი. ფუნქციის გამოძახება შესაძლებელი უნდა იყოს ერთი მთელი და ერთი წილადი რიცხვისთვის.

7.4.8. გადატვირთეთ ფუნქცია, რომელსაც გადაეცემა ერთი პარამეტრი. ფუნქცია განსაზღვრავს არის თუ არა პარამეტრი უარყოფითი. ფუნქციის გამოძახება შესაძლებელი უნდა იყოს ერთი მთელი და ერთი წილადი რიცხვისთვის.

7.4.9. გადატვირთეთ ფუნქცია, რომელსაც გადაეცემა ერთი პარამეტრი. ფუნქცია აბრუნებს პარამეტრის კვადრატს. ფუნქციის გამოძახება შესაძლებელი უნდა იყოს ერთი მთელი და ერთი წილადი რიცხვისთვის.

7.4.10. გადატვირთეთ ფუნქცია, რომელსაც გადაეცემა ერთი პარამეტრი. ფუნქცია აბრუნებს პარამეტრის კუბს. ფუნქციის გამოძახება შესაძლებელი უნდა იყოს ერთი მთელი და ერთი წილადი რიცხვისთვის.

7.4.11. გადატვირთეთ ფუნქცია, რომელსაც გადაეცემა ორი პარამეტრი და აბრუნებს მათ საშუალო არითმეტიკულს. ფუნქციის გამოძახება შესაძლებელი უნდა იყოს ორი მთელი და ორი წილადი რიცხვისთვის.

ფუნქციისთვის პარამეტრების გადაცემა გულისხმობის პრინციპით

7.5.1. შექმენით ფუნქცია, რომელსაც სამი მთელი ტიპის პარამერი აქვს. ერთი პარამეტრი ნაგულისხმევია. ფუნქცია გამოთვლის და აბრუნებს ორი ან სამი რიცხვის ჯამს. ფუნქცია გამოიძახეთ ორი არგუმენტის მითითებით, სამი არგუმენტის მითითებით.

- 7.5.2.** შექმენით ფუნქცია, რომელსაც სამი მთელი ტიპის პარამერი აქვს. ერთი პარამეტრი ნაგულისხმევი. ფუნქცია გამოთვლის და აბრუნებს ორი ან სამი რიცხვის ნამრავლს. ფუნქცია გამოიძახეთ ორი არგუმენტის მითითებით, სამი არგუმენტის მითითებით.
- 7.5.3.** შექმენით ფუნქცია, რომელსაც ერთი მთელი ტიპის ნაგულისხმევი პარამეტრი აქვს. ფუნქცია გამოთვლის და აბრუნებს ამ პარამეტრის კვადრატს. ფუნქცია გამოიძახეთ უარგუმენტოდ, ერთი არგუმენტის მითითებით.
- 7.5.4.** შექმენით ფუნქცია, რომელსაც ორი მთელი ტიპის ნაგულისხმევი პარამეტრი აქვს. ფუნქცია გამოთვლის და აბრუნებს ამ პარამეტრების ჯამს. ფუნქცია გამოიძახეთ უარგუმენტოდ, ერთი არგუმენტის მითითებით, ორი არგუმენტის მითითებით.
- 7.5.5.** შექმენით ფუნქცია, რომელსაც სამი მთელი ტიპის ნაგულისხმევი პარამეტრი აქვს. ფუნქცია გამოთვლის და აბრუნებს ამ პარამეტრების ჯამს. ფუნქცია გამოიძახეთ უარგუმენტოდ, ერთი არგუმენტის მითითებით, ორი არგუმენტის მითითებით, სამი არგუმენტის მითითებით.
- 7.5.6.** შექმენით ფუნქცია, რომელსაც სამი წილადი ტიპის პარამერი აქვს. ერთი პარამეტრი ნაგულისხმევი. ფუნქცია გამოთვლის და აბრუნებს ორი ან სამი რიცხვის ნამრავლს. ფუნქცია გამოიძახეთ ორი არგუმენტის მითითებით, სამი არგუმენტის მითითებით.
- 7.5.7.** შექმენით ფუნქცია, რომელსაც სამი წილადი ტიპის პარამერი აქვს. ერთი პარამეტრი ნაგულისხმევი. ფუნქცია გამოთვლის და აბრუნებს ორი ან სამი რიცხვის ნამრავლს. ფუნქცია გამოიძახეთ ორი არგუმენტის მითითებით, სამი არგუმენტის მითითებით.
- 7.5.8.** შექმენით ფუნქცია, რომელსაც ერთი წილადი ტიპის ნაგულისხმევი პარამეტრი აქვს. ფუნქცია გამოთვლის და აბრუნებს ამ პარამეტრის კვადრატს. ფუნქცია გამოიძახეთ უარგუმენტოდ, ერთი არგუმენტის მითითებით.
- 7.5.9.** შექმენით ფუნქცია, რომელსაც ორი წილადი ტიპის ნაგულისხმევი პარამეტრი აქვს. ფუნქცია გამოთვლის და აბრუნებს ამ პარამეტრების ნამრავლს. ფუნქცია გამოიძახეთ უარგუმენტოდ, ერთი არგუმენტის მითითებით, ორი არგუმენტის მითითებით.
- 7.5.10.** შექმენით ფუნქცია, რომელსაც სამი წილადი ტიპის ნაგულისხმევი პარამეტრი აქვს. ფუნქცია გამოთვლის და აბრუნებს ამ პარამეტრების ნამრავლს. ფუნქცია გამოიძახეთ უარგუმენტოდ, ერთი არგუმენტის მითითებით, ორი არგუმენტის მითითებით, სამი არგუმენტის მითითებით.

თავი 8. კლასები, ობიექტები, მეთოდები

კლასი. ინკაფსულაცია

- 8.1.1.** შექმენით თვითმფრინავის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ავზის ტევადობა და მანძილი, რომელსაც თვითმფრინავი 1 ლიტრი საწვავით გაიფრენს; საერთო წვდომის ცვლადებს: მგზავრების რაოდენობა და გაყიდული ბილეთების რაოდენობა.
- 8.1.2.** შექმენით სტუდენტის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: გვარი, სახელი და ასაკი; საერთო წვდომის ცვლადებს: უნივერსიტეტის დასახელება და კურსი.
- 8.1.3.** შექმენით მატარებლის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ვაგონების რაოდენობასა და მგზავრების რაოდენობას 1 ვაგონში; საერთო წვდომის ცვლადებს: ბილეთების ფასს და გაყიდული ბილეთების რაოდენობას.
- 8.1.4.** შექმენით სამკუთხედის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: პერიმეტრი და ფართობი; საერთო წვდომის ცვლადებს: სამკუთხედის სამივე გვერდის ზომებს.

ფუნქცია

- 8.2.1.** შექმენით სტუდენტის კლასი, რომელიც შეიცავს ღია ფუნქციას. ფუნქციას გადაეცემა სტუდენტის ნიშნები, რომელიც 10-ელემენტიაანი მთელრიცხვა მასივია, ფუნქცია გამოთვლის და აბრუნებს ნიშნების საშუალო არითმეტიკულს.

8.2.2. შექმენით სტუდენტის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: გვარი, სახელი და ასაკი; პრივატულ ფუნქციას, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; ღია ფუნქციებს: პირველი ფუნქცია იმახებს პრივატულ ფუნქციას და გადასცემს მნიშვნელობებს პრივატული ცვლადებისათვის მისანიჭებლად; მეორე ფუნქციას გამოაქვს პრივატული ცვლადების მნიშვნელობები.

8.2.3. შექმენით მატარებლის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ვაგონების რაოდენობასა და მგზავრების რაოდენობას 1 ვაგონში; საერთო წვდომის ცვლადებს: ბილეთების ფასს და გაყიდული ბილეთების რაოდენობას. შეიცავს ღია ფუნქციებს: პირველი ფუნქცია პრივატულ და ღია ცვლადებს მნიშვნელობებს ანიჭებს; მეორე ფუნქციას გამოაქვს პრივატული და ღია ცვლადების მნიშვნელობები; მესამე ფუნქცია გამოთვლის და აბრუნებს ბილეთების გაყიდვით მიღებულ თანხას.

8.2.4. შექმენით თვითმფრინავის კლასი, რომელიც შეიცავს ღია ცვლადებს: ავზის ტევადობა და მანძილი, რომელსაც გაიფრენს თვითმფრინავი 1 ლიტრი საწვავით; ღია ფუნქციებს: პირველი ფუნქცია ღია ცვლადებს მნიშვნელობებს ანიჭებს; მეორე ფუნქციას გამოაქვს ღია ცვლადების მნიშვნელობები; მესამე ფუნქცია გამოთვლის და აბრუნებს თვითმფრინავის მიერ სავსე ავზით გავლილ მანძილს.

6.2.5. შექმენით მართკუთხედის კლასი, რომელიც შეიცავს ღია ცვლადებს: მართკუთხედის ოთხივე გვერდის ზომებს; პრივატულ ფუნქციას, რომელიც ღია ცვლადებს მნიშვნელობებს ანიჭებს; ღია ფუნქციებს: პირველი ფუნქცია იმახებს პრივატულ ფუნქციას, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე ფუნქციას გამოაქვს პრივატული ცვლადების მნიშვნელობები; მესამე ფუნქცია გამოთვლის და აბრუნებს მართკუთხედის ფართობს.

კონსტრუქტორი

8.3.1. შექმენით თვითმფრინავის კლასი, რომელიც შეიცავს პრივატულ ცვლადებს: ავზის ტევადობა და მანძილი, რომელსაც გაიფრენს თვითმფრინავი 1 ლიტრი საწვავით; ღია ფუნქციებს: პირველი ფუნქციაა კონსტრუქტორი, რომელიც პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე ფუნქციას გამოაქვს პრივატული ცვლადების მნიშვნელობები; მესამე ფუნქცია გამოთვლის და აბრუნებს თვითმფრინავის მიერ სავსე ავზით გავლილ მანძილს.

8.3.2. შექმენით სამკუთხედის კლასი, რომელიც შეიცავს ღია ცვლადებს: სამკუთხედის სამივე გვერდის ზომებს; პრივატულ ცვლადებს: სამკუთხედის პერიმეტრსა და ფართობს; ღია ფუნქციებს: პირველი ფუნქციაა კონსტრუქტორი, რომელიც ღია და პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე ფუნქციას გამოაქვს პრივატული ცვლადების მნიშვნელობები.

8.3.3. შექმენით მართკუთხედის კლასი, რომელიც შეიცავს ღია ცვლადებს: მართკუთხედის ოთხივე გვერდის ზომებს; პრივატულ ცვლადებს: მართკუთხედის ფართობს და პერიმეტრს; ღია ფუნქციებს: პირველი ფუნქციაა კონსტრუქტორი, რომელიც ღია და პრივატულ ცვლადებს მნიშვნელობებს ანიჭებს; მეორე ფუნქციას გამოაქვს პრივატული ცვლადების მნიშვნელობები.

ფუნქციების გადატვირთვა

8.4.1. შეადგინეთ სამკუთხედის კლასი, რომელშიც განსაზღვრულია ერთი და იმავე სახელის მქონე 2 ფუნქცია. პირველ ფუნქციას 2 მთელირიცხვა პარამეტრი აქვს: სამკუთხედის სიმაღლე და ფუძე და აბრუნებს მართკუთხა სამკუთხედის ფართობს. მეორე ფუნქციას 3 მთელირიცხვა პარამეტრი აქვს: სამკუთხედის გვერდები და აბრუნებს სამკუთხედის პერიმეტრს.

8.4.2. შეადგინეთ ფიგურის კლასი, რომელშიც განსაზღვრულია ერთი და იმავე სახელის მქონე 3 ფუნქცია. პირველ ფუნქციას ერთი მთელირიცხვა პარამეტრი აქვს და აბრუნებს კვადრატის პერიმეტრს. მეორე ფუნქციას ორი მთელირიცხვა პარამეტრი აქვს და აბრუნებს ოთხკუთხედის პერიმეტრს. მესამე ფუნქციას სამი მთელირიცხვა პარამეტრი აქვს და აბრუნებს სამკუთხედის

პერიმეტრს.

8.4.3. შექმენით ავტომობილის კლასი, რომელშიც განსაზღვრულია ერთი და იმავე სახელის მქონე 2 ფუნქცია. პირველ ფუნქციას 2 მთელი რიცხვა პარამეტრი აქვს: ავზის ტევადობა და მანძილი, რომელსაც გაივლის ავტომობილი 1 ლიტრი საწვავით, და აბრუნებს ავტომობილის მიერ სავსე ავზით გავლილ მანძილს. მეორე ფუნქციას 2 წილადი პარამეტრი აქვს: მაქსიმალური სიჩქარე და მოძრაობის დრო, და აბრუნებს მანძილს, რომელსაც გაივლის ავტომობილი მითითებული დროის განმავლობაში მაქსიმალური სიჩქარით მოძრაობისას.

