ნაკადური შეტანა-გამოტანის შესწავლისათვის C++-ში

მადონა ჯღარკავა
ივ. ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი

ანოტაცია

C++-ს გააჩნია საკმარისად განვითარებული, მოქნილი და სრულყოფილი შეტანა-გამოტანის სისტემა. ობიექტ-ორიენტირებულ შეტანა-გამოტანაში კონსოლური და ფაილური შეტანა-გამოტანა დიდად არ განსხვავდება ერთმანეთისაგან. სტატიაში განიხილება შესაძლებლობათა კომპლექსი, რომელიც საკმარისია იმისათვის, რომ განვახორციელოთ შეტანა-გამოტანის ყველა, ზოგადად მიღებული ოპერაციები. სტატიის მაგალითებში გამოყენებულია შეტანა-გამოტანა კონსოლზე(ჩვენ შემთხვევაში მონიტორის ეკრანზე) და ასევე ფაილში.

სტატია მოიცავს C++-ის შეტანა-გამოტანის სისტემის რამოდენიმე ასპექტს, ფორმატირებული შეტანა-გამოტანისა და მისი მანიპულატორების ჩათვლით.

C++-ში შეტანა-გამოტანის, სტატიაში წარმოდგენილი სისტემა, განსაზღვრულია სტანდარტში Standard C++ და იგი C++-ის არსებული კომპილატორების უდიდეს ნაწილთან თავსებადია. თუ თქვენი კომპილატორი არაა თავსებადი თანამედროვე სისტემასთან, სტატიაში აღწერილი ყველა შესაძლებლობანი, თქვენთვის ხელმისაწვდომი ვერ იქნება.

შესავალი

C++ - სს გააჩნია საკმარისად განვითარებული, მოქნილი და სრულყოფილი შეტანა-გამოტანის სისტემა. ობიექტ-ორიენტირებულ პროგრამირებაში, კონსოლური და ფაილური შეტანა-გამოტანა დიდად არ განსხვავდება ერთმანეთისაგან. სტატიაში განიხილება შესაძლებლობათა კომპლექსი, რომელიც საკმარისია იმისათვის, რომ განვახორციელოთ შეტანა-გამოტანის ყველა, ზოგადად მიღებული ოპერაციები. სტატიის მაგალითებში გამოყენებულია შეტანა-გამოტანა კონსოლზე (ჩვენ შემთხვევაში მონიტორის ეკრანზე) და ასევე ფაილში.

სტატია მოიცავს C++ - ის შეტანა-გამოტანის სისტემის რამოდენიმე ასპექტს, ფორმატირებული შეტანა-გამოტანისა და მისი მანიპულატორების ჩათვლით.

პროგრამირების ნებისმიერი ენის ქვაკუთხედია ინფორმაციის შეტანა გამოტანა. მე ჩემ კოლეგებთან ერთად ვამზადებ დამხმარე სახელმძღვანელოს C++ ენაზე პროგრამირების შესასწავლად, როგორც სტუდენტებისათვის ასევე მასწავლებლებისათვის. პირველი ნაწილი უკვე მზადაა. ვფიქრობ ეს სტატია გამოადგება ყველას, ვინც გადაწყვეტს ნებისმიერი სირთულის პროგრამის დაწერას აღნიშნულ ენაზე. მიმაჩნია C++ საკმარისად რთული ენაა. სხვადასხვა ენაზე პროგრამირებაში ჩემი მრავალწლიანი გამოცდილებით ვეცდები ისე მივაწოდო მასალა, რომ შედარებით გავუმარტივო დამწყებებს და ადვილად გადავიყვანო ახალი სტილის რელსებზე ძველი სპეციალისტები.

ნაკადი მომხმარებლის ინფორმაციის მიმღები და გადამცემი ლოგიკური მოწყობილობაა. C++-ის შეტანა-გამოტანის სისტემის საშუალებით იგი დაკავშირებულია ფიზიკურ მოწყობილობასთან. მიუხედავად იმისა, რომ პროგრამისტს სხვადასხვა მახასიათებლების მქონე მოწყობილობებთან უხდება მუშაობა, იმის გამო, რომ შეტანა-გამოტანის ყველა ნაკადი ერთნაირად მოქმედებს, ყველა მათგანისათვის,

სისტემა გვთავაზობს ერთიან მოხერხებულ ინტერფეისს. მაგ. მონიტორის ეკრანზე ინფორმაციის ჩამწერი ფუნქცია შეგვიძლია გამოვიყენოთ როგორც ფაილში ჩასაწერად, ასევე პრინტერზე ინფორმაციის გამოსატანად.

C++- ზე პროგრამის გაშვებისას ავტომატურად იხსნება 4 ნაკადი.

ნაკადი	მნიშვნელობა	მოწყობილობა შეთანხმებით
cin cout cerr clog	სტანდარტული შეტანა სტანდარტული სტანდარტული შეცდომა cerr- ის ბუფერიზებული ვერსია	კლავიატურა ეკრანი ეკრანი ეკრანი

შეთანხმებით სტანდარტული ნაკადი დაკავშირებულია კლავიატურასთან და ეკრანთან, თუმცა შესაძლებელია მათი Gგადართვა სხვა მოწყობილობებზე.

C++- ში შეტანა-გამოტანის შესასრულებლად, პროგრამაში უნდა ჩავრთოთ სათაური ფაილი <Iostream>. ამ ფაილში განსაზღვრულია კლასთა რთული იერარქია. კლასთა იერარქია, რომელთანაც ჩვენ ხშირად გვექნება ურთიერთობა კლასი basic_ios წარმოებული კლასებია. ios- კლასი წევრებად შეიცავს ფუნქციებს და ცვლადებს, რითაც შეიძლება ვაკონტროლოთ ნაკადის შეტანა-გამოტანის ძირითადი ოპერაციები. ჩვენ ხშირად გამოვიყენებთ ios-კლასს და გვახსოვდეს ამისათვის აუცილებელია პროგრამაში <iostream> სათაურის ჩართვა.

C++-ში შესაძლებელია შეტანა-გამოტანა, როგორც ”დაბალ” ისე "მაღალ" დონეზე. "დაბალ" დონეზე ინფორმაციის შეტანა-გამოტანისას (უფორმატო შეტანა-გამოტანა) ყოველი ბაიტი დამოუკიდებელია და შესაძლებელია დიდი რაოდენობის ნაკადის, დიდი სიჩქარით გადაგზავნა, მაგრამ პროგრამისტებისათვის ასეთი გადაცემა მოუხერხებელია.

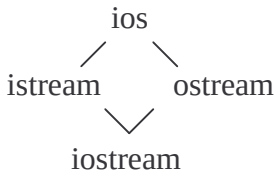
პროგრამისტები უპირატესობას, ფორმატირებულ შეტანა-გამოტანას ანიჭებენ. როცა ბაიტები შეკრებილია ისეთ ნიშნად ელემენტებად, როგორიცაა მთელი და მცოცავმძიმიანი რიცხვები, სიმბოლოები, სტრიქონები და თვით მომხმარებლის მიერ შემოტანილი მონაცემთა ტიპები. შეტანა-გამოტანის ოპერაციის შესასრულებლად iostream ბიბლიოთეკას გააჩნია უამრავი საშუალებანი.

ამ ბიბლიოთეკის ინტერფეისი შედგება რამოდენიმე სათაური ფაილისაგან. iostream-ის ობიექტებია : cin, cout, cerr, clog, რომელიც შესაბამისად შეტანის, გამოტანის, შეცდომების შეტყობნების არაბუფერიზებულ და ბუფერიზებულ სტანდარტულ ნაკადებს წარმოადგენს. გათვალისწინებულია, როგორც ფორმატირებული, ისე არაფორმატირებული შეტანა-გამოტანის შესაძლებლობანი. ფორმატირებული შეტანა-გამოტანის მანიპულატორებით სარგებლობისათვის, პროგრამაში აუცილებელია <iomanip> სათაური ფაილის ჩართვა.

<fstream> სათაური ფაილი შეიცავს მომხმარებლის მიერ ფაილებზე ოპერაციების ჩასატარებლად საჭირო ინფორმაციას.

iostream ბიბლიოთეკა შეიცავს ბევრ კლასს. კლას istream –ში შედის ოპერაციები ნაკადის შეტანისათვის, ostream კლასში გამოტანისათვის, ხოლო iostream მოიცავს ნაკადის როგორც შეტანის, ისე გამოტანის ოპერაციებს.

istream და ostream კლასები წარმოადგენს ios ბაზური კლასის პირდაპირ მემკვიდრეს. iostream კი istream და ostream –ის მრავალჯერად მემკვიდრედ ითვლება. ამ კლასების მემკვიდრეობითობა ასე გამოიხატება:



შეტანა-გამოტანის ოპერაციათა მოხერხებულ ჩაწერას უზრუნველყოფს ოპერაციათა გადატვირთვა. მარცხნივ ძვრის ოპერაცია (<<) გადატვირთულია ნაკადის გამოსატანად და მას ეწოდება ”ნაკადში მოთავსების" ოპერაცია. მარჯვნივ ძვრის ოპერაცია (>>) გადატვირთულია მეხსიერებაში შესატანად და ქვია "ნაკადიდან ამოღება". ეს ოპერაციები გამოიყენება სტანდარტული ნაკადების ობიექტების მიმართ: cin, cout, cerr და clog. ეს ოპერაციები ასევე გამოიყენება, მომხმარებლის მიერ განსაზღვრული ობიექტების მიმართაც.

istream კლასის, სტანდარტული ნაკადის შეტანის ობიექტი cin, როგორც მიღებულია, მიმაგრებულია შეტანის სტანდარტულ მოწყობილობასთან-კლავიატურასთან.

შემდეგი ჩანაწერი

cin>>grade // მონაცემი მოძრაობს ნაკადში მარჯვნივ.

"ამოვიღოთ ნაკადიდან ", ნიშნავს, რომ მთელი ტიპის - grade მნიშვნელობა cin ობიექტიდან გადადის მეხსიერებაში (თუ ჩავთვლით, რომ ცვლადი grade გამოცხადებულია როგორც int).

აქვე შევნიშნოთ, რომ "ნაკადიდან ამოღების" ოპერაცია "საკმარისად ინტელექტუალურია", იმისათვის, რომ განსაზღვროს გამოყენებული მონაცემების ტიპი. თუ ცვლადი ამოცანის მოთხოვნისამებრ იყო გამოცხადებული, ნაკადიდან მისი ამოღებისას არავითარი დამატებითი ინფორმაცია აღარაა საჭირო.

ostream კლასის, სტანდარტული ნაკადის გამოტანის ობიექტი cout, როგორც წესი მიმაგრებულია გამოტანის მოწყობილობასთან, ჩვეულებრივ დისპლეის ეკრანთან. შემდეგი ჩანაწერი

cout<< grade // მონაცემი მოძრაობს ნაკადში მარცხნივ.

"მოვათავსოთ ნაკადში", ნიშნავს, რომ მთელი ტიპის - grade მნიშვნელობა გამოვიტანოთ მეხსიერებიდან, გამოტანის სტანდარტულ მოწყობილობაზე.

"ნაკადში მოთავსების" ოპერაციაც " საკმარისად ჭკვიანია", იმისათვის, რომ განსაზღვროს grade ცვლადის ტიპი. (იგულისხმება, რომ მისი ტიპი გამოცხადებულია ამოცანის მოთხოვნისამებრ). დამატებითი ინფორმაცია საჭირო აღარაა.

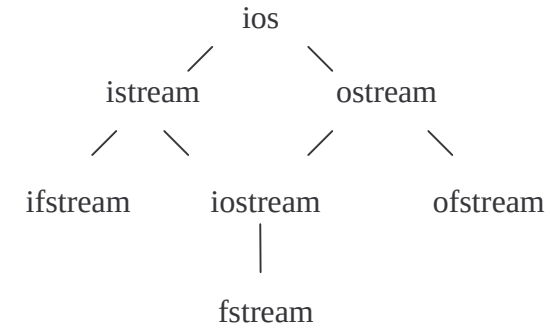
ostream კლასის cerr ობიექტი მიმაგრებულია შეცდომათა შეტყობინების გამოტანის მოწყობილობასთან. cerr ობიექტისათვის მონაცემთა გამომავალი ნაკადი არაა ბუფერიზებული. ეს ნიშნავს, რომ, ყოველი ოპერაცია "მოვათავსოთ cerr-ში", მომენტალურად გვაძლევს შეტყობინებას და მომხმარებელი დროულად და საჭიროებისამებრაა ინფორმირებული, სისტემაში არსებულ ხარვეზებზე.

ostream კლასის ობიექტი clog, ასევე მიმაგრებულია შეცდომების შეტყობინებათა სტანდარტულ მოწყობილობასთან. მონაცემთა გამომავალი ნაკადი ბუფერიზებულია: ეს ნიშნავს, რომ გამოტანა ინახება ბუფერში, მანამ სანამ ის არ გაივსება ან ჩვენ თვითონ არ მოვითხოვთ მის გამოტანას.

ფაილების დამუშავებისას c++-ში გამოიყენება შემდეგი კლასები:

ifstream- ფაილიდან მეხსიერებაში შეტანისათვის ოპერაციები.
ofstream- მეხსიერებიდან ფაილში გამოტანის ოპერაციები.
fstream- შეტანა-გამოტანის ოპერაციები ფაილებში.

ifstream istream -ის მემკვიდრეა, ofstream ostream -ის, ხოლო fstream iostream-ის.



ფორმატირებული შეტანა-გამოტანა.

C++-ში შესაძლებელია ინფორმაცია გამოვიტანოთ სხვადასხვა ფორმით და ასევე შევცვალოთ მისი გარკვეული პარამეტრები შეტანის დროს.

შეტანა-გამოტანის ყოველი ნაკადი დაკავშირებულია ფორმატის ალმების სიმრავლესთან. თითოეული ალამი ემსახურება ინფორმაციის ფორმატირებას. ალამი წარმოადგენს ios -კლასის გადათვლადი ტიპის fmtflags -ის კონსტანტას შემდეგი მნიშვნელობებით:

adjustifield	floatfield	right	skipws	basefield	hex
scientific	unitbuf	boolalpha	dec	fixed	internal
showbase	uppercase	left	showpoint	oct	showpos

ფორმატის ალამის შერჩევა ხდება ფუნქციით setf(). იგი წარმოადგენს ios – კლასის წევრ ფუნქციას. მისი ზოგადი სახე ასეთია:

```
fmtflags setf (fmtflags flags);
```

ეს ფუნქცია აბრუნებს ფორმატის ადრე შერჩეულ ალმებს და არჩევს ახლებს ისე, როგორც იქნება მითითებული ცვლად flags-ში. (ის ალამი, რომელიც აქ არ მოიხსენიება არ იცვლება). მაგ. თუ გვინდა ავარჩიოთ showpos , უნდა ჩავწეროთ:

```
შეტანა/გამოტანის_ნაკადი.setf (ios::showpos);
```

:: დანახვის არეს გაფართოების ოპერატორი აქ აუცილებელია, რადგან showpos წარმოადგენს ios კლასის გადათვლად კონსტანტას. ამ ოპერატორის გარეშე showpos მიუწვდომელია. ფუნქცია setf(), ios- კლასის წევრი ფუნქციაა და იგი მოქმედებს ამ

კლასის მიერ შექმნილ შეტანა- გამოტანის ნაკადზე. ამის გამო ფუნქციის გამოძახება ხდება კონკრეტულ ნაკადთან კავშირში და არა ზოგადად. ყოველ ნაკადს გააჩნია ინფორმაცია საკუთარი ფორმატის მდგომარეობის შესახებ.

setf() ფუნქციის ერთი გამოძახებით შესაძლებელია რამოდენიმე ალამის ერთდროული შერჩევა. სხვადასხვა ალამი ერთმანეთს (OR) (|) ოპერატორით უკავშირდება. მაგ. setf() ფუნქციის შემდგომი გამოძახებით ერთდროულად ვარჩევთ showbase და hex ალამს cout ნაკადისათვის.

cout.setf (ios::showbase | ios::hex);

ფორმატის ალმების ჩამოსაყრელად არსებობს setf() ფუნქციის დამატება, ფუნქცია unsetf().

შემდეგ მაგალითში ნაჩვენებია ფორმატის სხვადასხვა ალმების შერჩევა:

```
1.  #include <iostream>
2.  using namespace std;

3.  int main()
4.  {
5.  //შეთანხმებით ინფორმაცია ასე გამოიტანება
6.  cout << 123.23  << " hello  "  << 100 <<'\n';
7.  cout << 10 << ' ' << -10 << '\n';
8.  cout << 100.9  << '\n';
9.  //ფორმატის შეცვლა
10. cout.unsetf (ios::dec);
11. cout.setf( ios::hex  |  ios::scientific  );
12. cout<< 123.23  <<  " hello "  << 100  <<  '\n';
13. cout.setf ( ios:: showpos );
14. cout << 10  << ' ' << -10  << '\n';
15. cout.setf ( ios::showpoint  |  ios::fixed );
16. cout << 100.0;
17. return  0;
18. }
```

პროგრამის შესრულების შემდეგ ეკრანზე მივიღებთ:

```
        123.23      hello  100
    10      -10
100
1.232300e+02   hello  64
a  fffffff6
+100.000000
```

შევნიშნოთ, რომ showpos მხოლოდ ათობითი მნიშვნელობების გამოტანაზე მოქმედებს. თუ რიცხვი თექვსმეტობითში გამოგვყავს, ეს ალამი არ მოქმედებს.

ნაკადების გამოტანა

ostream კლასი უზრუნველყოფს ნაკადის ფორმატირებულ და არაფორმატირებულ გამოტანას, ესენია: "ნაკადში მოთავსების " ოპერაციით სტანდარტული ტიპების გამოტანა; put წევრი-ფუნქციით სიმბოლოთა გამოტანა; write წევრი-ფუნქციით არაფორმატირებული გამოტანა; მთელი რიცხვების ოროობით, რვაობით და თექვსმეტობით ფორმატში გამოტანა; სხვადასხვა სიზუსტის მცოცავმძიმიანი მნიშვნელობების გამოტანა: ან ათობითი წერტილის მითითებით ან ექსპონენციალური

ფორმატით ან ფიქსირებული მძიმით; გამოსატან ველში სასურველ ნაპირთან მიჯრით გამოტანა; შერჩეული სიმბოლოებით ველის ცარიელი ადგილების შევსება; ექსპონენციალურ ან თექვსმეტობით ფორმატში ზედა რეგისტრზე ასოების გამოჩენა.

ნაკადში გამოტანა ხორციელდება გადატვირთული ოპერაციით (<<) "ნაკადში მოთავსება". . ქვემოთ მოგვყავს პროგრამა ამ ოპერაციის გამოყენებით:

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5. cout<<" welcome to c++! \n ";
6. return 0;
7. }
```

ახალ სტრიქონზე გადასვლა ხდება '\n' მიმდევრობით (escape მიმდევრობით) ან endl მანიპულატორით. ნაკადის მანიპულატორს endl გადავყავართ ახალ სტრიქონზე და ამავე დროს ის ასუფთავებს გამოტანის ბუფერს (მაშინაც კი, როცა ბუფერი ჯერ არაა სავსე, იგი ჩამოიყრება). გამოტანის ბუფერის გასუფთავება ხდება ასევე ოპერატორით flush

```
cout<<flush;
```

"ნაკადში მოთავსების" ოპერატორი შეიძლება იყოს გადაჯაჭვული და ამ დროს ის სრულდება მარცხნიდან მარჯვნივ. მაგ.

```
cout<< "summa a b=" << "a+b" << "=" << a+b << endl;
```

როცა a=5 და b =9 გვექნება

```
summa a b= a+b = 14
```

A char * ტიპის ცვლადების გამოტანა

C-ენაში პროგრამირებისას, მონაცემთა შეტანა-გამოტანისათვის, პროგრამისტი იძლევა ინფორმაციას ამ მონაცემთა ტიპების შესახებ. c++-ში, ტიპები ავტომატურად განისაზღვრება და ეს ითვლება ამ ენის მნიშვნელოვან მიღწევად. მაგრამ ეს ზოგჯერ ართულებს საქმეს. მაგ. სიმბოლური სტრიქონის ტიპი არის char *. ვთქვათ გვინდა ამ მიმთითებლის მნიშვნელობის გამოტანა, ე.ი. გვინდა, ამ სტრიქონის პირველი სიმბოლოს მისამართი მეხსიერებაში. მაგრამ (<<) -ეს ოპერაცია გადატვირთულია 0-ით დამთავრებული სტრიქონების გამოსატანად. ამ ამოცანის გადაწყვეტისათვის მიმთითებლის ტიპი უნდა გადავიყვანოთ void * ტიპზე (ასე უნდა მოვიქცეთ ცვლადზე ნებისმიერ მიმთითებელზე, თუ პროგრამისტს უნდა მისამართის გამოტანა). ქვემოთ მოყვანილ პროგრამას გამოყავს ცვლადი char * როგორც სტრიქონი და როგორც მისამართი. გვახსოვდეს, რომ მისამართი გამოიტანება Qთექვსმეტობით სისტემაში. c++-ში თექვსმეტობითი რიცხვები იწყება 0x ან 0X –ით.

```
1. #include <iostream>
2. using namespace std;
3. int main()
4. {
5. char *string = "test";
6. cout<< " string ცვლადის მნიშვნელობა : "<< string
7. <<"\n string ცვლადის მისამართი: "
8. <<static_cast <void * > (string) <<endl;
9. return 0;
10. }
```

წევრი-ფუნქცია put;

წევრ-ფუნქციას put გამოაქვს ერთი სიმბოლო. მაგ. cout.put('A') ; გამოაქვს სიმბოლო A; შესაძლებელია ამათი გადაჯაჭვაც. მაგ. cout.put('A').put('\n'); ოპერატორი წერტილი სრულდება მარცხნიდან მარჯვნივ, წევრი-ფუნქცია put აბრუნებს მიმართვას ობიექტზე, რომელმაც ის გამოიძახა(ე.ი cout) – ჯერ გამოიტანს სიმბოლოს A, ისევ ბრუნდება cout-ზე და გადავა ახალ სტრიქონზე.

ნაკადების შეტანა

გადატვირთული ოპერაცია (>>) ნაკადიდან აღება, იგნორირებას უკეთებს გამყოფ სიმბოლოებს (ხარვეზი, ტაბულაცია, ახალ სტრიქონზე გადასვლა). მთელი რიცხვების წასაკითხად გამოვიყენებთ ობიექტს cin და გადატვირთულ ოპერაციას (>>) "ნაკადიდან აღება". გვახსოვდეს, რომ შესაძლებელია ამ ოპერაციების გადაჯაჭვა. მაგ.

cin>>x>>y;

"ნაკადიდან ამოღების " ოპერაცია იღებს ნულოვან მნიშვნელობას (false), თუ ნაკადში შეხვდა ფაილის ბოლოს ნიშანი, წინააღმდეგ შემთხვევაში იგი მიუთითებს ობიექტზე, რომელმაც იგი გამოიძახა.

გადატვირთვის ოპერაციებს (>>) და (<<) აქვს მაღალი პრიორიტეტი, ამიტომ რაიმე გამოსახულების ჩაწერისას, უნდა გამოვიყენოთ ფრჩხილები.

მაგ. ამ პროგრამაში ფრჩხილები აუცილებელია.

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.  int a;int b;
6.  cout<<"a da b=";
7.  cin>>a;
8.  cin>>b;
9.  cout<<a<<"  "  <<(a==b ? " ": "no    ")
10. <<"equel  "<< b << endl;
11. return 0;
12. }
```

"ნაკადიდან ამოღების " ოპერაციის გამოყენებით, წავიკითხოთ მთელი რიცხვები და ავარჩიოთ მათ შორის მაქსიმალური. ვიცით, ეს (>>) ოპერაცია იღებს 0-ან(false) მნიშვნელობას, თუ ნაკადში შეხვდება ფაილის ბოლოს ნიშანი. პროგრამა დაწერილია ამ პირობით, ციკლის ოპერატორში while და პოულობს მაქსიმალურს-max წაკითხულ მთელ რიცხვებს, grade-ს შორის.