8.4.4. შეადგინეთ მატარებლის კლასი, რომელშიც განსაზღვრულია ერთი და იმავე სახელის მქონე 2 ფუნქცია. პირველ ფუნქციას 2 მთელი რიცხვა პარამეტრი აქვს: ვაგონების რაოდენობა და ერთ ვაგონში მგზავრების რაოდენობა, და აბრუნებს მგზავრების საერთო რაოდენობას. მეორე ფუნქციას 2 წილადი პარამეტრი აქვს: 1 კილომეტრის გავლისას დახარჯული ელექტროენერგია და გავლილი მანძილი, და აბრუნებს მატარებლის მიერ მითითებული მანძილის გავლისას დახარჯულ ელექტროენერგიას.

კონსტრუქტორის გადატვირთვა

8.5.1. შექმენით კლასი, რომელიც მთელი ტიპის Min ცვლადს შეიცავს. კლასის I კონსტრუქტორი, რომელსაც გადაეცემა ერთგანზომილებიანი მთელი რიცხვა მასივი, Min ცვლადს ანიჭებს მასივის მინიმალურ ელემენტს. კლასის II კონსტრუქტორს პარამეტრად გადაეცემა ამავე კლასის ობიექტი, რომლის საშუალებით მოხდება ამავე კლასის Min ცვლადის ინიციალიზება.

8.5.2. შექმენით კლასი, რომელიც სამ გვერდს, პერიმეტრს, ფართობსა და სამ კონსტრუქტორს შეიცავს. I კონსტრუქტორს 1 მთელი ტიპის პარამეტრი აქვს და ქმნის კვადრატს (გამოთვლის პერიმეტრსა და ფართობს). II კონსტრუქტორს 2 მთელი ტიპის პარამეტრი აქვს და ქმნის მართკუთხედს. III კონსტრუქტორს 3 მთელი ტიპის პარამეტრი აქვს და ქმნის სამკუთხედს.

თავი 9. ფუნქციები უფრო დაწვრილებით

ჩასადგმელი ფუნქცია

ჩასადგმელი ფუნქციის სახით გააფორმეთ ფუნქცია, რომელიც გამოთვლის და აბრუნებს:

- 9.1.1. მართკუთხა სამკუთხედის ფართობს.
- 9.1.2. სამკუთხედის პერიმეტრს.
- 9.1.3. კვადრატის ფართობს.
- 9.1.4. კვადრატის პერიმეტრს.
- 9.1.5. მართკუთხედის ფართობს.
- 9.1.6. მართკუთხედის პერიმეტრს.
- 9.1.7. წრეწირის ფართობს.
- 9.1.8. წრეწირის სიგრძეს.
- 9.1.9. ტრაპეციის ფართობს.
- 9.1.10. ტრაპეციის პერიმეტრს.
- 9.1.11. გასცემს ორ რიცხვს შორის მაქსიმალურს.
- 9.1.12. გასცემს ორ რიცხვს შორის მინიმალურს.
- 9.1.13. დაადგენს ორ რიცხვს შორის არის თუ არა უარყოფითი და გასცემს შესაბამის შეტყობინებას.
- 9.1.14. დაადგენს ორ რიცხვს შორის არის თუ არა კენტი და გასცემს შესაბამის შეტყობინებას.
- 9.1.15. დაადგენს ორ რიცხვს შორის არის თუ არა 5-ის ჯერადი და გასცემს შესაბამის შეტყობინებას.
- 9.1.16. დაადგენს x რიცხვი არის თუ არა $0 < x \leq 17$ დიაპაზონში მოთავსებული და გასცემს შესაბამის შეტყობინებას.

9.1.17. დაადგენს x რიცხვი არის თუ არა 25-ზე ნაკლები ან 100-ზე მეტი და გასცემს შესაბამის შეტყობინებას.

9.1.18. დათვლის ლუწი რიცხვების რაოდენობას 5-დან 27-მდე დიაპაზონში.

9.1.19. შეკრებს კენტ რიცხვებს 10-დან 20-მდე დიაპაზონში.

9.1.20. შეკრებს 1-დან 20-მდე რიცხვებს მანამ, სანამ ჯამი არ გახდება 24-ზე მეტი.

9.1.21. შეკრებს 9-ის ჯერად რიცხვებს 2-დან 47-მდე დიაპაზონში მანამ, სანამ 9-ის ჯერადი რიცხვი იქნება 30-ზე ნაკლები.

ფუნქციის შაბლონი

9.3.1. ერთი ერთგანზომილებიანი მასივიდან მეორე ერთგანზომილებიან მასივში გადაწერეთ დადებითი რიცხვები. შეადგინეთ ფუნქციის შაბლონი, რომელიც იმუშავებს მთელი და წილადი რიცხვებისთვის.

9.3.2. ორგანზომილებიანი მასივიდან ლუწი რიცხვები გადაწერეთ ერთგანზომილებიან მასივში. შეადგინეთ ფუნქციის შაბლონი, რომელიც იმუშავებს მთელი და წილადი რიცხვებისთვის.

9.3.3. 4.1.1-4.2.81 ამოცანებისთვის შეადგინეთ ფუნქციის შაბლონი, რომელიც იმუშავებს მთელი და წილადი რიცხვებისთვის.

თავი 10. სტრუქტურები

სტრუქტურა

10.1.1. შეადგინეთ სტუდენტის სტრუქტურა, რომელიც შემდეგ ველებს მოიცავს: სტუდენტის სახელი, სტუდენტის გვარი, კურსი, გადახდილი თანხა. ძირითად პროგრამაში შექმენით ობიექტი. ამ ველებს მნიშვნელობები მიანიჭეთ.

10.1.2. შეადგინეთ მოსწავლის სტრუქტურა, რომელიც შემდეგ ველებს მოიცავს: მოსწავლის სახელი, მოსწავლის გვარი, კლასი, სკოლის ნომერი. ძირითად პროგრამაში შექმენით ობიექტი. ამ ველებს მნიშვნელობები მიანიჭეთ.

10.1.3. შეადგინეთ მატარებლის სტრუქტურა, რომელიც შემდეგ ველებს მოიცავს: ვაგონების რაოდენობა, ვაგონში მგზავრების რაოდენობა, ბილეთის ფასი, დანიშნულების სადგური. ძირითად პროგრამაში შექმენით ობიექტი. ამ ველებს მნიშვნელობები მიანიჭეთ.

10.1.4. შეადგინეთ თვითმფრინავის სტრუქტურა, რომელიც შემდეგ ველებს მოიცავს: მგზავრების რაოდენობა, ბილეთის ფასი, დანიშნულების აეროპორტი. ძირითად პროგრამაში შექმენით ობიექტი. ამ ველებს მნიშვნელობები მიანიჭეთ.

10.1.5. შეადგინეთ კომპიუტერის სტრუქტურა, რომელიც შემდეგ ველებს მოიცავს: ფასი, პროცესორის ტიპი, პროცესორის სწრაფქმედება, ოპერატიული მეხსიერების ზომა, მყარი დისკის ზომა. ძირითად პროგრამაში შექმენით ობიექტი. ამ ველებს მნიშვნელობები მიანიჭეთ.

10.1.6. შეადგინეთ უნივერსიტეტის სტრუქტურა, რომელიც შემდეგ ველებს მოიცავს: სტუდენტების რაოდენობა, მისამართი, დასახელება. ძირითად პროგრამაში შექმენით ობიექტი. ამ ველებს მნიშვნელობები მიანიჭეთ.

10.1.7. შეადგინეთ ფაკულტეტის სტრუქტურა, რომელიც შემდეგ ველებს მოიცავს: სტუდენტების რაოდენობა, სწავლის ფასი, დასახელება. ძირითად პროგრამაში შექმენით ობიექტი. ამ ველებს მნიშვნელობები მიანიჭეთ.

10.1.8. შეადგინეთ ტელევიზორის სტრუქტურა, რომელიც შემდეგ ველებს მოიცავს: მოხმარებული ენერჯია, ეკრანის დიაგონალის ზომა, გამომშვები ფორმა, ფასი. ძირითად პროგრამაში შექმენით ობიექტი. ამ ველებს მნიშვნელობები მიანიჭეთ.

10.1.9. შეადგინეთ კვადრატის სტრუქტურა, რომელიც შემდეგ ველებს მოიცავს: გვერდის ზომა, დიაგონალის ზომა, ფართობი, პერიმეტრი. ძირითად პროგრამაში შექმენით ობიექტი. ამ ველებს მნიშვნელობები მიანიჭეთ.

10.1.10. შეადგინეთ მართკუთხედის სტრუქტურა, რომელიც შემდეგ ველებს მოიცავს: სიგრძე, სიგანე, ფართობი, პერიმეტრი. ძირითად პროგრამაში შექმენით ობიექტი. ამ ველებს მნიშვნელობები მიანიჭეთ.

10.1.11. შეადგინეთ სამკუთხედის სტრუქტურა, რომელიც შემდეგ ველებს მოიცავს: სამივე გვერდის ზომა, ფართობი, პერიმეტრი. ძირითად პროგრამაში შექმენით ობიექტი. ამ ველებს მნიშვნელობები მიანიჭეთ.

ჩამოთვლა

10.2.1. შეადგინეთ ჩამოთვლადი ტიპი თვეებისთვის.

10.2.2. შეადგინეთ ჩამოთვლადი ტიპი კვირის დღეებისთვის.

10.2.3. შეადგინეთ ჩამოთვლადი ტიპი ქალაქების დასახელებებისთვის.

10.2.4. შეადგინეთ ჩამოთვლადი ტიპი ქვეყნების დასახელებებისთვის.

10.2.5. შეადგინეთ ჩამოთვლადი ტიპი ხილის დასახელებებისთვის.

10.2.6. შეადგინეთ ჩამოთვლადი ტიპი ბოსტნეულის დასახელებებისთვის.

10.2.7. შეადგინეთ ჩამოთვლადი ტიპი მდინარეების დასახელებებისთვის.

10.2.8. შეადგინეთ ჩამოთვლადი ტიპი გეომეტრიული ფიგურების დასახელებებისთვის.

10.2.9. შეადგინეთ ჩამოთვლადი ტიპი ზღვების დასახელებებისთვის.

სხვადასხვა

კომბინატორიკა

11.1.1. შეადგინეთ პროგრამა, რომელიც დააფორმირებს ხუთკაციანი გუნდის ყველა შესაძლო ვარიანტს არსებული ცხრა კანდიდატურისაგან.

11.1.2. შეადგინეთ პროგრამა, რომელიც მოძებნის ისეთ ოთხნიშნა რიცხვებს, რომელთა ციფრების ჯამი წინასწარ მოცემული N რიცხვის ტოლია.

11.1.3. შეადგინეთ პროგრამა, რომელიც დააფორმირებს 1,2,3,4 ციფრების ყველა შესაძლო გადაადგილებას.

11.1.4. შეადგინეთ პროგრამა, რომელიც 1,2,3,4,5,6,7 ციფრებისაგან დააფორმირებს ყველა შესაძლო ოთხთანრიგა რიცხვს.

დახარისხება

11.2.1. შეადგინეთ პროგრამა, რომელიც ერთგანზომილებიანი მასივის ელემენტებს ზრდადობის მიხედვით დაალაგებს.

11.2.2. შეადგინეთ პროგრამა, რომელიც ერთგანზომილებიანი მასივის ელემენტებს კლებადობის მიხედვით დაალაგებს.

11.2.3. შეადგინეთ პროგრამა, რომელიც ორგანზომილებიანი მასივის თითოეული სტრიქონის ელემენტებს კლებადობის მიხედვით დაალაგებს.

11.2.4. შეადგინეთ პროგრამა, რომელიც ორგანზომილებიანი მასივის თითოეული სვეტის ელემენტებს ზრდადობის მიხედვით დაალაგებს.

11.2.5. შეადგინეთ პროგრამა, რომელიც შეამოწმებს ერთგანზომილებიანი მასივი არის თუ არა ზრდადობის მიხედვით დალაგებული.

11.2.6. შეადგინეთ პროგრამა, რომელიც შეამოწმებს ერთგანზომილებიანი მასივი არის თუ არა კლებადობის მიხედვით დალაგებული.

11.2.7. მასივი დალაგებულია ზრდადობის მიხედვით. შეადგინეთ პროგრამა, რომელიც ამ მასივში ჩასვამს R რიცხვს მასივის ზრდადობის მიხედვით მოწესრიგების დაურღვევლად.

11.2.8. მასივი დალაგებულია კლებადობის მიხედვით. შეადგინეთ პროგრამა, რომელიც ამ მასივში ჩასვამს R რიცხვს მასივის კლებადობის მიხედვით მოწესრიგების დაურღვევლად.

11.2.9. მოცემულია ზრდადობის მიხედვით დალაგებული ორი მასივი. შეადგინეთ პროგრამა, რომელიც ამ ორი მასივისაგან შექმნის მესამე მასივს, რომლის ელემენტებიც, ასევე, ზრდადობის მიხედვით იქნება დალაგებული.

11.2.10. შეადგინეთ პროგრამა, რომელიც სტუდენტების გვარებს ანბანის მიხედვით დაალაგებს.