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.  int grade,max=-1;
6.  cout<<" grade =";
7.  while (cin>>grade ){
if (grade>max)
        max=grade;
8.  cout<<" grade =";
```

```
9.   }
10.  cout<<"max="<<max<<endl;
11.  return 0;
12.  }
```

წევრი-ფუნქციები get და getline

წევრი-ფუნქცია get, არგუმენტის გარეშე, მითითებული ნაკადიდან კითხულობს ერთ სიმბოლოს (მაშინაც კი, როცა ეს სიმბოლო გამყოფია) და აბრუნებს მას ფუნქციის მნიშვნელობის სახით. თუ ნაკადში შეხვდება ფაილის ბოლოს ნიშანი, ფუნქცია აბრუნებს EOF-ს.E

ქვემოთ მოგვყავს პროგრამა, რომელსაც შეტანის cin ნაკადის eof და get წევრი-ფუნქციებით და გამოტანის cout ნაკადის put წევრი-ფუნქციის გამოყენებით შეყავს და გამოყავს სიმბოლოები. თავდაპირველად პროგრამა გამოიტანს cin.eof() –ის მნიშვნელობას ე.ი. false(0), იმის საჩვენებლად რომ, ნაკადში ჯერ ფაილის ბოლო არ არის. მომხმარებელს შეყავს სტრიქონი, რომელიც თავდება ფაილის ბოლოთი (<ctrl>z). პროგრამა კითხულობს ყოველ სიმბოლოს და გამოყავს cout–ში print–ის გამოყენებით. როცა გამოჩნდება ფაილის ბოლოს ნიშანი, while ციკლი მთავრდება და პროგრამა გამოიტანს ისევ cin.eof()- ის ნმიშვნელობას, რომელიც უკვე არის true (1).

```
1.  #include <iostream>
2.  using  namespace std;
3.  int main()
4.  {
5.  char c;
6.  cout<< " cin.eof()    შეტანამდე:  "  <<cin.eof()
7.  <<"\n შემოიტანეთ სტრიქონი   :  \";
8.  while  ( (  c=cin.get()    )  != EOF   )
9.  cout.put(   c   );
10. cout<<"\n EOF ამ სისტემაში   :   "  << c;
11. cout  <<  "\n  cin.eof()   შეტანის შემდეგ:  "  <<  cin.eof()  << endl;
12. return 0;    }
```

სიმბოლური არგუმენტის მქონე, get წევრი-ფუნქციის მეორე ვარიანტი კითხულობს შემავალი ნაკადიდან მომდევნო სიმბოლოს (მათ შორის გამყოფსაც) და ინახავს არგუმენტში. ფაილის ბოლოს შემთხვევაში, ფუნქციის მნიშვნელობა მცდარია. სხვა შემთხვევებში ეს ფუნქცია მიუთითებს istream კლასის იმ ობიექტზე, რომლისთვისაც იყო გამოძახებული.

get ფუნქციის მესამე ვარიანტი იღებს სამ პარამეტრს: სიმბოლური მასივი, სიმბოლოთა მაქსიმალური მნიშვნელობა და შემზღუდავი სიმბოლო, შეთანხმებით ეს '\n' –ახალ ხაზზე გადასვლაა. ამ შემთხვევაში შემავალი ნაკადიდან ვკითხულობთ მაქსიმალურ მნიშვნელობაზე 1-ით ნაკლებ სიმბოლოთა რაოდენობას, ან ვიდრე შემზღუდავ სიმბოლომდე. შეტანილი სტრიქონის დამთავრებისთვის სიმბოლურ მასივში ჩაიწერება 0, შემზღუდავი სიმბოლო რჩება შემავალ ნაკადში და მასივში არ ჩაიწერება. (ის იქნება შემდეგი წასაკითხი სიმბოლო). ამ შემთხვევაში ფუნქციის ზედიზედ გამოყენებისას რეზულტატი იქნება ცარიელი სტრიქონი, თუ შემზღუდავს არ მოვიშორებთ შემავალი ნაკადიდან. ქვემოთ მოყვანილი პროგრამა ადარებს შეტანას cin-ისთვის, "ნაკადიდან აღების" ოპერაციით (<<) (როცა სიმბოლოები იკითხება პირველ

გამყოფამდე) , და cin.get-ით. cin.get-ზე მიმართვისას შემზღუდავ სიმბოლოდ იგულისხმება "\n", ის ცხადად არ მოიცემა.

```
1.  #include <iostream>
2.  using namespace std;

3.  int main()
4.  {
5.  const int SIZE=80;
6.  char buffer1 [ SIZE ], buffer2 [SIZE ];

7.  cout << " შემოიტანეთ სტრიქონი :  \n";
8.  cin>>buffer1;
9.  cout<< " \n  cin-ით წაკითხული სტრიქონი  : \n"
10. <<buffer1 << "\n\n";

11. cin.get( buffer2, SIZE );
12. cout << "  cin.get-ით წაკითხული სტრიქონი  :\n"
13. << buffer2 <<  endl;

14. return 0;
15. }
```

წევრი-ფუნქცია getline ისევე მოქმედებს, როგორც get ფუნქციის მესამე ვარიანტი და ათავსებს 0-ს სტრიქონის ბოლოში, სიმბოლურ მასივში. განსხვავებით get- სგან getline შემზღუდავ სიმბოლოს შლის ნაკადიდან (ე.ი. კითხულობს მას და აგდებს). იგი არ ინახება სიმბოლურ მასივში.

ქვემოთ მოყვანილია პროგრამა, რომელიც getline წევრი-ფუნქციის გამოყენებით კითხულობს ტექსტის სტრიქონს.

```
1.  #include <iostream>
2.  using namespace std;

3.  int main()
4.  {
5.  const SIZE = 80;
6.  char buffer[ SIZE ];

7.  cout << " შემოიტანეთ  ტექსტი : \n";
8.  cin.getline( buffer, SIZE );

9.  cout<< " \ n  შემოტანილი ტექსტი  : \n"  <<  buffer  <<  endl;
10. return 0;
11. }
```

istream კლასის წევრი-ფუნქციები: peek, putback, ignore.

ფუნქცია-წევრი ignore ტოვებს სიმბოლოთა მოცემულ რაოდენობას(შეთანხმებით ერთ სიმბოლოს) ან ამთავრებს თავის მუშაობას მითითებულ შემზღუდველამდე ნაკადში. (შეთანხმებით შემზღუდველად ითვლება EEOF, რომლის გავლენითაც

მოცემული ფუნქცია ფაილიდან კითხვისას, გამოტოვებს ყველა სიმბოლოს ფაილის ბოლომდე).

წევრი-ფუნქცია putback შემავალი ნაკადიდან წევრი-ფუნქცია get –ის საშუალებით მიღებულ წინა სიმბოლოს აბრუნებს უკან იმავე ნაკადში. ფუნქცია სასარგებლოა ისეთი პროგრამებისათვის, რომელიც ეძებს მოცემული სიმბოლოთი დაწყებულ ჩანაწერებს. როცა ასეთ სიმბოლოს ნახავს, პროგრამა დააბრუნებს მას ნაკადში და ეს სიმბოლო აღმოჩნდება შესატან მონაცემებში.

წევრი-ფუნქცია peek შემავალი ნაკადიდან აბრუნებს მომდევნო სიმბოლოს, მაგრამ მას ნაკადში ტოვებს.

არაფორმატირებული შეტანა-გამოტანა read, gcount და write- ის გამოყენებით.

არაფორმატირებული შეტანა-გამოტანა სრულდება read() და write() წევრი-ფუნქციების გამოყენებით. თითოეულ მათგანს შეყავსE ან გამოყავს ბაიტების რაღაც რაოდენობა მეხსიერების სიმბოლურ მასივში ან მისგან. ეს ბაიტები არანაირ ფორმატირებას არ განიცდის. მაგალითად

cout.write("abcdefgyijklmnopqrstuvwxyz",10);

გამოიტანს ეკრანზე ალფავიტის პირველ 10 სიმბოლოს.

წევრ ფუნქციას read() – შეყავს მითითებული რაოდენობის სიმბოლოები, სიმბოლურ მასივში. წაკითხულ სიმბოლოთა ნაკლებ რაოდენობაზე მიუთითებს ალამი failbit.

წევრი-ფუნქცია gcount() იძლევა, შეტანის ბოლო ოპერაციისას შეტანილი სიმბოლოების რაოდენობას.

მოვიყვანოთ პროგრამა, istream კლასის read() და gcount() წევრი-ფუნქციებისა და ostream კლასის write() წევრი- ფუნქციის მუშაობის საილუსტრაციოდ. პროგრამას შეყავს სიმბოლოთა მასივში buffer, 20 სიმბოლო(შემავალი მიმდევრობა უფრო გრძელია)წევრი-ფუნქციით read(). gcount()-ით განსაზღვრავს შეტანილ სიმბოლოთა რაოდენობას და გამოყავს სიმბოლოთა მასივში -buffer წევრი-ფუნქცია write() –ის გამოყენებით.

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.  const  nt SIZE=80;
6.  char buffer [ SIZE ];
7.  cout << " შემოიტანეთ სტრიქონი  :  \n";
8.  cin.read( buffer,20);
9.  cout <<" \n შემოტანილი სტრიქონი  :  \n";
10. cout.write( buffer, cin.gcount()  );
11. cout << endl;
12. return  0;
13. }
```

ნაკადების მანიპულატორები

ფორმატირების ამოცანა C++-ში გადაწყვეტილია ნაკადების მანიპულატორების გამოყენებით. მათი საშუალებით შესაძლებელია შემდეგი ოპერაციები: ველის სიგანის შერჩევა, სიზუსტის შერჩევა, ფორმატის ალმების დაყენება ან მოხსნა, ველის შესავსები სიმბოლოების არჩევა, ნაკადის მოხსნა, ახალი სტრიქონის სიმბოლოს ჩასმა გამომავალ

ნაკადში და ნაკადის მოხსნა, გამომავალ ნაკადში ნულოვანი სიმბოლოს ჩასმა და შემავალ ნაკადში გამყოფი სიმბოლოების გამოტოვება.

რიცხვებისათვის ფუძის ამრჩევი მანიპულატორები ნაკადისათვის
dec, oct, hex, setbase.

როგორც წესი, მთელი რიცხვები წარმოიდგინება 10-ით სისტემაში. თუ გვინდა ვიმუშაოთ 16-ობით სისტემაში, ნაკადში უნდა მივუთითოთ მანიპულატორი hex, 8-ობითი წარმოდგენისათვის საჭიროა ჩავწეროთ oct, თუ გვინდა დავუბრუნდეთ ნაკადის ათობით წარმოდგენას ვწერთ dec. ნაკადის ფუძე შეიძლება შეიცვალოს პარამეტრიანი მანიპულატორითაც - setbase , რომელსაც აქვს ერთი მთელი პარამეტრი , რიცხვითი სისტემის ფუძეების შესაბამისად, მნიშნელობებით 10,8, ან 16. ასეთ მანიპულატორს ეწოდება პარამეტრიზებული. ნებისმიერი პარამეტრიზებული მანიპულატორის გამოყენებისას პროგრამაში უნდა ჩავრთოთ სათაური ფაილი <iomanip>. ნაკადის ფუძე რჩება უცვლელი მანამ, სანამ თავიდან არ შევარჩევთ.

მოვიყვანოთ მანიპულატორების გამოყენებით პროგრამის მაგალითი:

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.  int n;
6.  cout<<" შემოიტანეთ ათობითი რიცხვი:  ";
7.  cin>>n;
8.  cout<< n << "  თექვსმეტობით ფორმატში :  "
9.  << hex  << n<< '\n'
10. <<dec  << n  << "  რვაობით ფორმატში  :  "
11. << oct <<  n << '\n'
12. <<setbase ( 10 ) << n  << "  ათობით ფორმატში  :  "
13. << n <<endl;
14. return 0;
15. }
```

მცოცავ მძიმიანი რიცხვების სიზუსტე.
(precision, setprecision)

მცოცავ მძიმიანი რიცხვების სიზუსტე შეგვიძლია ვმართოთ, ათობითი წერტილიდან მარჯვნივ თანრიგების რაოდენობით. ეს შეგვიძლია გავაკეთოთ ნაკადის მანიპულატორის setprecision ან წევრი ფუნქცია precision() გამოყენებით. თითოეული მათგანი მოქმედებს მანამ, სანამ არ მოხდება მათი თავიდან შერჩევა.

```
1.  #include <iostream>
2.  using namespace std;
3.  #include <cmath>
4.  #include <iomanip>
5.  int main()
6.  {
7.  double root2=sqrt(2.0);
8.  int places;
9.  //        cout<<setiosflags(ios::fixed)<<'\n';
10. for ( places=0; places<=9; places++)  {
```

```
            cout.precision(places);
            cout<<root2<<'\n';
11.  }
12.  cout<<endl;
            for ( places=0; places<=9; places++)  {
            cout<<setprecision(places)
            <<root2<<'\n';
            }

13.  return 0;
14.  }
```

ველის სიგანე
(setw,width)

ios კლასის წევრი-ფუნქცია width() არჩევს ველის სიგანეს.(ე.ი. სიმბოლოების იმ პოზიციათა რაოდენობას, სადაც უნდა გამოვიტანოთ მნიშნელობა ან სიმბოლოთა იმ რაოდენობას, რომელიც უნდა შევიტანოთ)და აბრუნებს ველის სიგანის წინა მნიშვნელობას. თუ ველის სიგანე მეტია, ვიდრე სიდიდის გამოსატანად საჭირო ადგილი, ზედმეტი პოზიციები შეივსება ჩვენს მიერ შერჩეული შემვსები სიმბოლოებით. თუ გამოსატანად მეტი პოზიციაა საჭირო, ზედმეტი სიმბოლოები არ დაიკარგება და რიცხვი სრულყოფილად გამოიტანება. ველის სიგანის შერჩევა ეხება ნაკადიდან მარტო მომდევნო ჩადებას ან აღებას. შემდგომ ველი ხდება ისეთი, რომელიც აუცილებელია. ლოგიკურ შეცდომად ითვლება ის წარმოდგენა, თითქოს ერთხელ შერჩეული სიგანე ეხება ყველა მომდევნო გამოტანას. ველის სიგანე ასევე, შეიძლება შევარჩიოთ ნაკადის მანიპულატორით setw.

ქვემოთ მოყვანილია პროგრამა, წევრი-ფუნქცია width –ის გამოყენებით.

```
1.  #include <iostream>
2.  using namespace std;
3.  #include <iomanip>
4.  int main()
5.  {
6.  int w=4;
7.  char string[10];
8.  cout<< " შემოიტანეთ სტრიქონი : \n";
9.  cin.width(5);
10. while  (  cin>> string )  {
          cout.width ( w++ );
          cout<< string  << endl;
          cin.width( 5 );
11. }
12. return 0;
13. }
```

მომხმარებლის მიერ განსაზღვრული მანიპულატორები

მანიპულატორები შეიძლება შევქმნათ ჩვენ თვითონ. ქვემოთ, პროგრამაში ნაჩვენებია მანიპულატორების შექმნის და მისი შემოწმების მაგალითები bell, ret, endline და tab - ისათვის.

```
1.  #include <iostream>
2.  using namespace std;
3.  // bell მანიპულატორი იყენებს   '\a'   escape მიმდევრობას.
4.  ostream& bell(ostream& output) {return output <<   'a';}
5.  //ret -   '\r'   მიმდევრობას
6.  ostream& ret( ostream& output ) {return output <<  'r';}

7.  //endline-   '\n'   და წევრ-ფუნქციას   flush  -ს.
8.  ostream& endline( ostream& output)
9.  {
10. return output <<  '\n'   <<   flush;
11. }
12. // tab – იყენებს   '\t'   escape  მიმდევრობას.
13. ostream& tab( ostream& output ) {return output <<  '\t';
14. }
15. // აქ მოწმდება მომხმარებლისგან შექმნილი მანიპულატორები.
16. int main()
17. {
18. cout << bell<< endl;
19. cout <<  ret  <<  "------------------"  <<endline;
20. cout<<  'a '  <<  tab  <<  'b'   << tab   <<  'c'   <<   endline;
21. return 0;
22. }
```

ნაკადის ფორმატის მდგომარეობა

ფორმატის სხვადასხვა ალამი იძლევა ფორმატირების სხვადასხვა სახეს, რომელიც შესაძლებელია შეტანა-გამოტანის დროს. ალამის შერჩევა ხდება წევრი-ფუნქციებით: setf(), unsetf() და flags().

ფორმატის მდგომარეობების ალმები

ამ ალმების მართვა ხდება წევრი-ფუნქციებით flags(), setf() და unsetf(), მაგრამ ზოგი პროგრამისტი, უპირატესობას ნაკადის მანიპულატორებს ანიჭებს. სხვადასხვა ოპციების გაერთიანებისათვის შეიძლება გამოვიყენოთ ბიტური ოპერაცია (|) - or . setf() ფუნქციის ერთი არგუმენტი იძლევა რამოდენიმე ალამს, რომელიც შეერთებულია (|) ოპერაციით.

ნაკადის პარამეტრიზებული მანიპულატორი setiosflags ასრულებს იგივეს, რასაც წევრი-ფუნქცია setf(), ხოლო მანიპულატორი resetiosflags – იგივეს, რასაც unsetf(). ნებისმიერი მანიპულატორით სარგებლობისთვის, საჭიროა პროგრამაში ჩავრთოთ სათაური <iomanip>.

ალამი skipws, აჩვენებს რომ "ნაკადიდან აღების" ოპერაცია შემავალი ნაკადიდან, გამყოფ სიმბოლოებს აგდებს, გამოტოვებს. შეთანხმებით ოპერაცია (>>) ტოვებს გამყოფ სიმბოლოებს. ეს რომ შევცვალოთ, ე.ი. უარი ვთქვათ ალამზე skipws, უნდა გამოვიძახოთ ფუნქცია unsetf(ios::skipws). არსებობს ნაკადის მანიპულატორი ws, რომლითაც შეგვიძლია მიუთითოთ გამოვტოვოთ თუ არა გამყოფი სიმბოლოები.

ფორმატის მდგომარეობის ალამი	**აღწერა**
ios::skipws	გამყოფი სიმბოლოების გამოტოვება შემავალ ნაკადში

ios::left	გამომავალი მონაცემების მიჯრა ველის მარცხენა მხარეს. აუცილებლობის შემთხვევაში შემვსები სიმბოლოები მოთავსდება მარჯვნივ.
ios::right	გამომავალი მონაცემების მიჯრა ველის მარჯვენა მხარეს. აუცილებლობის შემთხვევაში შემვსები სიმბოლოები მოთავსდება მარცხნივ.
ios::internal	მიუთითებს, რომ რიცხვის ნიშანი მოთავსდება ველის მარცხნივ, რიცხვის სიდიდე, იმავე ველის მარჯვნივ(შემვსები სიმბოლოები ჩაიწერება ნიშანსა და რიცხვს შორის).
ios::dec	განსაზღვრავს, რომ დამუშავება მიმდინარეობს ათობით სისტემაში(ფუძე 10).
ios::oct	განსაზღვრავს, რომ დამუშავება მიმდინარეობს რვაობით სისტემაში(ფუძე 8).
ios::hex	განსაზღვრავს, რომ დამუშავება მიმდინარეობს თექვსმეტობით სისტემაში(ფუძე 16).
ios::showbase	მიუთითებს, რომ რიცხვის ფუძე გამოიტანოს რიცხვის წინ(რვაობითი რიცხვებისათვის საწყისი 0, თექვსმეტობისათვის 0x ან 0X)
ios::showpoint	განსაზღვრავს, რომ მცოცავმძიმიანი რიცხვები გამოიტანოს წერტილით(მძიმით). ეს ძირითადად გამოიყენება ios::fixed - სთვის, წერტილის მარჯვნივ, გარკვეული რაოდენობის ციფრების გამოსატანად.
ios::uppercase	განსაზღვრავს, რომ თექვსმეტობით წარმოდგენაში გამოყენებული იყოს დიდი ასოები და მცოცავი მძიმით რიცხვის ჩაწერისას ექსპონენციალურ ფორმატში გამოვიყენოთ დიდი E.
ios::showpos	განსაზღვრავს, რომ დადებით და უარყოფით რიცხვებს ქონდეთ წინ ნიშანი, შესაბამისად + და -.
ios:: scientific	განსაზღვრავს მცოცავმძიმიანი რიცხვების გამოტანას, ექსპონენციალურ წარმოდგენაში
ios::fixed	განსაზღვრავს ფიქსირებული მძიმით რიცხვების გამოტანას, გარკვეული რაოდენობის ციფრებით, ათობითი წერტილიდან მარჯვნივ.

დაბალი ნულოვანი თანრიგები და ათობითი წერტილები (ios::showpoint)

ალამ showpoint-ს ვიყენებთ, როცა აუცილებელია ათობითი წერტილის და დაბალ თანრიგებში 0-ების გამოტანა (0-ები რიცხვის ბოლოს). ალამის გარეშე რიცხვი 79.0 გამოიტანება, როგორც 79, ხოლო ალამით იქნება 79.000000 (ნულების რაოდენობას განსაზღვრავს სიზუსტე).

ალამი internal მიუთითებს, რომ რიცხვის ნიშანი (ან ფუძე, როცა დაყენებულია ალამი ios::showbase) უნდა მიეჯრას მარცხენა მხარეს, ხოლო შუა ცარიელ ადგილებში

უნდა მოთავსდეს არჩეული შემვსები სიმბოლოები. setf ფუნქცია-წევრის მეორე პარამეტრად უნდა ავირჩიოთ არგუმენტი ios::adjustfield , თუ გვინდა დავაყენოთ ერთ-ერთი ამ ალმებიდან: left, right ან internal. ამით გარანტირებული ვართ, რომ ფუნქცია setf() აირჩევს ერთ-ერთ ამათგანს. (ისინი ერთმანეთის ურთიერთგამომრიცხავებია).

მოვიყვანოთ პატარა მაგალითი:

```
1.  #include <iostream>
2.  using namespace std;
3.  #include <iomanip>

4.  int main()
5.  {
6.  cout<< setioflags( ios:: internal   |  ios::showpos )
7.  <<setw ( 10 ) << 123  << endl;
8.  return 0;
9.  }
+       123
```

შედეგი ასეთი იქნება.

შევსება (fill, setfill)

fill() წევრ-ფუნქციაში მოიცემა სიმბოლო, რომლითაც უნდა შეივსოს ველი გამოტანისას, თუ არაა სიმბოლო არჩეული, გვექნება ცარიელი ადგილები. იგივე მიზნით გამოიყენება მანიპულატორი setfill.

ისევ პატარა მაგალითი:

```
1.  #include <iostream>
2.  using namespace std;
3.  #include <iomanip>
4.  int main()
5.  {
6.  int x = 10000;
7.  cout<< x <<endl;
8.  cout . setf (ios :: showbase );
9.  cout << setw  (10) << x << '\n';
10. cout.setf( ios:: internal, ios::adjustfield );
11. cout << setw( 10 ) << hex << x<<endl;
12. cout.setf( ios::right, ios::adjustfield);
13. cout.fill('*');
14. cout << setw(10) << dec << x << '\n';
15. cout.setf(ios::internal, ios::adjustfield );
16. cout << setw ( 10 ) << setfill ('^')  <<hex << x<< endl;
17. return 0;
18. }
```

შედეგი ასეთია:

```
10000
    10000
0x      2710
*****10000
0x^^^^2710
```

რიცხვითი სისტემების ფუძე
(ios::dec, ios::oct, ios::hex, ios::showbase)

სტატიკური ელემენტი ios::basefield შეიცავს ალმებს, ios::oct, ios::hex, ios::dec, რომელიც შეესაბამება მთელ რიცხვებს რვაობითი, თექვსმეტობითი და ათობითი მნიშვნელობებით. თუ წინასწარ არაა ცნობილი, რომელი სისტემით ვმუშაობთ, შეთანხმებით იგულისხმება ათობითი. "ნაკადიდან ამოღების" ოპერაციით, შეთანხმებისამებრ მონაცემების დამუშავება მიმდინარეობს ისე, როგორც ის შემოდის: მთელი რიცხვები, დაწყებული 0-დან აღიქმება, როგორც რვაობითი, 0x ან 0X-ით დაწყებული მთელები თექვსმეტობითია, ხოლო ყველა სხვა დანარჩენი - ათობითი. თუ ნაკადისათვის არჩეულია ფუძე, ამ ნაკადში დამუშავება მიმდინარეობს მოცემული ფუძით, მანამ სანამ ის არ შეიცვლება ან პროგრამა არ დამთავრდება.

ისევ მაგალითი:

```
1.  #include <iostream>
2.  using namespace std;
3.  #include <iomanip>
4.  int main()
5.  {
6.  int x=100;
7.  cout<< satiosflags ( ios:: showbase )
8.  <<x << '\n'
9.  << oct  <<  x << '\n'
10. <<hex  << x << endl;
11. return 0;
12. }
```

შედეგი

```
100
0144
0x64
```

გამოტანისათვის დავაყენოთ ალამი showbase, ჩვენთვის სასურველი ფუძით. ათობითი რიცხვები გამოგვყავს ჩვეულებრივ, რვაობითი 0-ით მაღალ თანრიგში, თექვსმეტობითი ინდიკატორებით 0x ან 0X მაღალ თანრიგებში.(uppercase ალამით ხდება არჩევა) ჩვენ მაგალითში გამოყენებული იყო ალამი showbase.

მცოცავმძიმიანი რიცხვები; ექსპონენციალური ფორმატი
(ios::scientific, ios::fixed)

როგორც setf საშუალებით გამოიყენება ios::adjustifield, ios::basefield საშუალებით- ალმების ios::oct, ios::hex, ios::dec ბიტები, ასევე ios::floatfield საშუალებით გამოიყენება ios::scientific და ios::fixed. ეს უკანასკნელები მართავს მცოცავი მძიმით რიცხვების გამოტანას. ალამი scientific გამოიყენება ექსპონენციალურ ფორმატში რიცხვის გამოტანისას(c++-ში მას სამეცნიეროს ეძახიან, მაგრამ ეს არაა ის, რასაც სხვა ენებში სამეცნიერო ქვია). რაც შეეხება fixed, მას გამოყავს მცოცავმძიმიანი რიცხვები ფიქსირებულ ფორმატში, ათობითი წერტილის მარჯვნივ გარკვეული რაოდენობის თანრიგებით(წევრი-ფუნქცია precision() –ით შერჩეული სიზუსტის და მიხედვით). ამ ალმების შერჩევის გარეშე თვითონ ხდება ფორმატის განსაზღვრა.

მოვიყვანოთ მაგალითი:

```
1.  #include <iostream>
2.  using namespace std;
3.  #include <iomanip>
4.  int main()
5.  {
6.  double x= .001234567,  y= 1.946e9;
7.  cout<<  " შეთანხმებით : \n"
8.  << x  << '\t'  <<  y << '\n';
9.  cout.setf (ios:: scientific,  ios::floatfield );
10. cout<<" ექსპონენციალური  ფორმა  : \n"
    << x  << '\t'  <<  y << '\n';
11. cout.unsetf (ios::scientific);
12. cout<< "შეთანხმებით unsetf –ის შემდეგ:  \n"
13. << x  << '\t'  <<  y << '\n';
14. cout.setf(  ios::fixed, ios::floatfield );
15. cout.precision(6);
16. cout<< "ფიქსირებული წერტილით :\n"
17. << x  << '\t'  <<  y << '\n';
18. return  0;
19. }
```

გამოძახება cout.setf (0,ios::floatfield) აღადგენს მცოცავმძიმიანი რიცხვების გამოტანას შეთანხმებით. ამ მაგალითში რიცხვების გამოტანა ხდება ექსპონენციალურ და ფიქსირებულ ფორმატებში setf() ფუნქციის ორი არგუმენტისა და ios::floatfield ალამის გამოყენებით.

ქვედა და ზედა რეგისტრებში გამოტანის მართვა (ios::uppercase)

ios::uppercase ალამის არსებობის შემთხვევაში სიმბოლოები X და E თექვსმეტობით და ექსპონენციალურ ფორმატებში, გამოიტანება ზედა რეგისტრში, შესაბამისად. ამ ალამის არსებობისას თექვსმეტობით ფორმატში ყველა ასო ზედა რეგისტრში გამოიტანება.

ისევ მაგალითი:

```
1.  #include <iostream>
2.  using namespace std;
3.  #include <iomanip>
4.  int main()
5.  {
6.  cout<< setiosflags ( ios::uppercase )
7.  <<  " ზედა რეგისტრში ექსპონენციალური და თექვსმეტობითი
    ფორმატით გამოტანა  \n"
8.  << 4.345e10  << '\n' << hex  << 123456789 <<  endl;
9.  return 0;
10. }
```

შედეგი

ზედა რეგისტრში ექსპონენციალური და თექვსმეტობითი ფორმატით გამოტანა
4.345E+10
75BCD15

ფორმატის ალმების შერჩევა და გადაგდება
(flags, setiosflags და resetiosflags)

თუ წევრ-ფუნქციას flags() არგუმენტი არა აქვს, იგი აბრუნებს ფორმატის ალმებს long ტიპის მნიშნელობით.

ყოველი სისტემისათვის ალამთა საწყისი მნიშვნელობა სხვადასხვაა. მოვიყვანოთ მაგალითი, სადაც წევრი-ფუნქცია flags() გამოყენებით, შევარჩევთ ფორმატის ახალ მდგომარეობას, შევინახავთ ძველს და შემდგომ ისევ აღვადგენთ საწყის მდგომარეობას.

```
1.  #include <iostream>
2.  using namespace std;
3.  #include <iomanip>
4.  int main()
5.  {
6.  int i=1000;
7.  double d= 0.0947628;
8.  cout<< "  flags ცვლადის მნიშვნელობა  = "
9.  << cout.flags()
10. << "\n  int  i და  double  d  საწყის ფორმატში   \n"
11. << i  << '\t' <<d  <<  "\n\n";
12. long originalformat=
          cout.flags ( ios::oct   |  ios::scientific);
13. cout<< "   flags   ცვლადის მნიშვნელობა = "
14. << cout.flags()
15. << "\n   int  i და  double  d   ახალ ფორმატში   \n"
16. << i  << '\t' <<d  <<  "\n\n";
17. cout.flags( originalformat );
18. cout<< "   flags   ცვლადის მნიშვნელობა  = "
19. << cout.flags()
20. << "\n   გავიმეოროთ  int  i   და double  d  საწყის ფორმატში   \n"
21. << i  << '\t' <<d  <<  "\n\n";
22. return 0;
23. }
```

წევრი-ფუნქცია setf() არგუმენტის სახით არჩევს ფორმატის ალმებს და აბრუნებს ამ ალმების წინა მნიშვნელობას long ტიპის სახით. როგორც ქვემოთ არის ნაჩვენები:

```
long previousFlagsSettings=
        cout.setf( ios: showpoint   |  ios :: showpos );
```

წევრ-ფუნქციას setf() აქვს long ტიპის ორი არგუმენტი:

```
cout.setf ( ios:: left,  ios :: adjustfield );
```

ეს ოპერატორი თავდაპირველად ასუფთავებს ios::adjustfield ბიტს და შემდეგ არჩევს ios::left ალამს. setf() ალამის ეს ვარიანტი მოქმედებს ბიტურ ველზე, რომელიც დაკავშირებულია ios::basefield(შეიცავს ios::dec, ios::oct და ios::hex), ios::floatfield (შეიცავს ios::scientific და ios::fixed) და ios::adjustfield(შეიცავს ios::left, ios::right და ios::internal) – თან.

წევრი-ფუნქცია unsetf გადაყრის აღნიშნულ ალმებს და აბრუნებს მათ წინა მნიშვნელობას.

ნაკადის შეცდომების მდგომარეობა

ნაკადის მდგომარეობის შემოწმება შესაძლებელია ჩვენს მიერ, შეტანა-გამოტანისათვის გამოყენებული კლასების istream, ostream და iostream- ios ბაზური კლასის ბიტების დახმარებით.

შემავალი ნაკადისათვის ფაილის ბოლოს მაჩვნებელია ბიტი eofbit. პროგრამაში, ნაკადში ფაილის ბოლოს აღმოსაჩენად, შეიძლება eof() ელემენტი-ფუნქციის გამოყენება. გამოძახება cin.eof () აბრუნებს ჭეშმარიტებას- true , თუ cin- ში შეხვდა ფაილის ბოლო, წინააღმდეგ შემთხვევაში –false.

failbit- ისეთი ნაკადის ბოლოს ფიქსირდება, სადაც მოხდა ფორმატირების შეცდომა, მაგრამ სიმბოლოები არ დაკარგულა. წევრი-ფუნქცია fail() განსაზღვრავს ნაკადთან მუშაობისას ხომ არ ყოფილა უარი; როგორც წესი ასეთი ოპერაციის შემდეგ შესაძლებელია მონაცემთა აღდგენა.

badbit – ნაკადის ბოლოს, ნიშნავს მონაცემთა დანაკარგს. წევრი-ფუნქცია bad() გვაუწყებს შეცდომის შესახებ და აღდგენა, როგორც წესი არ ხდება.

goodbit ისეთი ფაილების ბოლოშია, სადაც არ არის არცერთი ბიტი eofbit, failbit და badbit.

წევრი-ფუნქცია good() აბრუნებს true-ს მაშინ, როცა მოცემული ნაკადისათვის, ყველა სხვა ფუნქციები eof(), fail() და bad() აბრუნებს false. შეტანა-გამოტანის ოპერაციები ჩატარებულად შეიძლება ჩაითვალოს, მხოლოდ ასეთ "კარგ" ნაკადებთან.

არსებობს კიდევ წევრი-ფუნქცია rdstate() , რომლითაც შესაძლებელია შეცდომის მდგომარეობის შემოწმება, მაგრამ ზემოთ განხილული ფუნქციებით: eof(), fail(), bad() და good() მუშაობა გაცილებით მოხერხებულია.

გამომავალი ნაკადის დაკავშირება შემავალ ნაკადთან

ჩვეულებრივ, ინტერაქტიულ გამოყენებით პროგრამებში ჩართულია შეტანისათვის istream, ხოლო გამოტანისათვის ostream კლასი. მომხმარებელს შეყავს შესაბამისი მონაცემები, მას შემდეგ რაც ეკრანზე გამოჩნდება შეტყობინება შეტანის შესახებ. რა თქმა უნდა, შეტყობინება უნდა გამოჩნდეს მანამ, სანამ შესრულდება შეტანის ოპერაცია. გამომავალი მონაცემების ბუფერიზაციის შემთხვევაში, ეკრანზე მათ დავინახავთ მას შემდეგ, რაც ბუფერი გაივსება ან პროგრამის ბოლოს. istream და ostream ნაკადებზე ოპერაციების სინქრონიზაციისათვის (დაკავშირებისათვის) C++-ში არსებობს წევრი-ფუნქცია tie(). სწორედ ეს ფუნქცია განაპირობებს, შეტყობინების მიღებას შეტანამდე.

გამოძახება cin.tie(& cout) ;

აკავშირებს (ostream კლასის) cout – ს (istream კლასის) cin-თან. პროგრამირების ენა C++ ქმნის რა მომხმარებლისათვის შეტანა-გამოტანის გარემოს, ამ ოპერაციას ავტომატურად ასრულებს და ზემოთ მოყვანილი გამოძახება ზედმეტია. თუმცა, მომხმარებელმა, ჯობს ცხადად დააკავშიროს istream და ostream კლასების ნაკადების სხვა წყვილები. inputstream- ის შემავალი ნაკადი, გამომავალს რომ დავაშოროთ, უნდა გამოვიყენოთ შემდეგი გამოძახება:

inputStream.tie(o);

ფაილური შეტანა-გამოტანის საფუძვლები

ფაილური შეტანა-გამოტანა მჭიდროდაა დაკავშირებული კონსოლურ შეტანა-გამოტანასთან. აქაც კლასთა იგივე იერარქიაა. თუმცა იქნება სიახლეებიც.

ფაილებთან მუშაობისათვის Aპროგრამაში უნდა ჩავრთოთ სათაური <fstream> , მასში განსაზღვრულია რამოდენიმე კლასი და მათ შორის ifstream, ofstream და fstream ,

რომელიც თავის მხრივ istream და ostream კლასებიდანაა წარმოებული. ეს უკანასკნელი კი თავის მხრივ ios კლასიდანაა წარმოებული. სწორედ ამის გამო ifstream, ofstream და fstream კლასებში დასაშვებია ყველა ის ოპერაციები, რაც განმარტებულია ios- კლასში.

C++- ში ფაილი რომ გავხსნათ, იგი უნდა დავუკავშიროთ ნაკადს. არსებობს სამი ტიპის ნაკადი: შეტანის, გამოტანის და შეტანა/გამოტანის. ფაილის გახსნამდე, სწორედ ნაკადის შექმნაა საჭირო. შეტანის, გამოტანის და შეტანა/გამოტანის ნაკადის შესაქმნელად, აუცილებელია გამოვაცხადოთ შესაბამისად ifstream, ofstream და fstream ტიპის ობიექტები, ე.ი სულ სამი.

```
ifstream in;
ofstream  out;
fstream  io;
```

ამის შემდეგ ჩვენ უკვე გვაქვს ნაკადი და ფაილთან მისი დაკავშირების ერთ-ერთი საშუალებაა ფუნქცია open(_). ეს ფუნქცია სამივე ნაკადური კლასის წევრი ფუნქციაა. მისი პროტოტიპი ასეთია:

```
void    ifstream:: open  ( const  char   * file_name,
                           openmode  regim=ios::in);
```

file_name - აქ სახელში შეიძლება გზის მითითება.

regim - აქ ფაილის გახსნის რეჟიმია მოცემული, ios- კლასში განსაზღვრული openmode გადათვლადი ტიპის ცვლადი. რეჟიმი შეიძლება იყოს ერთ-ერთი ამათგანი:

```
ios::app            ios::in
ios::ate            ios::uot
ios::binary         ios::trunc
```

OOR ოპერატორით შესაძლებელია ორი ან მეტი რეჟიმის გაერთიანება.

ios::app რეჟიმით, გამოტანისათვის, ხდება ჩასაწერად გახსნილი ფაილის ბოლოში ინფორმაციის დამატება.

ios::in და ios::out რეჟიმები გულისხმობს ფაილის გახსნას (შეტანა-კითხვა)-ნაკადიდან ამოღებისათვის და (გამოტანა-ჩაწერა)-ნაკადში ჩასმისათვის შესაბამისად.

ios::binary ფაილის გახსნა ორობით რეჟიმში. შეთანხმებით ყველა ფაილი იხსნება ტექსტურ რეჟიმში. ამ რეჟიმში, ორობითისაგან განსხვავებით ხდება ზოგიერთი სიმბოლოების გარდაქმნა. საერთოდ, ნებისმიერი ფაილი, ფორმატირებულ ტექსტს შეიცავს ის, თუ დაუმუშავებელ მონაცემებს, შეიძლება გაიხსნას როგორც ტექსტურ ისე ორობით რეჟიმში, განსხვავება მხოლოდ ზოგიერთი სიმბოლოების გარდაქმნაშია.

ios::trunc რეჟიმი წაშლის ფაილს ნულოვან სიგრძემდე. იგივე ხდება, როცა ofstream –ით ვაცხადებთ არსებულ ფაილს. ფაილის შემცველობა წაიშლება, ემზადება ახალის ჩასაწერად(rewriteOD) ;

მაგ.

```
ofstream mystream;
   mystream.open( “test”);
```

ფაილი სახელად “test” გავხსენით ჩასაწერად. აქ ნაკადის ტიპი მიუთითებს რა რეჟიმში ვმუშაობთ და open()- ში ცხადად ჩაწერა არაა საჭირო. ფაილის გახსნა მოხდა თუ არა ჩვენ შეგვიძლია შევამოწმოთ შემდეგი ინსტრუქციით:

```
if ( ! mystream )  {
    cout  <<  “file  ar gaixsna  /n”;   return 1;    }
```

ნაკადის მნიშვნელობა ჭეშმარიტია, თუ ყველაფერი კარგადაა, შეცდომის შემთხვევაში იგი მცდარია-false.

როგორც წესი, ფაილის გასახსნელად open() ფუნქციას არ გამოვიყენებთ ხოლმე, შემოწმება ხდება ავტომატურად. ფაილს გავხსნით შემდეგი წესით:

ifstream mystream (“myfile “);

აქ ჩვენ გავხსენით ფაილი შესატანად-წასაკითხად.

ფაილის დასახურავად გამოიყენება წევრი ფუნქცია close(). მაგ. mystream ნაკადთან დაკავშირებული ფაილის დახურვა ხდება შემდეგი ინსტრუქციით:

mystream.close();

ფუნქცია eof() -ით გავიგებთ მივაღწიეთ თუ არა ფაილის ბოლომდის. მისი პროტოტიპი ასეთია: bool eof(); ფუნქციის მნიშვნელობა ჭეშმარიტია, თუ ფაილის ბოლოში ვართ, თუ არა იგი მცდარია.

მას შემდეგ რაც ფაილი გავხსენით, მასში ჩაწერა და მისგან კითხვა ხდება ისევე როგორც კონსოლური შეტანა-გამოტანის დროს. ოპერატორები (<<)- წერა და (>>)- კითხვა იგივეა, მხოლოდ საჭირო ნაკადები cin და cout უნდა შევცვალოთ ფაილთან დაკავშირებული ნაკადებით შესაბამისად. მთელი ინფორმაცია ფაილში ინახება იგივე ფორმატში, როგორც ეკრანზე. როგორც წესი (<<) და (>>) ამ ოპერატორებით დამუშავებისას ფორმატირებული ტექსტიანი ფაილები, უნდა გაიხსნას ტექსტურ რეჟიმში და არა ორობითში.

მოვიყვანოთ მაგალითი, სადაც იქმნება ფაილი გამოსატანად, მასში ვწერთ ინფორმაციას და ვხურავთ. შემდეგ ფაილი იხსნება უკვე წასაკითხად და ჩვენს მიერ ჩაწერილ ინფორმაციას ვკითხულობთ და ფაილს ისევ ვხურავთ:

```
1.  #include  <iostream>
2.  #include  <fstream>
3.  using   namespase  std;

4.  int main()
5.  {
6.  ofstream  fout ( “test”  );  //  ფაილის შექმნა გამოსატანად- ნაკადში ჩასმა
7.  if ( !fout )   {
8.  cout << “ faili ar ixsneba \n“;
9.  return 1;
10. }
11. fout << “ hello! \n “;  //  ფაილში ჩაწერა
12. fout << 100 << ‘   ‘ << hex << 100 << endl;
13. fout.close();
14. ifstream  fin ( “test “ );  //  ფაილის გახსნა შესატანად- ნაკადიდან ამოღება-
    კითხვა
15. if ( !fin ) {
16. cout << “faili ar ixsneba  \n”;
17. return 1;
18. }
19. char  str[80];
20. int   i,j;
21. fin >> str;  //  ფაილიდან კითხვა
22. fin>>i>>j;
23. cout << str << '  ' <<  i <<'  ' <<endl;
24. fin.close();
25. return 0;
26. }
```

დადგენილია, რომ თუ ფაილური შეტანა-გამოტანა შევასრულეთ (<<) და (>>) ამ ოპერატორებით, ინფორმაცია ისე ფორმატირდება Qთითქოს ის ეკრანზე იყოს მოთავსებული.

შემდეგ მაგალითში ინფორმაციას ვკითხულობთ კლავიატურიდან და ვწერთ ფაილში. პროგრამა წყვეტს მუშაობას, თუ წაკითხული სტრიქონის პირველი სიმბოლო $-ია.