დანართი 2. სავარჯიშოების ამოხსნები

თავი 2. C++ ენის საფუძვლები

არიტმეტიკული გამოსახულება

2.1.1.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int n;
    double y;

    cin >> n;
    y = n * 0.6 + 4.25;
    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}
```

2.1.3.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int m;
    double y;

    cin >> m;
    y = -8 * m * m * m + 4 * m * m - 5 * m + 6;
    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}
```

2.1.4.

```
#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int a, b, y;
    double z;
```

```

    cin >> a >> b >> y;
    z = 1.25 * a * y * y + 0.8 * ( b - y );
    cout << "z = " << z << endl;
    //
    system("pause");
    return 0;
}

```

2.1.7.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    double y, a, b, c, d;

    cin >> a >> b >> c >> d;
    y = a * b / ( b + c ) - ( a + d ) / ( a + b );
    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}

```

2.1.9.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    double y, k, m;

    cin >> k >> m;
    y = k * m / ( 5 * m - 2 * k );
    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}

```

2.1.13.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    double y, x, z;

```

```

cin >> x >> z;
y = ( x * x + z * z ) / ( 1 - ( x * x - z * z ) / 2 );
cout << "y = " << y << endl;
//
system("pause");
return 0;
}

```

ინკრემენტისა და დეკრემენტის ოპერატორები

2.3.1.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int y, a, b, c;

    a = 5;
    b = 7;
    c = 12;
    y = a + --b + c++;
    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}

```

2.3.9.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int y, a, b, c;

    a = 5;
    b = 7;
    c = 12;
    y = a-- * ++b - c--;
    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}

```

2.3.15.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    int y, a, b, c;

    a = 5;
    b = 7;
    c = 12;
    y = ( ++a - b-- ) / c;
    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}

```

2.3.16.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    int a, b, c;
    double y;

    a = 5;
    b = 7;
    c = 12;
    y = ( a-- - b++ + c-- ) / ( --a - ++b - ++c );
    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}

```

შედარებისა და ლოგიკის ოპერატორები

2.4.3.

```

int _tmain(int argc, _TCHAR* argv[])
{
    bool y, x1, x2, x3, x4;

    x1 = true;
    x2 = true;
    x3 = false;
    x4 = false;
    y = !x1 && !x2 && x3 || x4;
    cout << "y = " << y << endl;

    system("pause");
    return 0;
}

```

```
}
```

2.5.3.

```
#include "stdafx.h"  
#include "iostream"  
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])  
{  
    bool y;  
    int x;  
  
    cin >> x;  
    y = x == 30;  
    cout << "y = " << y << endl;  
    //  
    system("pause");  
    return 0;  
}
```

2.5.7.

```
#include "stdafx.h"  
#include "iostream"  
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])  
{  
    bool y;  
    int x;  
  
    cin >> x;  
    y = x > 10 && x <= 20;  
    cout << "y = " << y << endl;  
    //  
    system("pause");  
    return 0;  
}
```

2.5.11.

```
#include "stdafx.h"  
#include "iostream"  
using namespace std;
```

```
int _tmain(int argc, _TCHAR* argv[])  
{  
    bool y;  
    int x;  
  
    cin >> x;  
    y = x < 5 || x > 20;  
    cout << "y = " << y << endl;  
    //
```

```

    system("pause");
    return 0;
}

```

მათემატიკის ფუნქციები

2.6.1.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    double x, y;

    cin >> x;
    y = sqrt(1 + sqrt(abs(x)));
    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}

```

2.6.2.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    double x, y;

    cin >> x;
    y = ( 1 + x ) * ( x - sqrt(x - 1) / 4 ) / ( exp(x) + 1 / ( pow(x,2) + 4 ) );
    cout << "y = " << y << endl;

    system("pause");
    return 0;
}

```

2.6.3.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    double x, y;

    cin >> x;
    y = sin(pow(x,3)) + pow(cos(pow(x,4)),3) - exp(sqrt(x));
}

```

```

    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}

```

2.6.6.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    int a, b;
    double y;

    cin >> a >> b;
    y = sqrt(pow(a,3) + b) / ( a + pow(b,3));
    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}

```

2.6.11.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    double x, y;

    cin >> x;
    y = pow(x,3) - ( exp(pow(-x,2)) + cos(x)) / (3.7 + pow(x,2) / (5.8 - sqrt(x+1)));
    cout << "y = " << y << endl;
    //
    system("pause");
    return 0;
}

```

თავი 3. მმართველი ოპერატორი if ოპერატორი

3.1.3.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi;

    cin >> ricxvi;
    if ( ricxvi % 5 == 0 ) cout << "ricxvi 5-is jeradia" << endl;
}

```



```

        else cout << "ricxvi 5-is jeradi ar aris" << endl;

    system("pause");
    return 0;
}

```

3.1.9.

```

    int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi1, ricxvi2;

    cin >> ricxvi1 >> ricxvi2;
    if ( ricxvi1 >= ricxvi2 ) cout << "pirveli ricxvi meoreze metia an misi toli" << endl;
        else cout << "meore ricxvi pirvelze metia" << endl;

    system("pause");
    return 0;
}

```

3.1.11.

```

    int _tmain(int argc, _TCHAR* argv[])
{
    int raodenoba = 0;
    int ricxvi1, ricxvi2, ricxvi3;

    cin >> ricxvi1 >> ricxvi2 >> ricxvi3;

    if ( ricxvi1 % 2 == 1 ) raodenoba++;
    if ( ricxvi2 % 2 == 1 ) raodenoba++;
    if ( ricxvi3 % 2 == 1 ) raodenoba++;
    cout << raodenoba << endl;

    system("pause");
    return 0;
}

```

ჩადგმული if ოპერატორი

3.2.1.

```

    int _tmain(int argc, _TCHAR* argv[])
{
    int int ricxvi1, ricxvi2, ricxvi3;

    cin >> ricxvi1 >> ricxvi2 >> ricxvi3;
    if ( ricxvi1 >= ricxvi2 )
        if ( ricxvi1 >= ricxvi3 ) cout << "pirveli ricxvi udidesia" << endl;
            else cout << "mesame ricxvi udidesia" << endl;
        else
            if ( ricxvi2 >= ricxvi3 ) cout << "meore ricxvi udidesia" << endl;
                else cout << "mesame ricxvi udidesia " << endl;
}

```

```
system("pause");
return 0;
```

```
}
```

3.2.8.

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
int ricxvi1, ricxvi2, ricxvi3;
```

```
cin >> ricxvi1 >> ricxvi2 >> ricxvi3;
```

```
if ( ricxvi1 < 0 ) cout << "sam ricxvs shoris aris uaryofiti" << endl;
```

```
else if ( ricxvi2 < 0 ) cout << "sam ricxvs shoris aris uaryofiti" << endl;
```

```
else if ( ricxvi3 < 0 ) cout << "sam ricxvs shoris aris uaryofiti" << endl;
```

```
else cout << "sam ricxvs shoris ar aris uaryofiti" << endl;
```

```
system("pause");
```

```
return 0;
```

```
}
```

ეს ამოცანა შეიძლება გადაწყდეს ლოგიკის გამოყენებით:

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
int ricxvi1, ricxvi2, ricxvi3;
```

```
cin >> ricxvi1 >> ricxvi2 >> ricxvi3;
```

```
if ( ( ricxvi1 < 0 ) || ( ricxvi2 < 0 ) || ( ricxvi3 < 0 ) )
```

```
cout << "sam ricxvs shoris aris uaryofiti" << endl;
```

```
else cout << "sam ricxvs shoris ar aris uaryofiti" << endl;
```

```
system("pause");
```

```
return 0;
```

```
}
```

ლოგიკა

3.3.2.

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
int ricxvi;
```

```
cin >> ricxvi;
```

```
if ( ( ricxvi >= 10 ) && ( ricxvi < 19 ) )
```

```
cout << "ricxvi motavsebulia mititebul diapazonshi" << endl;
```

```
else cout << "ricxvi motavsebulia ar aris mititebul diapazonshi" << endl;
```

```
system("pause");
```

```
return 0;
```

```
}
```

3.3.3.

```
int _tmain(int argc, _TCHAR* argv[])
```

```
{
```

```
int ricxvi;
```

```

cin >> ricxvi;
if ( ( ricxvi < 25 ) || ( ricxvi > 100 ) )
    cout << "ricxvi motavsebulia mititebul diapazonshi" << endl;
    else cout << "ricxvi motavsebulia ar aris mititebul diapazonshi" << endl;

system("pause");
return 0;
}

```

switch ოპერატორი

3.4.1.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int tvis_nomeri;

    cin >> tvis_nomeri;
    switch (tvis_nomeri)
    {
        case 1: cout << "ianvari" << endl; break;
        case 2: cout << "tebervali" << endl; break;
        case 3: cout << "marti" << endl; break;
        case 4: cout << "aprili" << endl; break;
        case 5: cout << "maisi" << endl; break;
        case 6: cout << "ivnisi" << endl; break;
        case 7: cout << "ivlisi" << endl; break;
        case 8: cout << "agvisto" << endl; break;
        case 9: cout << "seqtemberi" << endl; break;
        case 10: cout << "oqtomberi" << endl; break;
        case 11: cout << "noemberi" << endl; break;
        case 12: cout << "dekemberi" << endl; break;
        default: cout << "araswori nomeri" << endl;
    }
    //
    system("pause");
    return 0;
}

```

3.4.5.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    char simbolo_cifri;

```

```

cin >> simbolo_cifri;
switch (simbolo_cifri)
{
case '0':
case '2':
case '4':
case '6':
case '8': cout << "cifri luwia" << endl; break;
case '1':
case '3':
case '5':
case '7':
case '9': cout << "cifri kenia" << endl; break;
default : cout << "araswori cifri" << endl;
}
//
system("pause");
return 0;
}

```

for, while, do-while ოპერატორები

3.5.1.

```

int _tmain(int argc, _TCHAR* argv[])
{
double jami = 0;
int mnishvneli, minusi = -1;

for (mnishvneli = 1; mnishvneli <= 5; mnishvneli++)
{
    minusi *= -1;
    jami += minusi * ( 1.0 / mnishvneli);
}
cout << jami << endl;

system("pause");
return 0;
}

```

3.5.13.

```

int _tmain(int argc, _TCHAR* argv[])
{
double ricxvi, jami = 0;
int mnishvneli;

cin >> ricxvi;
for ( mnishvneli = 1; mnishvneli <= 7; mnishvneli += 2 )
    jami += pow(ricxvi, mnishvneli) / mnishvneli;
jami *= 2;
}

```

```

cout << jami << endl;

system("pause");
return 0;
}

```

3.5.16.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int jami = 0;

    for ( int ricxvi = 10; ricxvi <= 20; ricxvi++ )
        if ( ricxvi % 2 == 1 ) jami += ricxvi;
    cout << jami << endl;
}

```

ამ ამოცანის გადაწყვეტის მეორე გზა

```

int _tmain(int argc, _TCHAR* argv[])
{
    int jami = 0;
    for ( int ricxvi = 11; ricxvi <= 20; ricxvi += 2 )
        jami += ricxvi;
    cout << jami << endl;

    system("pause");
    return 0;
}

```

continue, break ოპერატორები

3.6.1.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int jami = 0;

    for ( int ricxvi = 1; ricxvi <= 20; ricxvi++ )
        if ( jami > 24 ) break;
        else jami += ricxvi;
    cout << jami << endl;

    system("pause");
    return 0;
}

```

3.6.2.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int raodenoba = 0;

    for ( int ricxvi = 6; ricxvi <= 79; ricxvi++ )
    {

```

```

    if ( ricxvi % 4 == 0 ) raodenoba++;
    if ( ricxvi > 50 ) break;
}
cout << raodenoba<< endl;

system("pause");
return 0;
}

```

თავი 4. მასივი, სტრიქონი და ბიტობრივი ოპერაცია ერთგანზომილებიანი მასივი

4.1.5.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[5] = { 1, -5, 20, -2, 4 };
    int max = masivi[0], maxind = 0;

    for (int indexi = 1; indexi < 5; indexi++)
        if (max < masivi[indexi])
        {
            max = masivi[indexi];
            maxind = indexi;
        }
    cout << "masivis maximaluri elementi = " << max
         << "\nmaximaluri elementis indexi = " << maxind << endl;
    for (int indexi = 0; indexi < 5; indexi++)
        cout << masivi[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}

```

4.1.8.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[5] = { 1, -5, 20, -2, 4 };
    int jami = 0;

    for ( int indexi = 0; indexi < 5; indexi++ )
        if ( masivi[indexi] >= 0 ) jami += masivi[indexi];
    cout << "dadebiri elementebis jami = " << jami << endl;
    for ( int indexi = 0; indexi < 5; indexi++ )
        cout << masivi[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}

```

4.1.9.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[5] = { 1, -5, 20, -2, 4 };
    int raodenoba = 0;

    for ( int indexi = 0; indexi < 5; indexi++ )
        if ( masivi[indexi] < 0 ) raodenoba++;
    cout << "uaryofoto elementebis raodenoba = " << raodenoba<< endl;
    for ( int indexi = 0; indexi < 5; indexi++ )
        cout << masivi[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}
```

4.1.12.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[5] = { 1, -5, 20, -2, 4 };
    int jami = 0;

    for ( int indexi = 0; indexi < 5; indexi += 2 )
        if ( masivi[indexi] >= 0 ) jami += masivi[indexi];
    cout << "luwi indexis mqone elementebis jami = " << jami << endl;
    for ( int indexi = 0; indexi < 5; indexi++ )
        cout << masivi[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}
```

4.1.15.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[10] = { 1 , 5 , 20 , -2 , 4 , 75 , -3 , 85 , 9 , 21 };
    int indexi;

    for ( indexi = 0 ; indexi < 10; indexi++ )
        if ( masivi[indexi] < 0 ) break;
        else continue;
    cout << "pirveli uaryofiti elementi = " << masivi[indexi] << "   misi indexi = " << indexi << endl;
    for ( indexi = 0 ; indexi < 10; indexi++ )
        cout << masivi[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}
```

```
}
```

4.1.17.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[10] = { 1, 5, -20, -2, 4, 75, -3, -85, 9, -21 };
    int indexi, max = masivi[0], maxind = 0;

    for ( indexi = 0 ; indexi < 10; indexi++ )
        if ( masivi[indexi] < 0 ) break;
    max = masivi[indexi];
    maxind = indexi;
    for ( ; indexi < 10; indexi++ )
        if ( ( masivi[indexi] < 0 ) && ( max < masivi[indexi] ) )
        {
            max = masivi[indexi];
            maxind = indexi;
        }
    cout << "maqsimaluri uaryofiti ricxvi = " <<
        max << endl << "misi indexi = " << maxind << endl;
    for ( indexi = 0 ; indexi < 10; indexi++ )
        cout << masivi[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}
```

4.1.22.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[10] = { 1, 5, -20, -2, 4, 75, -3, -85, 9, -21 };
    int indexi, min, max, raodenoba = 0;

    cin >> min >> max;
    for ( indexi = 0 ; indexi < 10; indexi++ )
        if ( ( min <= masivi[indexi] ) && ( max >= masivi[indexi] ) ) raodenoba++;
    cout << "raodenoba = " << raodenoba << endl;
    for ( indexi = 0 ; indexi < 10; indexi++ )
        cout << masivi[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}
```

4.1.25.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[10] = { 3, 9, 6, -1, 12, 10, 17, 4, 6, 19 };
    int ind, min_ind, max_ind, jami = 0;
    // საწყისი ელემენტის ინდექსის შეტანა
```



```

cin >> min_ind;
//      საბოლოო ელემენტის ინდექსის შეტანა
cin >> max_ind;

for (ind = min_ind; ind <= max_ind; ind++)
    jami += masivi [ind];
cout << "jami = " << jami << endl;

    system("pause");
    return 0;
}

```

4.1.26.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi[10] = { 3, 9, 6, -1, 12, 10, 17, 4, 6, 19 };
int ind, max , min_ind, max_ind, jami = 0;
//      საწყისი ელემენტის ინდექსის შეტანა
cin >> min_ind;
//      საბოლოო ელემენტის ინდექსის შეტანა
cin >> max_ind;
max = masivi[min_ind];
for (ind = min_ind; ind <= max_ind; ind++)
    if ( max < masivi[ind] ) max = masivi[ind];
cout << "max = " << max << endl;

    system("pause");
    return 0;
}

```

4.1.28.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi1[10] = { 3, 29, 6, -1, 12, 40, 17, 64, 6, 79 };
    int masivi2[10];
    int ind1, ind2 = 0, ricxvi;
    cin >> ricxvi;
    for (ind1 = 0; ind1 < 10; ind1++)
        if (masivi1[ind1] < ricxvi) masivi2[ind2++] = masivi1[ind1];
    for (ind1 = 0; ind1 < ind2; ind1++)
        cout << masivi2[ind1] << " ";
    cout << endl;

    system("pause");
    return 0;
}

```

4.1.31.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi1[10] = { 1, -5, 20, -2, 4, 75, -3, 85, 9, 21 };
    int masivi2[10];

```

```

int index1, index2 = 0;

for (index1 = 0; index1 < 10; index1++)
if (masivi1[index1] % 5 == 0) masivi2[index2++] = masivi1[index1];

cout << "sawyisi masivi: ";
for (index1 = 0; index1 < 10; index1++)
    cout << masivi1[index1] << " ";
cout << endl;
cout << "shedegi: ";
for (index1 = 0; index1 < index2; index1++)
    cout << masivi2[index1] << " ";
cout << endl;

system("pause");
return 0;
}

```

4.1.36.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi1[10] = { 1, -5, 20, -2, 4, 75, -3, 85, 9, 21 };
int masivi2[10];
int index1, index2 = 0;

for ( index1 = 0 ; index1 < 10; index1++ )
    if ( masivi1[index1] >= 0 ) masivi2[index2++] = index1;

for ( index1 = 0 ; index1 < 10; index1++ )
    cout << masivi1[index1] << " ";
cout << endl;
for ( index1 = 0 ; index1 < index2 ; index1++ )
    cout << masivi2[index1] << " ";
cout << endl;

system("pause");
return 0;
}

```

4.1.45.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi[10] = { 1, -5, 20, -2, 4, 75, -3, 85, 9, 21 };
int indexi, temp;

for ( indexi = 0 ; indexi < 10; indexi++ )
    cout << masivi[indexi] << " ";
cout << endl;
for ( indexi = 0 ; indexi < 10 / 2 ; indexi++ )
{
    temp = masivi[indexi];

```

```

    masivi[indexi] = masivi[9 - indexi];
    masivi[9 - indexi] = temp;
}
for ( indexi = 0 ; indexi < 10; indexi++ )
    cout << masivi[indexi] << " ";
cout << endl;

system("pause");
return 0;
}

```

4.1.48.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[10] = { 1, -5, 0, -2, 0, 0, -3, 0, 9, 0 };
    int indexi, temp, Alami;

    for ( indexi = 0 ; indexi < 10; indexi++ )
        cout << masivi[indexi] << " ";
    cout << endl;
M1: Alami = 0;
    for ( indexi = 0 ; indexi < 9; indexi++ )
        if ( ( masivi[indexi] == 0 ) && ( masivi[indexi + 1] != 0 ) )
        {
            temp = masivi[indexi];
            masivi[indexi] = masivi[indexi + 1];
            masivi[indexi + 1] = temp;
            Alami = 1;
        }
    if ( Alami == 1 ) goto M1;
    for ( indexi = 0 ; indexi < 10; indexi++ )
        cout << masivi[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}

```

4.1.49.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[5] = { 1, 5, 2, 4, 7 };
    int indexi, temp, Zvris_Raodenoba;
    cin >> Zvris_Raodenoba;

    cout << "sawyisi masivi: ";
    for (indexi = 0; indexi < 5; indexi++)
        cout << masivi[indexi] << " ";
    cout << endl;
    for (int i = 1; i <= Zvris_Raodenoba; i++)
    {

```

```

    temp = masivi[4];
    for (indexi = 0; indexi < 4; indexi++)
        masivi[4 - indexi] = masivi[3 - indexi];
    masivi[0] = temp;
}
cout << "shedegi: ";
for (indexi = 0; indexi < 5; indexi++)
    cout << masivi[indexi] << " ";
cout << endl;

system("pause");
return 0;
}

```

4.1.50.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[] = { 1, 5, 2, 4, 7 };
    int indexi, temp, Zvris_Raodenoba ;

    cin >> Zvris_Raodenoba;
    for ( indexi = 0 ; indexi < 5; indexi++ )
        cout << masivi[indexi]<< " ";
    cout << endl;
    for ( int i = 1 ; i <= Zvris_Raodenoba ; i++ )
    {
        temp = masivi[0];
        for ( indexi = 0 ; indexi < 4; indexi++ )
            masivi[indexi] = masivi[indexi + 1];
        masivi[4] = temp;
    }
    for ( indexi = 0 ; indexi < 5; indexi++ )
        cout << masivi[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}

```

ორგანზომილებიანი მასივი

4.2.2.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[3][3] = { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
    int striqoni, sveti, minstriqoni = 0, minsveti = 0, min = masivi[0][0];

    for ( striqoni = 0; striqoni < 3; striqoni++ )
        for ( sveti = 0; sveti < 3; sveti++ )
            if ( min > masivi[striqoni][sveti] )
            {

```

```

        min = masivi[striqoni][sveti];
        minstriqoni = striqoni;
        minsveti = sveti;
    }
    cout << "minimaluri elementi = " << min << endl
        << "striqonis indexi = " << minstriqoni << endl
        << "svetis indexi = " << minsveti << endl;
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            cout << masivi[striqoni][sveti] << " ";
        cout << endl;
    }

    system("pause");
    return 0;
}

```

4.2.7.

```

    int _tmain(int argc, _TCHAR* argv[])
    {
        int masivi[3][3] = { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
        int striqoni, sveti, jami = 0;

        for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
            for ( sveti = 0 ; sveti < 3 ; sveti++ )
                if ( masivi[striqoni][sveti] % 2 == 1 )
                    jami += masivi[striqoni][sveti];
        cout << "kenti elementebis jami = " << jami << endl;
        for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        {
            for ( sveti = 0 ; sveti < 3 ; sveti++ )
                cout << masivi[striqoni][sveti] << " ";
            cout << endl;
        }

        system("pause");
        return 0;
    }

```

4.2.8.

```

    int _tmain(int argc, _TCHAR* argv[])
    {
        int masivi[3][3] = { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
        int striqoni, sveti, raodenoba = 0;

        for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
            for ( sveti = 0 ; sveti < 3 ; sveti++ )
                if ( masivi[striqoni][sveti] % 2 == 0 ) raodenoba++;
        cout << "luwi elementebis raodenoba = " << raodenoba << endl;
        for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )

```

```

{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        cout << masivi[striqoni][sveti] << " ";
    cout << endl;
}

```

```

system("pause");
return 0;
}

```

4.2.11.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[3][3] = { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
    int striqoni, sveti, jami = 0;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        jami += masivi[striqoni][striqoni];
    cout << "jami = " << jami << endl;
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            cout << masivi[striqoni][sveti] << " ";
        cout << endl;
    }

    system("pause");
    return 0;
}

```

4.2.12.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[3][3] = { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
    int striqoni, sveti, namravli = 1;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        namravli *= masivi[striqoni, 2 - striqoni];
    cout << namravli << endl;
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            cout << masivi[striqoni][sveti] << " ";
        cout << endl;
    }

    system("pause");
    return 0;
}

```

4.2.20.

```

int _tmain(int argc, _TCHAR* argv[])

```

```

{
int masivi1[3][3] = { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
int masivi2[9];
int indexi2 = 0, striqoni, sveti;

for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        if ( masivi1[striqoni][sveti] % 2 == 0 ) masivi2[indexi2++] = masivi1[striqoni][sveti];
cout << "sawyisi masivi:" << endl;
for (striqoni = 0; striqoni < 3; striqoni++)
{
    for (sveti = 0; sveti < 3; sveti++)
        cout << masivi1[striqoni][sveti] << " ";
    cout << endl;
}
cout << "shedegi:" << endl;
for ( striqoni = 0 ; striqoni < indexi2 ; striqoni++ )
    cout << masivi2[striqoni] << " ";
cout << endl;

system("pause");
return 0;
}

```

4.2.22.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi1[3][3] = { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
int masivi2[3] = {0, 0, 0};
int striqoni, sveti;

for (sveti = 0; sveti < 3; sveti++)
for (striqoni = 0; striqoni < 3; striqoni++)
    masivi2[sveti] += masivi1[striqoni][sveti];
cout << "sawyisi masivi:" << endl;
for (striqoni = 0; striqoni < 3; striqoni++)
{
    for (sveti = 0; sveti < 3; sveti++)
        cout << masivi1[striqoni][sveti] << " ";
    cout << endl;
}
cout << "shedegi:" << endl;
for (sveti = 0; sveti < 3; sveti++)
    cout << masivi2[sveti] << " ";
cout << endl;

```

```

system("pause");
return 0;

```

4.2.23.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi1[3][3] = { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
int masivi2[3] = { 1, 1, 1 };
int striqoni, sveti;

for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        masivi2[striqoni] *= masivi1[striqoni][sveti];
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        cout << masivi1[striqoni][sveti] << " ";
    cout << endl;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    cout << masivi2[striqoni] << " ";
cout << endl;

system("pause");
return 0;
}

```

4.2.25.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi1[3][3] = { { 1, -2, 3 }, { 4, 5, -6 }, { 7, 8, 9 } };
int masivi2[3] = { 0, 0, 0 };
int striqoni, sveti;

for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        if ( masivi1[striqoni][sveti] %2 == 1 ) masivi2[striqoni] += masivi1[striqoni][sveti];
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        cout << masivi1[striqoni][sveti] << " ";
    cout << endl;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    cout << masivi2[striqoni] << " ";
cout << endl;

system("pause");
return 0;
}

```

4.2.26.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi1[3][3] = { { -1, 4, 13 }, { 7, 5, -6 }, { 4, 8, 9 } };