```
1.  #include  <iostream>
2.  #include  <fstream>
3.  using namespace  std;

4.  int main()
5.  {
6.  ofstream  out (“ test “ );  // ფაილის გახსნა ჩასაწერად
7.  if ( !out) {
8.  cout << “faili ar ixsneba \n”;
9.  return 1;
10. }
11. char  str[80];
12. cout <<”waikiTxeT striqonebi, $-iT daamTavreT\n”;
13. do  {
14. cout << “:   “;
15. cin >> str;
16. out << str  <<endl;
17. }   while   ( *str    !=  ‘$’);
18. out.close();
19. return 0;
20. }
```

შემდეგ პროგრამაში ხდება არსებული ტექსტური ფაილის კოპირება, თანაც ცარიელი ადგილები (ხარვეზი) იცვლება სიმბოლოთი (|). ფაილის ბოლოს კონტროლისათვის გამოიყენება ფუნქცია eof() . ეს ასევე skipws- ალამის გამოყენების კარგი მაგალითიცაა:

```
1.  #include  <iostream>
2.  #include  <fstream>
3.  using namespace  std;

4.  int main()
5.  {
6.  ifstream  fin ( “test1 “);   //Fფაილის გახსნა წასაკითხად
7.  ofstream  fout ( “test2 “);  //  ფაილის გახსნა ჩასაწერად
8.  char  ch;
9.  fin.unsetf( ios::skipws);   //  არ გამოვტოვოთ ცარიელი ადგილები
10. while ( !fin.eof() )  {
11. fin  >>  ch;
12. if( ch==’ ‘)   ch= ‘ |’;
13. if ( !fin.eof()  )  fout  <<  ch;
14. }
15. fin.close();
```

```
16. fout.close();
17. return 0;
18. }
```

სავარჯიშო

1. შეადგინეთ პროგრამა ტექსტური ფაილის კოპირებისათვის და გამოიტანეთ კოპირებული სიმბოლოების რაოდენობა.
2. შეადგინეთ პროგრამა შემდეგი ინფორმაციის ფაილში phone შესატანად:
   Isaak Newton, 415 555-3423
   Robert Goddard, 213 555-2312
   Enriko Fermi, 202 555-1111
3. შეადგინეთ პროგრამა ფაილში სიტყვათა რაოდენობის დასათვლელად. სიმარტივისათვის სიტყვად ჩავთვალოთ ყველაფერი, რომელიც ორივე მხარეს სიცარიელეს შეიცავს.

არაფორმატირებული ორობითი შეტანა-გამოტანა

ბევრ შემთხვევაში მიღებულია ტექსტური ფაილებით (ASCII კოდებში ჩაწერილი ინფორმაციის მატარებელი) მუშაობა, მაგრამ ზოგჯერ აუცილებელია ინფორმაციის არაფორმატირებული, საწყისი სახით შეტანა-გამოტანა. ამ მიზნით არსებობს ფუნქციათა ფართო დიაპაზონი.

ორობითი შეტანა-გამოტანის დაბალ დონეზე არსებობს ფუნქციები get() და put() ; get() ბაიტის კითხვა და put() ჩაწერა. ეს ორი ფუნქცია ყველა ნაკადური კლასის წევრი ფუნქციაა. ჩვენ განვიხილავთ ამ ფუნქციათა რამოდენიმე ვერსიას:

```
istream &get( char &simvol );
ostream &put ( char simvol);
```

get() ფუნქცია მასთან დაკავშირებული ნაკადიდან კითხულობს ერთ სიმბოლოს და გადასცემს მის მნიშვნელობას არგუმენტს simvol. ფუნქცია put() წერს ნაკადში simvol-ის მნიშვნელობას.

ორობითი მონაცემების ბლოკების წასაკითხად და ჩასაწერად გამოიყენება ფუნქციები read() და write(); მათი პროტოტიპები ასეთია:

```
istream &read (char *buf, streamsize n_bite);
ostream &write ( const char *buf, streamsize n_bite);
```

read() ფუნქცია (n_bite-1) რაოდენობის ბაიტებს კითხულობს ნაკადიდან და გადასცემს მას *buf მიმთითებლით განსაზღვრულ მასივს.

write() ფუნქცია, პირიქით, ბუფერიდან წერს ნაკადში (n_bite-1) რაოდენობის ბაიტებს. streamsize მთელი ტიპის ერთერთი ფორმაა.

თუ კითხვის დროს ფაილის ბოლო უფრო ადრე აღმოჩნდება, ვიდრე ბაიტების მითითებული რაოდენობაა, read() ფუნქცია წყვეტს მუშაობას და ბუფერში იმდენი სიმბოლო აღმოჩნდება, რამდენიც ეწერა ფაილში. სიმბოლოთა რაოდენობის გაგება შესაძლებელია წევრი ფუნქცია gcount()-ის გამოყენებით, რომლის პროტოტიპი შემდეგია:

```
streamsize gcount();
```

ყველა ეს ფუნქციები გამოიყენება ორობითი ფაილების შემთხვევაში და საჭიროა წინასწარ მივუთითოთ რეჟიმი ios::binary. შევნიშნოთ, რომ ამ ფუნქციების გამოყენება ტექსტურ რეჟიმში გახსნილ ფაილებზეც შეგვიძლია, მაგრამ უნდა გვახსოვდეს ზოგიერთი არასასურველი გარდაქმნების შესახებ.

შემდეგ პროგრამაში get() ფუნქციის გამოყენებით, ეკრანზე გამოგვაქვს ფაილის შემცველობა:

```
1.  #include  <iostream>
2.  #include  <fstream>
3.  using   namespace  std;

4.  int main()
5.  {
6.  char  ch;
7.  ifstream   in (“ test “);
8.  if  (!in)  {
9.  cout << “ faili ar ixsneba \n”;
10. return 1;
11. }
12. while  ( !in.eof() )  {
13. in.get(ch);
14. cout<< ch;
15. }
16. in.close();
17. return 0;
18. }
```

შემდეგ პროგრამაში მომხმარებლისგან შეტანილი ინფორმაციის ჩასაწერად გამოიყენება ფუნქცია put() . სიმბოლო $ გამოყენებულია პროგრამის დასასრულებლად:

```
#include  <iostream>
#include  <fstream>
using   namespace  std;
int main()
{
char  ch;
ofstrem  out ( “test  “);
if  (!out) {
          cout  << “ faili ar ikiTxeba\n”;
       return 1;
}
 cout << “ damTavrebsaTvis   waikiTxeT  $  \n”;
do   {
       cout  <<  “:  “;
       cin.get(ch);
        out.put(ch);
}   while   (ch!=’$’);
out.close();
return  0;
}
```

როგორც ვხედავთ, cin ნაკადიდან სიმბოლოების კითხვა ხდება ფუნქციით get() , ამიტომ ჩვენ საწყის ცარიელ სიმბოლოებს არ დავკარგავთ.

შემდეგ პროგრამაში, სტრიქონის და რიცხვის ჩასაწერად ფაილში, გამოყენებულია ფუნქცია write():

```
1.  #include  <iostream>
2.  #include  <fstream>
3.  #include  <cstring>
4.  using   namespace  std;

5.  int main()
6.  {
7.  ofstream  out (“test “ ,ios::out   |    ios::binary );
8.  if  (!out)  {
9.  cout  << “faili ar ixsneba  \n”;
10. return 1;
11. }

12. double  num=100.45;
13. char  str [ ] = “ es Semowmebaa “;
14. out.write((char *)   &num, sizeof(double));
15. out.write(str, strlen(str));

16. out.close();
17. return 0;
18. }
```

C++-ში არსებობს მკაცრი კონტროლი ტიპებზე, რომელიმე ერთ ტიპზე მიმთითებელი ავტომატურად არ გარდაიქმნება სხვა ტიპზე მიმთითებლად. სწორედ ამ მიზეზით write() ფუნქციის გამოძახებისას, თუ გამოტანის ბუფერი არაა სიმბოლური მასივი (char*) ტიპზე დაყვანა აუცილებელია.

შევადგინოთ პროგრამა, რომელიც უკანასკნელ მაგალითში შექმნილი ფაილიდან ფუნქცია read()-ის გამოყენებით წაიკითხავს მონაცემებს:

```
1.  #include  <iostream>
2.  #include  <fstream>
3.  using   namespace  std;

4.  int main()
5.  {
6.  ifstream  in ( “test”,  ios::in  |   ios::binary );
7.  if ( !in ) {
8.  cout << “faili ar ixsneba \n”;
9.  return 1;
10. }

11. double num;
12. char  str(80);
13. in.read( (char * )  &num,  sizeof ( double ));
14. in.read (str, 15);
15. str[14]=’\0’;
16. cout << num << ‘ ‘ << str;
```

```
17. in.close();
18. return 0;
19. }
```

შევადგინოთ პროგრამა ფაილში მასივის ჩასაწერად და შემდეგ მის წასაკითხად. გარდა ამისა გამოვიტანოთ წაკითხულ სიმბოლოთა რაოდენობა:

```
1.  #include  <iostream>
2.  #include  <fstream>
3.  using   namespace  std;

4.  int main()
5.  {
6.  ofstream  out ( “test”, ios::out  |  ios::binary );
7.  if  (!out)  {
8.  cout << “faili ar ixsneba \n”;
9.  return  1;
10. }
11. double  nums[4] = { 1.1, 2.2, 3,3, 4.4 };
12. out . write ((char *)   nums,  sizeof(nums));
13. out. close();
14. ifstream  in (“test “,  ios::in   |  ios::binary );
15. if ( !in ) {
16. cout << “faili ar ixsneba \n”;
17. return 1;
18. }
19. in.read ( (char *) &nums, sizeof (nums));
20. int I;
21. for (I=0; I<4;  I++ )
22. cout << nums[I]  << ‘ ‘;
23. cout <<’\n’;
24. cout << in.gcount ()  <<  “ simboloebi  \n”;
25. in.close();
26. return 0;
27. }
```

დამატებითი ინფორმაცია შეტანა-გამოტანის ფუნქციებზე

ადრე წარმოდგენილი ფორმის გარდა get() ფუნქცია შეიძლება გადაიტვირთოს კიდევ შემდეგ სამ ფორმად:

```
istream   &get ( char  *buf,  streamsize  n_bite);
istream   &get ( char  *buf, streamsize  n_bite, char  stop_mark);
int get();
```

პირველი ფუნქცია კითხულობს (n_bite-1) რაოდენობის სიმბოლოებს ნაკადიდან, buf მიმთითებლით განსაზღვრულ მასივში, ან ფაილის ბოლოს ნიშნის შეხვედრისას ამთავრებს კითხვას. თუ შემავალ ნაკადში შეხვდა ახალი სტრიქონის ნიშანი, get() ფუნქცია ათავსებს მასივში ნულს, ხოლო ნიშანს ტოვებს ნაკადში, შეტანის შემდეგ ოპერაციამდე.

მეორე ფუნქცია კითხულობს (n_bite-1) რაოდენობის სიმბოლოებს მასივში, ან stop_mark სიმბოლოს ან ფაილის ბოლოს ნიშნის შეხვედრისას წყვეტს კითხვას და buf – ით განსაზღვრული მასივის ბოლოში წერს ნულს. stop_mark სიმბოლოს ტოვებს ნაკადში, შეტანის შემდეგ ოპერაციამდე.

მესამე ფუნქცია ნაკადიდან იღებს შემდგომ სიმბოლოს. ფაილის ბოლოს მიღწევისას იგი აბრუნებს EOF-ს.

შეტანის რეალიზაცია ხდება ასევე ფუნქციით getline(). იგი ყველა ნაკადური კლასის წევრი ფუნქციაა. მისი პროტოტიპებია:

istream &getline(char *buf, streamsize n_bite);
istream & getline(char *buf, streamsize n_bite, char stop_mark);

პირველი ფუნქცია , ნაკადიდან კითხულობს სიმბოლოებს buf მიმთითებლით განსაზღვრულ მასივში ან (n_bite-1) რაოდენობისას ან ახალი სტრიქონის სიმბოლოს შეხვედრამდე ან ფაილის ბოლოს სიმბოლოს შეხვედრამდე. მასივის ბოლოს ფუნქცია ათავსებს ნულს, ახალი სტრიქონის სიმბოლოს ნაკადიდან იღებს, მაგრამ მასივში არ ათავსებს.

მეორე ფუნქცია კითხულობს (n_bite-1) რაოდენობის სიმბოლოებს , ან ფაილის ბოლოს ნიშნის ან stop_mark სიმბოლოს შეხვედრამდე. მიმთითებლით განსაზღვრული მასივის ბოლოში ფუნქცია getline() ათავსებს ნულს. stop_mark სიმბოლოს ნაკადიდან იღებს, მაგრამ მასივში არ ათავსებს.

როგორც ვხედავთ ფუნქციები get() და getline() ძალიან გავს ერთმანეთს. განსხვავება იმაშია, რომ getline() კითხულობს და გადააგდებს ნაკადიდან stop_mark სიმბოლოს, ხოლო get() ფუნქცია კითხულობს, მაგრამ ტოვებს ნაკადში კითხვის შემდეგ ოპერაციამდე.

ფუნქცია peek() იძლევა ნაკადიდან მომდევნო სიმბოლოს და ტოვებს მას ნაკადში. ეს ფუნქცია შეტანის ნაკადური კლასების წევრი ფუნქციაა და მისი პროტოტიპი ასეთია:

int peek();

იგი აბრუნებს მომდევნო სიმბოლოს და თუ ფაილის ბოლოში ვართ, აბრუნებს EOF-ს.

ფუნქცია putback () შეტანის ნაკადური კლასების წევრი ფუნქციაა და მას შეუძლია ნაკადიდან ბოლოს წაკითხული სიმბოლო დააბრუნოს ისევ უკან ნაკადში. პროტოტიპი ასეთია:

istream &putback (char c); c- ნაკადიდან ბოლოს წაკითხული სიმბოლოა.

გამოტანის დროს, ნაკადთან დაკავშირებულ ფიზიკურ მოწყობილობებზე უცებ არ ხდება ჩაწერა. ინფორმაცია დროებით ინახება შიდა ბუფერში. დისკზე მისი გადაწერა ხდება, ბუფერის შევსების შემდეგ. არსებობს ფუნქცია flush(), რომელიც ინფორმაციას დისკზე გადაწერს, ბუფერის შევსებამდე. ეს ფუნქცია გამოტანის ნაკადური კლასების წევრი ფუნქციაა და მისი პროტოტიპი ასეთია:

ostream &flush();

ცხოვრების ცუდი პირობების შემთხვევაში, როცა ხშირად ხდება კვების წყაროს მოულოდნელი გამორთვები, გამართლებული და სასურველიცაა flush() ფუნქციის გამოძახება.

სტრიქონის (>>) ამ ოპერატორით კითხვისას, პირველივე გამყოფი სიმბოლოთი კითხვა წყდება. ამიტომ ეს ოპერატორი მოუხერხებელია. ეს პრობლემა ადვილად გადაწყდება getline() ფუნქციის გამოყენებით. ვნახოთ ეს მაგალითზე, დავწეროთ პროგრამა, რომელიც კითხულობს სტრიქონს ცარიელი ადგილებით:

```
1.  #include  <iostream>
2.  #include  <fstream>
3.  using   namespace  std;

4.  int main()
5.  {
6.  char  str(80);
7.  cout<< " momeci    Seni  saxeli da gvari: ";
8.  cin.getline ( str, 79);
9.  cout  << str  << '\n';
10. return 0;
11. }
```

პროგრამირებაში ხშირად გამოიყენება ფუნქციები peek() და putback(); მათი საშუალებით ადვილად მართვადია სიტუაცია, როცა წინასწარ არ ვიცით ყოველ კონკრეტულ მომენტში როგორი ტიპის ინფორმაცია შეგვყავს. ამის მაჩვენებელია შემდეგი პროგრამა. პროგრამაში ვკითხულობთ ან სტრიქონს ან მთელ რიცხვებს. მთელები და სტრიქონები შეიძლება იყოს ნებისმიერი მიმდევრობით.

```
1.  #include <iostream>
2.  #include <fstream>
3.  #include <cctype>
4.  using namespace std;

5.  int main()  {
6.  char ch;
7.  ofstream out("test",ios::out | ios::binary );
8.  if(!out) {
9.  cout << "faili ar ixsneba \n";
10. return 1;
11. }
12. char str[80],  *p;
13. out <<123 << "this is a test "  <<23;
14. out << "Hello there !  " << 99 << "sdf"  <<endl;
15. out.close();
16. ifstream  in( "test", ios::out  |  ios::binary );
17. if (!in) {
18. cout <<"faili ar ixsneba \n";
19. return 1;
20. }
21. do  {
22. p=str;
23. ch= in.peek();
24. if (isdigit(ch) )  {
          while (isdigit( *p=in.get())) p++;
          in.putback(*p);
          *p= '\0';
          cout<< "integer " <<atoi(str);
25. }
26. else if (isalpha(ch) ) {
          while (isalpha( *p=in.get())) p++;
```

```
            in.putback(*p);
            *p='\0';
            cout<<" string: "<<str;
27. }
28. else in.get();
29. cout<<'\n';
30. }  while (!in.eof());
31. in.close();
32. return 0;
33. }
```

სავარჯიშო

1. წინა მაგალითი გადავწეროთ ისე, რომ ფუნქცია getline() შეიცვალოს ფუნქცია get()-ით. ხომ არ იმუშავებს პროგრამა განხვავებულად?
2. ფუნქცია getline-ის გამოყენებით დაწერეთ პროგრამა, რომელიც წაიკითხავს ტექსტური ფაილის ყოველ სტრიქონს და გამოიტანს მას ეკრანზე.

თავისუფალი წვდომა

C++-ის შეტანა-გამოტანის სისტემაში თავისუფალი წვდომა(random access) ხორციელდება ნაკადური ფუნქციებით seekg() და seekp(). მოვიყვანოთ მათი მთავარი ფორმა:

istream &seekg (off_type wanacvleba, seekdir amocana);
ostream &seekp (off_type wanacvleba, seekdir amocana);

off_type ios – კლასში განსაზღვრული მთელი ტიპია. seekdir ios-ში განსაზღვრული გადათვლადი ტიპია, რომელიც შემდეგ მნიშვნელობებს იღებს:

მნიშვნელობა	აზრი
ios::beg	ძებნა ფაილის დასაწყისიდან
ios::cur	ძებნა ფაილის მიმდინარე პოზიციიდან
ios::end	ძებნა ფაილის ბოლოდან

C++-ში შეტანა-გამოტანის სისტემაში გვაქვს ორი მმართველი მიმთითებელი. პირველი- კითხვის მიმთითებელი(get pointer) , რომელიც გვაძლევს ფაილში შემდგომ ადგილს, საიდანაც ხდება ინფორმაციის შეტანა. მეორე- ჩაწერის მიმთითებელი(put pointer), რომელიც გვაძლევს ფაილში ადგილს, სადაც გამოვიტანთ ინფორმაციას. ყოველი შეტანის და გამოტანის დროს მიმთითებელი გადაადგილდება. თუმცა seekg() და seekp() ფუნქციებით შეგვიძლია არათანმიმდევრული წვდომა.

ორივე ფუნქციაში მიმთითებელი გადაადგილდება amocana- ცვლადის შესაბამისად wanacvleba-ს ტოლი სიდიდით;

ქვემოთ მოყვანილი პროგრამით ფაილის დასაწყისიდან მე-10 პოზიციაში ჩაიწერება სიმბოლო ''a'

```
1. #include <iostream>
2. #include <fstream>
3. using namespace std;

4. int main()
5. {
6. fstream out("mmm.txt",ios::in | ios::out |ios::binary);
```

```
7.  if ( !out ) {
                    cout<<"faili ar ixsneba\n";
                    return 1;
8.  }
9.  out.seekp( 10, ios::beg );
10. out.put( 'a' );
11. out.close();

12. return 0;
13. }
```

ქვემოთ მოყვანილ პროგრამაში ვკითხულობთ ინფორმაციას ფაილის მე-3 პოზიციიდან:

```
1.  #include <iostream>
2.  #include <fstream>
3.  using namespace std;

4.  int main()
5.  {       char ch;
6.  ifstream in("mmm.txt");
7.  if( !in) {
                    cout<<"faili ar ixsneba\n";
                        return 1;
8.  }
9.  in.seekg(3,ios::beg);
10. while (!in.eof() )  {
                    in.get(ch);
                    cout<<ch<<endl;
11. }
12. in.close();
13. return 0;
14. }
```

სავარჯიშო

დავწეროთ პროგრამა ტექსტური ფაილის ეკრანზე შებრუნებულად გამოსატანად

დასკვნა

C++-ში მიმდინარეობს ბაიტების ნაკადების შეტანა-გამოტანა.

შეტანა-გამოტანის მექანიზმი მდგომარეობს, რომელიმე მოწყობილობიდან მეხსიერებაში და პირიქით, ეფექტური და საიმედო გზით, მონაცემთა გადაცემაში.

C++-ში შეტანა-გამოტანა შესაძლებელია როგორც "დაბალ" ისე "მაღალ" დონეზე. "დაბალ" დონეზე შეტანა-გამოტანის დროს(არაფორმატირებული) ბაიტების გარკვეული რაოდენობა გადაეცემა მოწყობილობიდან მეხსიერებაში და პირიქით. "მაღალ" დონეზე(ფორმატირებული შეტანა-გამოტანა) – მონაცემები ჯგუფდება რაღაც ნიშნად ელემენტებად, მაგალითად მთელი რიცხვები, მცოცავმძიმიანი რიცხვები, სიმბოლოები, სტრიქონები.

C++-ის პროგრამების დიდი ნაწილი შეიცავს სათაურ ფაილ-<iostream>-ს, რომელიც მოიცავს აუცილებელ ინფორმაციას, კონსოლური შეტანა-გამოტანის ყველა ოპერაციებზე და ფაილებთან სამუშაოდ ასევეა საჭირო სათაური ფაილი <fstream>.

სტატიაში თითქმის ყველა მანიპულატორისათვისაა შერჩეული მაგალითი და მეტიც ვასწავლით, როგორ განვსაზღვროთ ახალი მანიპულატორები.

მიუხედავად იმისა, რომ მონაცემთა შეტანა-გამოტანისათვის არსებობს წევრ-ფუნქციათა ფართო სპექტრი, სტატიაში თითქმის ყველა შემთხვევისთვისაა მოყვანილი სასწავლო მაგალითი. ამით მომხმარებელი საჭიროების შესატყვის ფუნქციას აირჩევს.

ლიტერატურა

1. Г. Шилдт. Самоучитель C++:Пер.с англ. 3-е изд. :-СПб.:БХВ- Петербург, 2001г.
2. Х.М. Деител, Р.Дж.Деител. Как программировать на C++. 3-е изд. Москва .изд. Бином . 2001г.

სტატია მიღებულია: 2004-09-10