```



```

int masivi2[3] = { 0, 0, 0 };
int striqoni, sveti, max;

for ( sveti = 0 ; sveti < 3 ; sveti++ )
{
    max = masivi1[0][sveti];
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        if ( max < masivi1[striqoni][sveti] ) max = masivi1[striqoni][sveti];
    masivi2[sveti] = max;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        cout << masivi1[striqoni][sveti] << " ";
    cout << endl;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    cout << masivi2[striqoni] << " ";
cout << endl;

system("pause");
return 0;
}

```

4.2.28.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi1[3][3] = { { 11, -2, 3 }, { 4, 15, -6 }, { -7, 8, 9 } };
int masivi2[3] = { 0, 0, 0 };
int striqoni, sveti, max = masivi1[0][0], maxstriqoni = 0;

for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    if ( max < masivi1[striqoni][striqoni] )
    {
        max = masivi1[striqoni][striqoni];
        maxstriqoni = striqoni;
    }
for ( sveti = 0 ; sveti < 3 ; sveti++ )
    masivi2[sveti] = masivi1[maxstriqoni][sveti];
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        cout << masivi1[striqoni][sveti] << " ";
    cout << endl;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    cout << masivi2[striqoni] << " ";
cout << endl;

system("pause");
}

```

```
    return 0;
```

```
}
```

4.2.34.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int masivi1[3][3] = { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
    int masivi2[3] = { 0, 0, 0 };
    int striqoni, sveti;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        masivi2[striqoni] = masivi1[striqoni][striqoni];
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            cout << masivi1[striqoni][sveti] << " ";
        cout << endl;
    }
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        cout << masivi2[striqoni] << " ";
    cout << endl;

    system("pause");
    return 0;
}
```

4.2.35.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int masivi1[3][3] = { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
    int masivi2[3] = { 0, 0, 0 };
    int striqoni, sveti;

    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        masivi2[striqoni] = masivi1[striqoni][2 - striqoni];
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            cout << masivi1[striqoni][sveti] << " ";
        cout << endl;
    }
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        cout << masivi2[striqoni] << " ";
    cout << endl;

    system("pause");
    return 0;
}
```

4.2.36.

```
int _tmain(int argc, _TCHAR* argv[])
{
```

```

int masivi1[3][3] = { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
double masivi2[3] = { 0, 0, 0 };
int striqoni, sveti, indexi2 = 0;
double sashualo;

for ( sveti = 0; sveti < 3; sveti += 2 )
{
    sashualo = 0;
    for ( striqoni = 0; striqoni < 3; striqoni++ )
        sashualo += masivi1[striqoni][sveti];
    masivi2[indexi2++] = sashualo / 3;
}
for ( striqoni = 0; striqoni < 3; striqoni++ )
{
    for ( sveti = 0; sveti < 3; sveti++ )
        cout << masivi1[striqoni][sveti] << " ";
    cout << endl;
}
for ( striqoni = 0; striqoni < indexi2; striqoni++ )
    cout << masivi2[striqoni] << " ";
cout << endl;

system("pause");
return 0;
}

```

4.2.37.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi1[4][4] = { { 10, 21, 3, 2 }, { 4, 51, 6, 5 }, { 7, 3, 9, 6 }, { 4, 1, 8, 9 } };
double masivi2[4] = { 0, 0, 0, 0 };
int striqoni, sveti, indexi2 = 0;
double sashualo;

for ( striqoni = 1; striqoni < 4; striqoni += 2 )
{
    sashualo = 1;
    for ( sveti = 0; sveti < 4; sveti++ )
        sashualo *= masivi1[striqoni][sveti];
    masivi2[indexi2++] = sashualo / 4;
}
for ( striqoni = 0; striqoni < 4; striqoni++ )
{
    for ( sveti = 0; sveti < 4; sveti++ )
        cout << masivi1[striqoni][sveti] << " ";
    cout << endl;
}
for ( striqoni = 0; striqoni < indexi2; striqoni++ )
    cout << masivi2[striqoni] << " ";
cout << endl;
}

```

```

system("pause");
return 0;
}

```

4.2.38.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi1[4][4] = { { 10, 21, 3, 2 }, { 0, 0, 0, 0 }, { 0, 0, 0, 0 }, { 4, 1, 8, 9 } };
int masivi2[4] = { 0, 0, 0, 0 };
int striqoni, sveti, indexi2 = 0, raodenoba = 0;

for ( striqoni = 0 ; striqoni < 4 ; striqoni++ )
{
    raodenoba = 0;
    for ( sveti = 0 ; sveti < 4 ; sveti++ )
        if ( masivi1[striqoni][sveti] == 0 ) raodenoba++;
    if ( raodenoba == 4 ) masivi2[indexi2++] = striqoni;
}
for ( striqoni = 0 ; striqoni < 4 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 4 ; sveti++ )
        cout << masivi1[striqoni][sveti] << " ";
    cout << endl;
}
for ( striqoni = 0 ; striqoni < indexi2 ; striqoni++ )
    cout << masivi2[striqoni] << " ";
cout << endl;

system("pause");
return 0;
}

```

4.2.39.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi1[3][3] = { { 10, 21, 3 }, { 4, 51, 61 }, { 7, 3, 9 } };
int masivi2[3] = { 0, 0, 0 };
int striqoni, sveti, raodenoba = 0, indexi2 = 0;

for ( sveti = 0 ; sveti < 3 ; sveti++ )
{
    raodenoba = 0;
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        if ( masivi1[striqoni][sveti] % 2 == 1 ) raodenoba++;
    if ( raodenoba == 3 ) masivi2[indexi2++] = sveti;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        cout << masivi1[striqoni][sveti] << " ";
}
}

```

```

    cout << endl;
}
for ( striqoni = 0 ; striqoni < indexi2 ; striqoni++ )
    cout << masivi2[striqoni] << " ";
cout << endl;

system("pause");
return 0;
}

```

4.2.46.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi[3][3] = { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
int striqoni, sveti, temp;

for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        cout << masivi[striqoni][sveti] << " ";
    cout << endl;
}
// შეგვაქვს სვეტის ინდექსი, რომლის მნიშვნელობა უნდა იყოს 3-ზე ნაკლები
cin >> sveti;
if ( sveti >= 3 )
{
    cout << "araswori indeqsi, gaimeoret shetana";
    return 0;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    temp = masivi[striqoni][sveti];
    masivi[striqoni][sveti] = masivi[striqoni][2];
    masivi[striqoni][2] = temp;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        cout << masivi[striqoni][sveti] << " ";
    cout << endl;
}

system("pause");
return 0;
}

```

4.2.47.

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi[3][3] = { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
int striqoni, sveti, temp;

```

```

for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        cout << masivi[striqoni][sveti] << " ";
    cout << endl;
}
// შეგვაქვს სტრიქონის ინდექსი, რომლის მნიშვნელობა უნდა იყოს 3-ზე ნაკლები
cin >> striqoni;
if ( striqoni >= 3 )
{
    cout << "araswori indwqsi, gaimeoret shetana";
    return 0;
}
for ( sveti = 0 ; sveti < 3 ; sveti++ )
{
    temp = masivi[striqoni][sveti];
    masivi[striqoni][sveti] = masivi[0][sveti];
    masivi[0][sveti] = temp;
}
for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
{
    for ( sveti = 0 ; sveti < 3 ; sveti++ )
        cout << masivi[striqoni][sveti] << " ";
    cout << endl;
}

system("pause");
return 0;
}

```

4.2.65.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi1[3][3] = { { 11, -2, 3 }, { 4, -15, -6 }, { -7, 8, 9 } };
    int masivi2[3] = { 0, 0, 0 };
    int striqoni, sveti, min, minind, raodenoba = 0;

    for ( sveti = 0 ; sveti < 3 ; sveti++ )
    {
        raodenoba = 0;
        for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
            if ( masivi1[striqoni][sveti] >= 0 ) raodenoba++;
        masivi2[sveti] = raodenoba;
    }
    min = masivi2[0];
    minind = 0;
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
        if ( min > masivi2[striqoni] )
            {

```

```

        min = masivi2[striqoni];
        minind = striqoni;
    }
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            cout << masivi1[striqoni][sveti] << " ";
        cout << endl;
    }
    cout << minind << endl;

    system("pause");
    return 0;
}

```

4.2.78.

```

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[3][3] = { { 10, 21, 3 }, { 4, 51, 6 }, { 7, 3, 9 } };
    int striqoni, sveti, temp;
    //   ინდექსების მნიშვნელობები უნდა იყოს 3-ზე ნაკლები
    cin >> striqoni >> sveti;
    temp = masivi[striqoni][sveti];
    for ( int striqoni1 = 0; striqoni1 < 3; striqoni1++ )
        masivi[striqoni1][sveti] = 0;
    for ( int sveti1 = 0 ; sveti1 < 3 ; sveti1++ )
        masivi[striqoni][sveti1] = 0;
    masivi[striqoni][sveti] = temp;
    for ( striqoni = 0 ; striqoni < 3 ; striqoni++ )
    {
        for ( sveti = 0 ; sveti < 3 ; sveti++ )
            cout << masivi[striqoni][sveti] << " ";
        cout << endl;
    }

    system("pause");
    return 0;
}

```

შემთხვევითი რიცხვების გენერატორი

4.3.4.

```

#include "stdafx.h"
#include "iostream"
#include "ctime"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    srand(time(NULL));
    int min_rixvi, max_rixvi, ramdenjer, mocemuli_rixvi;

```

```

int raodenoba = 0, ricxvi;
int generirebuli_ricxvebis_raodenoba = 0;

cin >> min_rixcvi;           //    1
cin >> max_rixcvi;           //   30
cin >> ramdenjer;           //    7
cin >> mocemuli_rixcvi;      //   10

for ( ;; )
{
    ricxvi = rand() % (max_rixcvi + 1 - min_rixcvi) + min_rixcvi;
    generirebuli_ricxvebis_raodenoba++;
    if ( ricxvi == mocemuli_rixcvi ) raodenoba++;
    if (raodenoba == ramdenjer) break;
}
cout << generirebuli_ricxvebis_raodenoba << endl;

system("pause");
return 0;
}
4.3.5.
#include "stdafx.h"
#include "iostream"
#include "ctime"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    srand(time(NULL));
    int masivi[5] = { 20, 20, 20, 20, 20 };
    int shemtxveviti_rixcvi, asantebis_raodenoba = 0;

    for ( ;; )
    {
        shemtxveviti_rixcvi = rand() % ( 5 - 0 ) + 0;
        masivi[shemtxveviti_rixcvi]--;
        asantebis_raodenoba++;
        if (masivi[shemtxveviti_rixcvi] == 0) break;
    }
    cout << "unda amovigot " << asantebis_raodenoba
        << " asanti imistvis, rom ert-erti kolofi dacarieldes" << endl;

    system("pause");
    return 0;
}

```


თავი 5. სტრიქონები

5.1.1.

```
#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    string str1 = "Roman Samkharadze";
    int raodenoba;
    raodenoba = str1.size();
    cout << raodenoba << endl;
    //
    system("pause");
    return 0;
}
```

5.1.2.

```
#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    string str1 = "Roman Samkharadze";
    int indexi, raodenoba = 0;

    for (indexi = 0; indexi < str1.length(); indexi++)
        if (str1[indexi] == 'a') raodenoba++;
        cout << "a simbolos raodenoba = " << raodenoba << endl;
    //
    system("pause");
    return 0;
}
```

5.1.3.

```
#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    string str1 = "a1b2bc34de8ot6r";
    int indexi, raodenoba = 0;

    for (indexi = 0; indexi < str1.length(); indexi++)
        if (str1[indexi] >= '0' && str1[indexi] <= '9') raodenoba++;
}
```

```

        cout << "cifrebis raodenoba = " << raodenoba << endl;
    //
    system("pause");
    return 0;
}

```

5.1.4.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    string str1 = "a,b2:c;34d!8o?6,r!";
    int indexi, raodenoba = 0;

    for (indexi = 0; indexi < str1.length(); indexi++)
        switch (str1[indexi])
        {
            case ',':
            case ';':
            case '!':
            case '?':
            case ':': raodenoba++; break;
        }

    cout << "sasveni nishnebis raodenoba = " << raodenoba << endl;
    //
    system("pause");
    return 0;
}

```

5.1.6.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    string str1 = "Roman Samkharadze";
    int indexi, raodenoba = 0;

    indexi = str1.find('n');
    cout << "n simbolos pozicia = " << indexi << endl;
    //
    system("pause");
    return 0;
}

```

5.1.7.

```

#include "stdafx.h"

```

```

#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    string str1 = "Roman Samkharadze";
    int indexi, raodenoba = 0;

    cout << "sawyisi striqoni - " << str1 << endl;
    for (indexi = 0; indexi < str1.length(); indexi++)
        if (str1[indexi] == 'a') str1[indexi] = 'i';
    cout << "shecvili striqoni - " << str1 << endl;
    //
    system("pause");
    return 0;
}

```

5.1.8.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    string str1 = "a1b2c34de8m9";
    int indexi, raodenoba = 0;

    for (indexi = 0; indexi < str1.length(); indexi++)
        if (str1[indexi] >= '0' && str1[indexi] <= '9') str1[indexi] = 'r';
    cout << "shecvili striqoni - " << str1 << endl;
    //
    system("pause");
    return 0;
}

```

თავი 6. ინფორმაციის შეტანა-გამოტანა

6.1.1.

```

#include "stdafx.h"
#include "iostream"
#include "fstream"
#include "ctime"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    srand(time(NULL));
}

```

```

int indexi, jami = 0;
int masivi[5];
// ფაილის შექმნა შემთხვევითი რიცხვების ჩასაწერად
ofstream out_file("file_1.dat");
for ( indexi = 0; indexi < 5; indexi++ )
out_file << rand() % (10 - 1) + 1 << " "; // [1-9]
out_file.close();
// ფაილის გახსნა რიცხვების წასაკითხად და მასივში მოსათავსებლად
ifstream in_file("file_1.dat");
for ( indexi = 0; indexi < 5; indexi++ )
in_file >> masivi[indexi];
in_file.close();
// ჯამის გამოთვლა
for ( indexi = 0; indexi < 5; indexi++ )
jami += masivi[indexi];
// ფაილში ჯამის ჩაწერა
ofstream out_jami("jami.dat");
out_jami << jami;
out_jami.close();
// მასივის ელემენტების ეკრანზე გამოტანა
for ( indexi = 0; indexi < 5; indexi++ )
cout << masivi[indexi] << " ";
cout << endl;
cout << "jami = " << jami << endl;

system("pause");
return 0;
}

```

6.1.2.

```

#include "stdafx.h"
#include "iostream"
#include "fstream"
#include "ctime"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    srand(time(NULL));
    int indexi, jami = 0;
    int masivi[5];
    // ფაილის შექმნა შემთხვევითი რიცხვების ჩასაწერად
    ofstream out_file("file_1.dat");
    for ( indexi = 0; indexi < 5; indexi++ )
out_file << rand() % (10 - 1) + 1 << " "; // [1-9]
out_file.close();
    // ფაილის გახსნა რიცხვების წასაკითხად და მასივში მოსათავსებლად
    ifstream in_file("file_1.dat");
    for ( indexi = 0; indexi < 5; indexi++ )
in_file >> masivi[indexi];
}

```

```

in_file.close();
//      ჯამის გამოთვლა
for ( indexi = 0; indexi < 5; indexi++ )
    jami += masivi[indexi];
ofstream out_jami("jami_1.dat");
out_jami << jami;
for ( indexi = 0; indexi < 5; indexi++ )
    out_jami << masivi[indexi] << " ";
out_jami.close();
//      მასივის ელემენტების ეკრანზე გამოტანა
for ( indexi = 0; indexi < 5; indexi++ )
    cout << masivi[indexi] << " ";
cout << endl;
cout << "jami = " << jami << endl;

system("pause");
return 0;

```

```

}

```

6.1.3.

```

#include "stdafx.h"
#include "iostream"
#include "fstream"
#include "ctime"
using namespace std;

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    srand(time(NULL));
    int indexi, ricxvi;
    //      ფაილის შექმნა შემთხვევითი რიცხვების ჩასაწერად
    ofstream out_file("file_1.dat");
    for ( indexi = 0; indexi < 10; indexi++ )
        out_file << rand() % (10 - 1) + 1 << " "; // [1-9]
    out_file.close();
    //
    ofstream out_file1("file_2.dat");
    ifstream in_file("file_1.dat");
    for ( indexi = 0; indexi < 10; indexi++ )
    {
        in_file >> ricxvi;
        out_file1 << ricxvi;
    }
    in_file.close();
    out_file1.close();
    //      მასივის ელემენტების ეკრანზე გამოტანა
    ifstream in_file1("file_1.dat");
    ifstream in_file2("file_2.dat");
    for ( indexi = 0; indexi < 5; indexi++ )
        {

```

```

        in_file1 >> ricxvi;
        cout << ricxvi << " ";
        in_file2 >> ricxvi;
        cout << ricxvi << " ";
        cout << endl;
    }
    in_file1.close();
    in_file2.close();

    system("pause");
    return 0;
}

```

6.1.7.

```

#include "stdafx.h"
#include "iostream"
#include "fstream"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int indexi;
    char simbolo;
    char simboloebis_masivi[5] = { 'r', 'o', 'm', 'a', 'n' };
    char simboloebis_masivi2[5];
    ofstream file_out("file_1.dat");

    for ( indexi = 0; indexi < 5; indexi++ )
    {
        file_out << simboloebis_masivi[indexi];
    }
    file_out.close();
    //
    ifstream file_in("file_1.dat");
    for ( indexi = 0; indexi < 5; indexi++ )
    {
        file_in >> simboloebis_masivi2[indexi];
        cout << simboloebis_masivi2[indexi] << " ";
    }
    file_in.close();

    ofstream file_out_2("file_2.dat");
    for ( indexi = 0; indexi < 5; indexi++ )
        file_out_2 << simboloebis_masivi2[indexi] << " ";
    file_out_2.close();
    cout << endl;

    system("pause");
    return 0;
}

```

6.1.9.

```
#include "stdafx.h"
#include "iostream"
#include "fstream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    string striqoni = "Romani", striqoni1, striqoni2;
    //
    ofstream file_out("file_1.txt");
    file_out << striqoni;
    file_out.close();
    //
    ifstream file_in("file_1.txt");
    file_in >> striqoni1;
    file_in.close();
    //
    ofstream file_out2("file_2.txt");
    file_out2 << striqoni1;
    file_out2.close();
    //
    ifstream file_in2("file_2.txt");
    file_in2 >> striqoni2;
    file_in2.close();
    //
    cout << "striqoni - " << striqoni2 << endl;

    system("pause");
    return 0;
}
```

6.1.10.

```
#include "stdafx.h"
#include "iostream"
#include "fstream"
#include "string"
using namespace std;

int _tmain(int argc, _TCHAR* argv[])
{
    int indexi;
    string masivi[4] = { "saba", "ana", "lika", "romani" };
    string masivi2[4];
    ofstream file_out("file.txt");
    for (indexi = 0; indexi < 4; indexi++)
        file_out << masivi[indexi] << " ";
    file_out.close();
    ifstream file_in("file.txt");
}
```

```

    for (indexi = 0; indexi < 4; indexi++)
        file_in >> masivi2[indexi];
    file_in.close();
    for (indexi = 0; indexi < 4; indexi++)
        cout << masivi2[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}

```

თავი 7. ფუნქციები

ფუნქციები

7.1.1.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

double Fartobi(int simagle, int puze)
{
    return simagle * puze / 2;
}
int _tmain(int argc, _TCHAR* argv[])
{
    int simagle, puze;
    double fartobi;
    cin >> simagle >> puze;
    fartobi = Fartobi(simagle, puze);
    cout << "martkuxa samkuxedis fartobi = " << fartobi << endl;
    //
    system("pause");
    return 0;
}

```

7.1.4.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// ფუნქციის პროტოტიპი
int Perimetri(int);

int _tmain(int argc, _TCHAR* argv[])
{
    int gverdi;
    double perimetri;
    cin >> gverdi;
    perimetri = Perimetri(gverdi);
    cout << "kvadratis perimetri = " << perimetri << endl;
}

```



```

    //
    system("pause");
    return 0;
}
int Perimetri(int gverdi)
{
    return 4 * gverdi;
}

```

7.1.13.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

// ფუნქციის პროტოტიპი
string Uaryopiti(int, int);

int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi1, ricxvi2;
    string shedegi;

    cin >> ricxvi1 >> ricxvi2;
    shedegi = Uaryopiti(ricxvi1, ricxvi2);
    cout << shedegi << endl;
    //
    system("pause");
    return 0;
}
string Uaryopiti(int ricxvi1, int ricxvi2)
{
    if (ricxvi1 < 0 || ricxvi2 < 0)
        return "ert-erti ricxvi uaryopitia";
    else return "arc erti ricxvi ar aris uaryopiti";
}

```

7.1.15.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// ფუნქციის პროტოტიპი
void Diapazoni(int);

int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi;

    cin >> ricxvi;
    Diapazoni(ricxvi);
}

```

```

//
system("pause");
return 0;
}
void Diapazoni(int ricxvi)
{
    if (ricxvi > 0 && ricxvi <= 17)
        cout << "ricxvi mititebul diapazonshia" << endl;
    else cout << "ricxvi ar aris mititebul diapazonshi" << endl;
}

```

7.1.18.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// ფუნქციის პროტოტიპი
int Luwi_Raodenoba();

int _tmain(int argc, _TCHAR* argv[])
{
    int raodenoba;

    raodenoba = Luwi_Raodenoba();
    cout << "5-dan 27-mde diapazonshi luwi ricxvebis raodenoba = " << raodenoba << endl;
    //
    system("pause");
    return 0;
}
int Luwi_Raodenoba()
{
    int cvladi, raodenoba = 0;
    for (cvladi = 6; cvladi <= 27; cvladi += 2)
        raodenoba++;
    return raodenoba;
}

```

7.1.21.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// ფუნქციის პროტოტიპი
int Jeradi_Ricxvebi();

int _tmain(int argc, _TCHAR* argv[])
{
    int jami;

    jami = Jeradi_Ricxvebi();
    cout << "2-dan 47-mde diapazonshi 30-ze naklebi 9-is jeradi ricxvebis jami = " << jami << endl;
}

```

```

        //
        system("pause");
        return 0;
    }
int Jeradi_Ricxvebi()
{
    int cvladi, jami = 0;
    cout << "30-ze naklebi 9-is jeradi ricxvebi: ";
    for (cvladi = 2; cvladi <= 47; cvladi++)
        if (cvladi % 9 == 0 && cvladi < 30)
        {
            cout << cvladi << " ";
            jami += cvladi;
        }
    cout << endl;
    return jami;
}

```

ფუნქციისთვის მასივების გადაცემა

7.2.1.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// ფუნქციის პროტოტიპი
int Masvi_Jami(int []);

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[] = { 2, -1, 5, -3, 9 };
    int jami;

    jami = Masvi_Jami(masivi);
    cout << "masivis elementebis jami = " << jami << endl;
    //
    system("pause");
    return 0;
}
int Masvi_Jami(int masivi[5])
{
    int indexi, jami = 0;

    for (indexi = 0; indexi < 5; indexi++)
        jami += masivi[indexi];
    return jami;
}

```

7.2.9.

```

#include "stdafx.h"
#include "iostream"

```

```

using namespace std;

// ფუნქციის პროტოტიპი
int Masvi_Jami(int []);

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[] = { 28, -1, 56, -35, 9 };
    int raodenoba;

    raodenoba = Masvi_Jami(masivi);
    cout << "masivis 7-is jeradi elementebis raodenoba = " << raodenoba << endl;
    //
    system("pause");
    return 0;
}
int Masvi_Jami(int masivi[5])
{
    int indexi, raodenoba = 0;

    for (indexi = 0; indexi < 5; indexi++)
        if (masivi[indexi] % 7 == 0) raodenoba++;
    return raodenoba;
}

```

7.2.20.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// ფუნქციის პროტოტიპი
void Masvis_Gadawera(int [], int []);

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi1[] = { 28, -1, 56, -35, 9 };
    int masivi2[5];
    int indexi;

    Masvis_Gadawera(masivi1, masivi2);
    cout << "pirveli masivi - " << endl;
    for (indexi = 0; indexi < 5; indexi++)
    {
        cout << masivi1[indexi] << " ";
    }
    cout << endl;
    cout << "meore masivi - " << endl;
    for (indexi = 0; indexi < 5; indexi++)
    {
        cout << masivi2[indexi] << " ";
    }
}

```

```

    }
    cout << endl;
    //
    system("pause");
    return 0;
}
void Masvis_Gadawera(int masivi1[], int masivi2[])
{
    int indexi, raodenoba = 0;

    for (indexi = 0; indexi < 5; indexi++)
    {
        masivi1[indexi] += 10;
        masivi2[indexi] = masivi1[indexi];
    }
}

```

7.2.27.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

//      ფუნქციის პრეპროცესინგი
int Aranulovani_Raodenoba(int []);

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[] = { 28, 0, 56, -35, 0 };
    int indexi, raodenoba;

    raodenoba = Aranulovani_Raodenoba(masivi);
    for (indexi = 0; indexi < 5; indexi++)
        cout << masivi[indexi] << " ";
    cout << endl;
    cout << "masivis aranulovani elementebis raodenoba = " << raodenoba << endl;
    //
    system("pause");
    return 0;
}
int Aranulovani_Raodenoba(int masivi[])
{
    int indexi, raodenoba = 0;

    for (indexi = 0; indexi < 5; indexi++)
        if (masivi[indexi] != 0) raodenoba++;
    return raodenoba;
}

```

7.2.28.

```

#include "stdafx.h"
#include "iostream"

```

```

using namespace std;

// ფუნქციის პროტოტიპი
int Aranulovani_Raodenoba(int [][][3]);

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi[3][3] = { { 8, 0, 5 }, { -5, 0, -6 }, { 0, 0, 9 } };
    int str, sv, raodenoba;

    raodenoba = Aranulovani_Raodenoba(masivi);
    for (str = 0; str < 3; str++)
    {
        for (sv = 0; sv < 3; sv++)
            cout << masivi[str][sv] << " ";
        cout << endl;
    }
    cout << endl;
    cout << "masivis aranulovani elementebis raodenoba = " << raodenoba << endl;
    //
    system("pause");
    return 0;
}

int Aranulovani_Raodenoba(int masivi[][][3])
{
    int str, sv, raodenoba = 0;

    for (str = 0; str < 3; str++)
        for (sv = 0; sv < 3; sv++)
            if (masivi[str][sv] == 0) raodenoba++;
    return raodenoba;
}

```

პარამეტრების გადაცემა მიმთითებლებისა და მიმართვების გამოყენებით

7.3.1.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// ფუნქციის პროტოტიპი
int Fartobi_Perimetri(int, int *);

int _tmain(int argc, _TCHAR* argv[])
{
    int gverdi;
    int fartobi, perimetri;

    cin >> gverdi;
    fartobi = Fartobi_Perimetri(gverdi, &perimetri);
}

```

```

        cout << "kvadratis fartobi = " << fartobi << "\nkvadratis perimetri = " << perimetri << endl;
        //
        system("pause");
        return 0;
    }
    int Fartobi_Perimetri(int gverdi, int *perimetri)
    {
        *perimetri = 4 * gverdi;
        return gverdi * gverdi;
    }

```

7.3.7.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// ფუნქციის პროტოტიპი
int Fartobi_Perimetri(int, int &);

int _tmain(int argc, _TCHAR* argv[])
{
    int gverdi;
    int fartobi, perimetri;

    cin >> gverdi;
    fartobi = Fartobi_Perimetri(gverdi, perimetri);
    cout << "kvadratis fartobi = " << fartobi << "\nkvadratis perimetri = " << perimetri << endl;
    //
    system("pause");
    return 0;
}
int Fartobi_Perimetri(int gverdi, int &perimetri)
{
    perimetri = 4 * gverdi;
    return gverdi * gverdi;
}

```

ფუნქციის გადატვირთვა

7.4.1.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// ფუნქციის პროტოტიპი
int Max(int, int);
double Max(double, double);

int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi1, ricxvi2, max1;

```

```

double ricxvi3, ricxvi4, max2;

cout << "sheitaniet mteli ricxvebi ";
cin >> ricxvi1 >> ricxvi2;
max1 = Max(ricxvi1, ricxvi2);
cout << "or mtel ricxvs shoris maqsimaluri = " << max1 << endl;
cout << "\nsheitaniet wiladi ricxvebi ";
cin >> ricxvi3 >> ricxvi4;
max2 = Max(ricxvi3, ricxvi4);
cout << "or wilad ricxvs shoris maqsimaluri = " << max2 << endl;
//
system("pause");
return 0;
}
int Max(int ricxvi1, int ricxvi2)
{
    if (ricxvi1 > ricxvi2) return ricxvi1;
    else return ricxvi2;
}
double Max(double ricxvi1, double ricxvi2)
{
    if (ricxvi1 > ricxvi2) return ricxvi1;
    else return ricxvi2;
}

```

7.4.8.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

// ფუნქციის პროტოტიპი
void Uaryofiti(int);
void Uaryofiti(double);

int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi1;
    double ricxvi2;

    cout << "sheitaniet mteli ricxvi ";
    cin >> ricxvi1;
    Uaryofiti(ricxvi1);
    cout << "\nsheitaniet wiladi ricxvi ";
    cin >> ricxvi2;
    Uaryofiti(ricxvi2);
    //
    system("pause");
    return 0;
}

```



```

void Uaryofiti(int ricxvi)
{
    if (ricxvi < 0) cout << "parametri uaryofitia" << endl;
    else cout << "parametri ar aris uaryofiti" << endl;
}
void Uaryofiti(double ricxvi)
{
    if (ricxvi < 0) cout << "parametri uaryofitia" << endl;
    else cout << "parametri ar aris uaryofiti" << endl;
}

```

ფუნქციისთვის პარამეტრების გადაცემა გულისხმობის პრინციპით

7.5.2.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

// ფუნქციის პროტოტიპი
int Namravli(int, int, int = 1);

int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi1, ricxvi2, ricxvi3, jami;

    cout << "sheitaniet 3 mteli ricxvi ";
    cin >> ricxvi1 >> ricxvi2 >> ricxvi3;
    jami = Namravli(ricxvi1, ricxvi2, ricxvi3);
    cout << "3 mteli ricxvis namravli = " << jami << endl;
    cout << "sheitaniet 2 mteli ricxvi ";
    cin >> ricxvi1 >> ricxvi2;
    jami = Namravli(ricxvi1, ricxvi2);
    cout << "2 mteli ricxvis namravli = " << jami << endl;
    //
    system("pause");
    return 0;
}
int Namravli(int ricxvi1, int ricxvi2, int ricxvi3)
{
    return ricxvi1 * ricxvi2 * ricxvi3;
}

```

თავი 8. კლასი, ინკაფსულაცია, მეთოდი
კლასი. ინკაფსულაცია

8.1.1.

```

class Tvitmfrinavi
{
    int banis_tevadoba;
    int manzili;
}

```

```

public :
int mgzavrebis_raodenoba;
int gayiduli_bilitebi;
};

int _tmain(int argc, _TCHAR* argv[])
{
Tvitmfrinavi obieqti_1;
cin >> obieqti_1.mgzavrebis_raodenoba;
cin >> obieqti_1.gayiduli_bilitebi;
cout << obieqti_1.mgzavrebis_raodenoba << endl;
cout << obieqti_1.gayiduli_bilitebi << endl;

system("pause");
return 0;
}

```

8.1.2.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

```

```

class Studenti
{
string gvare;
string saxeli;
int asaki;
public:
string universitetis_dasaxeleba;
int kursi;
};

```

```

int _tmain(int argc, _TCHAR* argv[])
{
Studenti obieqti_1;
cin >> obieqti_1.universitetis_dasaxeleba >> obieqti_1.kursi;
cout << "universitetis dasaxeleba - " << obieqti_1.universitetis_dasaxeleba << "      kursi - " <<
obieqti_1.kursi << endl;

system("pause");
return 0;
}

```

ფუნქციის

8.2.1.

```

class Studenti_1
{
public:
double Sashualo_qula(int masivi1[], int size)

```

```

    {
        int jami = 0;
        for (int ind = 0; ind < size; ind++)
            jami += masivi1[ind];
        return jami / size;
    }
};

```

```

int _tmain(int argc, _TCHAR* argv[])
{
int masivi[10] = { 60, 87, 71, 90, 94, 58, 83, 57, 70, 65 };
double shedegi;
Studenti_1 obieqti_1;
shedegi = obieqti_1.Sashualo_qula(masivi, 10);
cout << shedegi << endl;

```

```

system("pause");
return 0;
}

```

8.2.2.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

```

```

class Studenti_2
{
    string gvari;
    string saxeli;
    int asaki;
    void minicheba(string par1, string par2, int par3)
    {
        gvari = par1;
        saxeli = par2;
        asaki = par3;
    }
public:
    void gadacema(string par1, string par2, int par3)
    {
        minicheba(par1, par2, par3);
    }
    void gamotana()
    {
        cout << "studentis gvari: " << gvari << "\nstudentis saxeli: " + saxeli
            << "\nstudentis asaki: " << asaki << endl;
    }
};

```

```

int _tmain(int argc, _TCHAR* argv[])

```

```

{
    Studenti_2 obieqti_1;
    string gvari;
    string saxeli;
    int asaki;

    cin >> gvari >> saxeli >> asaki;
    obieqti_1.gadacema(gvari, saxeli, asaki);
    obieqti_1.gamotana();

    system("pause");
    return 0;
}

```

8.2.3.

class Matarebeli

```

{
    int vagonebis_raodenoba;
    int mgzavrebis_raodenoba;
    public:
    double biletis_fasi;
    int gayiduli_biletebis_raodenoba;
    void minicheba(double par1, int par2, int par3, int par4)
    {
        biletis_fasi = par1;
        gayiduli_biletebis_raodenoba = par2;
        vagonebis_raodenoba = par3;
        mgzavrebis_raodenoba = par4;
    }
    void gamotana()
    {
        cout << "biletis fasi - " << biletis_fasi <<
            "\ngayiduli biletebis raodenoba - " << gayiduli_biletebis_raodenoba <<
            "\nvagonebis raodenoba - " << vagonebis_raodenoba <<
            "\nmgzavrebis raodenoba - " << mgzavrebis_raodenoba << endl;
    }
    double mogeba()
    {
        return biletis_fasi * gayiduli_biletebis_raodenoba;
    }
};

```

int _tmain(int argc, _TCHAR* argv[])

```

{
    int vagonebis_raodenoba, mgzavrebis_raodenoba;
    double shedegi;
    Matarebeli obieqti_1;

    cin >> vagonebis_raodenoba >> mgzavrebis_raodenoba;
    cin >> obieqti_1.biletis_fasi >> obieqti_1.gayiduli_biletebis_raodenoba;
}

```

```

obieqti_1.minicheba(obieqti_1.biletis_fasi, obieqti_1.gayiduli_biletebis_raodenoba,
                    vagonebis_raodenoba, mgzavrebis_raodenoba);
obieqti_1.gamotana();
shedegi = obieqti_1.mogeba();
cout << "biletebis gayidvit migebuli tanxa = " << shedegi << endl;

system("pause");
return 0;
}

```

კონსტრუქტორი

8.3.1.

```

class Tvitmfrinavi
{
    int bakis_tevadoba;
    int manzili_1_litri;
public:
    Tvitmfrinavi(int par1, int par2)
    {
        bakis_tevadoba = par1;
        manzili_1_litri = par2;
    }
    void Gamotana()
    {
        cout << "avzis tevadoba = " << bakis_tevadoba <<
            "\n1 litri sawvavit gavlili mandzili = " << manzili_1_litri << endl;
    }
    int Gamotvla()
    {
        return bakis_tevadoba * manzili_1_litri;
    }
};

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    int bakis_tevadoba, manzili_1_litri;

    cin >> bakis_tevadoba >> manzili_1_litri;
    Tvitmfrinavi obieqti_1(bakis_tevadoba, manzili_1_litri);
    obieqti_1.Gamotana();
    int shedegi = obieqti_1.Gamotvla();
    cout << "savse avzit gavlili mandzili = " << shedegi << endl;

    system("pause");
    return 0;
}

```

8.3.3.

```

class Martkutxedi
{

```

```

int perimetri;
int fartobi;
public:
int gverdi_1;
int gverdi_2;
Martkutxedi(int par1, int par2)
{
    gverdi_1 = par1;
    gverdi_2 = par2;
    perimetri = ( gverdi_1 + gverdi_2 ) * 2;
    fartobi = gverdi_1 * gverdi_2;
}
void Gamotana()
{
    cout << "perimetri = " << perimetri << "\nfartobi = " << fartobi << endl;
}
};

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    int gverdi_1, gverdi_2;

    cin >> gverdi_1 >> gverdi_2;
    Martkutxedi obieqti_1 (gverdi_1, gverdi_2);
    obieqti_1.Gamotana();

    system("pause");
    return 0;
}

```

ფუნქციების გადატვირთვა

8.4.2.

```

class Figura
{
public:
int Perimetri(int par1)
{
    return 4 * par1;
}
int Perimetri(int par1, int par2)
{
    return 2 * (par1 + par2);
}
int Perimetri(int par1, int par2, int par3)
{
    return par1 + par2 + par3;
}
};

```

```

int _tmain(int argc, _TCHAR* argv[])
{
Figura obieqti_1;
int gverdi_1, gverdi_2, gverdi_3;

cin >> gverdi_1 >> gverdi_2 >> gverdi_3;
int kvadratis_perimetri = obieqti_1.Perimetri(gverdi_1);
int otxkutxedis_perimetri = obieqti_1.Perimetri(gverdi_1, gverdi_2);
int samkutxedis_perimetri = obieqti_1.Perimetri(gverdi_1, gverdi_2, gverdi_3);

cout << "kvadratis perimetri = " << kvadratis_perimetri << endl;
cout << "\notxkutxedis perimetri = " << otxkutxedis_perimetri << endl;
cout << "\nsamkutxedis perimetri = " << samkutxedis_perimetri << endl;

system("pause");
return 0;
}

```

8.4.3.

```

class Avtomobili
{
public : int Metodi_1(int par1, int par2)
{
return par1 * par2;
}
double Metodi_1(double par1, double par2)
{
return par1 * par2;
}
};

```

```

int _tmain(int argc, _TCHAR* argv[])
{
Avtomobili obieqti_1;
int bakis_tevadoba, manzili_1_litri;
double maqsimaluri_sichqare, dro;

cin >> bakis_tevadoba >> manzili_1_litri >> maqsimaluri_sichqare >> dro;
int manzili_savse_baki = obieqti_1.Metodi_1(bakis_tevadoba, manzili_1_litri);
double manZili_dro = obieqti_1.Metodi_1(dro, maqsimaluri_sichqare);

cout << "savse avzit gavlili mandzili = " << manzili_savse_baki << endl
<< "maqsilamuri sichqarit modzraobisas gavlili mandzili = "
<< manZili_dro << endl;

system("pause");
return 0;
}

```

კონსტრუქტორის გადატვირთვა

8.5.1.

```
class ChemiKlasi
{
    int Min;
    public :
    ChemiKlasi(int masivi1[], int size)
    {
        Min = masivi1[0];
        for ( int ind = 1; ind < size; ind++ )
            if ( masivi1[ind] < Min ) Min = masivi1[ind];
    }
    ChemiKlasi(int p1, int p2)
    {
        if ( p1 > p2 ) cout << "min = " << p2 << endl;
        else cout << "min = " << p1 << endl;
    }
};

int _tmain(int argc, _TCHAR* argv[])
{
    int masivi1[5] = { 6, 2, -9, 4, -7 };
    int ricxvi1, ricxvi2;

    cin >> ricxvi1 >> ricxvi2;
    ChemiKlasi obieqti1(masivi1, 5);
    ChemiKlasi obieqti2(ricxvi1, ricxvi2);

    system("pause");
    return 0;
}
```

8.5.2.

```
class Chemi_Klasi
{
    int gverdi1;
    int gverdi2;
    int gverdi3;
    int perimetri;
    double fartobi;
    public:
    Chemi_Klasi(int par1)
    {
        gverdi1 = par1;
        perimetri = 4 * par1;
        fartobi = gverdi1 * gverdi1;
    }
    Chemi_Klasi(int par1, int par2)
    {
        gverdi1 = par1;
        gverdi2 = par2;
    }
};
```



```

    perimetri = 2 * ( par1 + par2 );
    fartobi = par1 * par2;
}
Chemi_Klasi(int par1, int par2, int par3)
{
    gverdi1 = par1;
    gverdi2 = par2;
    gverdi3 = par3;
    perimetri = par1 + par2 + par3;
    fartobi = ( par1 * par2 ) / 2;
}
public : void Naxva()
{
    cout << "figuris perimetri = " << perimetri
        << "\nfiguris fartobi = " << fartobi << endl << endl;
}
};

int _tmain(int argc, _TCHAR* argv[])
{
    int gverdi1, gverdi2, gverdi3;

    cin >> gverdi1 >> gverdi2 >> gverdi3;
    Chemi_Klasi obieqti1(gverdi1);
    Chemi_Klasi obieqti2(gverdi1, gverdi2);
    Chemi_Klasi obieqti3(gverdi1, gverdi2, gverdi3);
    obieqti1.Naxva();
    obieqti2.Naxva();
    obieqti3.Naxva();

    system("pause");
    return 0;
}

```

თავი 9. ფუნქციები უფრო დაწვრილებით ჩასადგმელი ფუნქცია

9.1.2.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

class Klasi
{
    int gverdi1, gverdi2, gverdi3;
public:
    Klasi(int par1, int par2, int par3);
    int Perimetri(); // ჩადგმა სრულდება ამ გამოცხადებაში
};

```

```

Klasi::Klasi(int par1, int par2, int par3)
{
    gverdi1 = par1;
    gverdi2 = par2;
    gverdi3 = par3;
}
inline int Klasi::Perimetri() // Perimetri() ფუნქცია არის ჩასადგმელი
{
    return gverdi1 + gverdi2 + gverdi3;
}

int _tmain(int argc, _TCHAR* argv[])
{
    int gverdi1, gverdi2, gverdi3;

    cin >> gverdi1 >> gverdi2 >> gverdi3;
    Klasi obj(gverdi1, gverdi2, gverdi3);
    int perimetri = obj.Perimetri();
    cout << "samkutxedis perimetri = " << perimetri << endl;

    //
    system("pause");
    return 0;
}

```

9.1.15.

```

#include "stdafx.h"
#include "iostream"
using namespace std;

class Klasi
{
    int ricxvi1, ricxvi2;
public:
    Klasi(int par1, int par2);
    void Jeradi_5(); // ჩადგმა სრულდება ამ გამოცხადებაში
};
Klasi::Klasi(int par1, int par2)
{
    ricxvi1 = par1;
    ricxvi2 = par2;
}
inline void Klasi::Jeradi_5() // Jeradi_5() ფუნქცია არის ჩასადგმელი
{
    if (ricxvi1 % 5 == 0 || ricxvi2 % 5 == 0) cout << "ert-erti ricxvi 5-is jeradia" << endl;
    else cout << "arcerti ricxvi ar aris 5-is jeradi" << endl;
}

```

```

int _tmain(int argc, _TCHAR* argv[])
{
    int ricxvi1, ricxvi2;

    cin >> ricxvi1 >> ricxvi2;
    Klasi obj(ricxvi1, ricxvi2);
    obj.Jeradi_5();
    //
    system("pause");
    return 0;
}

```

ფუნქციის შაბლონი

9.3.1.

```

template < class Tipi_1 > void Funqcia(Tipi_1 masivi1[], Tipi_1 masivi2[], int &indexi2)
{
    int indexi;
    indexi2 = 0;
    // ერთი ერთგანზომილებიანი მასივიდან მეორე ერთგანზომილებიან მასივში დადებითი
    // რიცხვების გადაწერა
    for (indexi = 0; indexi < 5; indexi++)
        if (masivi1[indexi] >= 0) masivi2[indexi2++] = masivi1[indexi];
}
int _tmain(int argc, _TCHAR* argv[])
{
    /*int mteli1, mteli2;
    double wiladi1, wiladi2;

    cin >> mteli1 >> mteli2 >> wiladi1 >> wiladi2;
    int shedegi1 = Funqcia(mteli1, mteli2);
    double shedegi2 = Funqcia(wiladi1, wiladi2);
    cout << shedegi1 << endl;
    cout << shedegi2 << endl;*/
    int masivi1[] = { 1, -2, -3, 4, 5 };
    int masivi2[5] = { 0 };
    int indexi, raodenoba;

    Funqcia(masivi1, masivi2, raodenoba);
    cout << "masivi1 = ";
    for (indexi = 0; indexi < 5; indexi++)
        cout << masivi1[indexi] << " ";
    cout << endl;
    cout << "masivi2 = ";
    for (indexi = 0; indexi < raodenoba; indexi++)
        cout << masivi2[indexi] << " ";
    cout << endl;
}

```

```

    system("pause");
    return 0;
}
9.3.2.
template < class Tipi_1 > void Funqcia(Tipi_1 masivi1[][3], Tipi_1 masivi2[], int &indexi2)
{
    int str, sv;
    indexi2 = 0;
    // ორგანზომილებიანი მასივიდან ლუწი რიცხვების გადაწერა ერთგანზომილებიან მასივში
    for (str = 0; str < 3; str++)
        for (sv = 0; sv < 3; sv++)
            if (masivi1[str][sv] % 2 == 0)
                masivi2[indexi2++] = masivi1[str][sv];
}
int _tmain(int argc, _TCHAR* argv[])
{
    int masivi1[3][3] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
    int masivi2[9] = { 0 };
    int str, sv, indexi, raodenoba;

    Funqcia(masivi1, masivi2, raodenoba);
    cout << "masivi1 =\n";
    for (str = 0; str < 3; str++)
    {
        for (sv = 0; sv < 3; sv++)
            cout << masivi1[str][sv] << " ";
        cout << endl;
    }
    cout << endl;
    cout << "masivi2 = ";
    for (indexi = 0; indexi < raodenoba; indexi++)
        cout << masivi2[indexi] << " ";
    cout << endl;

    system("pause");
    return 0;
}

```

თავი 10. სტრუქტურები

სტრუქტურა

10.1.1.

```

#include "stdafx.h"
#include "iostream"
#include "string"
using namespace std;

```

```

struct Studenti
{
    //      ცვლადების გამოცხადება
    string saxeli;
    string gvari;
    int  kursi;
    double tanxa;
    //      კონსტრუქტორის განსაზღვრა
    Studenti(string par1, string par2, int par3, double par4)
    {
        saxeli = par1;
        gvari = par2;
        kursi = par3;
        tanxa = par4;
    }
};

int _tmain(int argc, _TCHAR* argv[])
{
    string saxeli, gvari;
    int kursi;
    double tanxa;
    cin >> saxeli >> gvari >> kursi >> tanxa;
    Studenti struqtura1(saxeli, gvari, kursi, tanxa);
    cout << "saxeli = " << saxeli << "\ngvari = " << gvari
        << "\nkursi = " << kursi << "\ntanxa = " << tanxa << endl;
    //
    system("pause");
    return 0;
}

```

ჩამოთვლა

10.2.3.

```

#include "stdafx.h"
#include "iostream"
using namespace std;
enum Qalaqi { Tbilisi, Batumi, Zestafoni, Gori, Qutaisi, Telavii, Qobuleti };
int _tmain(int argc, _TCHAR* argv[])
{
    Qalaqi kd = Qalaqi::Zestafoni;
    cout << kd << endl;           //      2
    cout << Qalaqi::Telavii << endl; //      5
    //
    system("pause");
    return 0;
}

```

ლიტერატურა

1. რ. სამხარაძე. Visual_C++_CLI_.NET. საქართველოს ტექნიკური უნივერსიტეტი. 2013. - 493 გვ.
2. რ. სამხარაძე. Visual C#.NET. საქართველოს ტექნიკური უნივერსიტეტი. 2014. - 507 გვ.
3. რ. სამხარაძე. SQL სერვერი. საქართველოს ტექნიკური უნივერსიტეტი. 2016. - 453 გვ.
4. Шилдт. Г. Самоучитель С++ : Пер. с англ. СПб. 2001. – 688 с.
5. Хортон А. Visual С++ 2005. Базовый курс.
6. Юлин В., Булатова. И. Приглашение к СИ. – Мн.: Выш. шк., 1990. – 224 с.
7. Галявов И. Borland С++ для себя. – М.: ДМК Пресс, 2001. – 432 с.
8. Керниган Б., Ричи Д. Язык С.
9. Лаптев В. С++. Экспресс курс. СПб.: БХВ-Петербург, 2004. - 512 с.
10. Липпман. С++ для начинающих.
11. Франка П. С++: учебный курс. — СПб.: Питер, 2003. — 521 с.
12. Культин Н. С/С++ в задачах и примерах. — СПб.: БХВ-Петербург, 2005. - 288 с.
13. Каррано Ф., Причард Дж. Абстракция данных и решение задач на С+-I-. Стены и зеркала, - издание. : Пер. с англ. — М.: Издательский дом "Вильяме", 2003. — 848 с .
14. Штерн В. Основы С++: Методы программной инженерии.
15. Дж. Сик, Л. Ли, Э. Ламсдэйн. С++ Boost Graph Library. Библиотека программиста / Пер. английского Сузи Р. - СПб.: Питер, 2006. — 304 с.
16. Элджер Д. С++ .
17. Якушев Д. «Философия» программирования на языке С++. - М.: Бук_пресс, 2006. — 320 с.
18. Рассохин Д. От Си к Си++.
19. Романов Е. Практикум по программированию на С++: Уч. пособие. СПб: БХВ-Петербург, 2004. - 432 с.
20. Седжвик Р. Фундаментальные алгоритмы на С++. Анализ/Структуры данных/Сортировка/ Поиск: Пер. с англ./Роберт Седжвик. - К.: Издательство «ДиаСофт», 2001.- 688 с.
21. Прайс дж., Гандерлой М. Visual C#.NET .
22. O'Reilly. Programming C# .
23. Г. Дейтел. Введение в операционные системы. В 2-х томах. Пер. с англ. - М.: Мир, 1987.
24. Э. Таненбаум. Современные операционные системы. - СПб.: Питер, 2002. - 1040 с.: ил.
25. E.Butow, T. Ryan. Your visual blueprint for building .NET applications.
26. Harold Davis. Visual C# .NET Programming.
27. A. Hejlsberg, S.Wiltamuth. C# Language Reference.
28. Г. Шилдт. Полный справочник по С#.

რედაქტორი

გადაეცა წარმოებას 15.11.2018. ხელმოწერილია დასაბეჭდად 20.01.2018. ქალაქის ზომა 60X84 1/8. პირობითი ნაბეჭდი თაბახი 13. ტირაჟი 50 ეგზ.

საგამომცემლო სახლი "ტექნიკური უნივერსიტეტი", თბილისი, კოსტავას 77



Verba volant,
scripta manent