C++ podsetnik

Ivan Drecun, Vladan Kovačević

19.10.2021.

1 Pisanje i prevođenje C++ programa

Osnovna struktura C++ programa ista je kao u programskom jeziku C. Program se piše u datoteci sa ekstenzijom `.cpp`.

```
#include <...>

int main() {
    ...
    return 0;
}
```

Program se prevodi na isti način kao za programski jezik C, pri čemu se sada koristi odgovarajući *g++* prevodilac. Sve opcije prevodioca *gcc* koje ste upoznali su dostupne.

```
g++ main.cpp
```

U zavisnosti od korišćenih elemenata programskog jezika C++ možda je potrebno podesiti verziju standarda koju prevodilac koristi. To je moguće izvesti zadavanjem opcije *std*.

```
g++ main.cpp -std=c++17
```

2 Prostor imena *std*

Prostori imena (*namespace*) služe za izbegavanje kolizija u imenima (funkcija, tipova, objekata, ...) definisanih u različitim bibliotekama. Sva imena iz standardne biblioteke jezika C++, poput onih spomenutih u ovom dokumentu (cin, cout, vector, string, count, find, sort, ...), pripadaju prostoru imena *std*.

Imena iz prostora imena *std* moguće je koristiti na dva načina. Prvi je upotrebom prefiksa *std::* ispred svakog korišćenog imena. Naredni primer prikazuje učitavanje elemenata u vektor (detaljnije opisano u odgovarajućem poglavlju).

```
1  int main() {
2      int n;
3      std::cin >> n;
4
5      std::vector<int> vec(n);
6      for(int i = 0; i < n; i++)
7          std::cin >> vec[i];
8      ...
9  }
```

Drugi način je upotrebom naredbe *using namespace*. Ovim se otklanja potreba za pisanjem prefiksa pored svakog imena. Za potrebe algoritamskih kurseva je ovo rešenje zadovoljavajuće, ali u realnoj upotrebi je bolje izbegavati uključivanje celog prostora imena.

```
1  using namespace std;
2
3  int main() {
4      int n;
5      cin >> n;
6
7      vector<int> vec(n);
8      for(int i = 0; i < n; i++)
9          cin >> vec[i];
10     ...
11 }
```

3 Standardni ulaz i izlaz

Za rad sa standardnim ulazom i izlazom potrebno je uključiti zaglavlje *iostream*.

```
#include <iostream>
```

Čitanje sa standardnog ulaza vrši se upotrebom objekta *cin* operatorom >>. Analogno, pisanje na standardni izlaz vrši se upotrebom objekta *cout* operatorom <<. Za razliku od funkcija *scanf* i *printf* u programskom jeziku C, nije potrebno zadavati format ulaza i izlaza zato što kompilator ume da prepozna na koji način treba pročitati promenljivu na osnovu tipa promenljive.

```
1  int n;
2  cin >> n;
3
4  cout << n;
```

Čitanje ili pisanje više promenljivih (potencijalno različitih tipova) moguće je realizovati nadovezivanjem odgovarajućih operatora.

```
1  int a = 3, b = 5;
2  cout << a << ' ' << b << '\n';
```

Naredni primer prikazuje na koji način je moguće vršiti učitavanje celih brojeva do kraja standardnog ulaza.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      int x;
7      while(cin >> x) {
8          ...
9      }
10
11     return 0;
12 }
```

Broj cifara iza zareza prilikom ispisa realnih vrednosti moguće je postaviti na fiksnu vrednost, za šta je potrebno uključiti zaglavlje *iomanip*. Na primer, naredni program ispisuje 1.23.

```cpp
#include <iostream>
#include <iomanip>

using namespace std;

int main() {
    cout << setprecision(2) << fixed;

    float x = 1.23456;
    cout << x << '\n';

    return 0;
}
```

4 Struktura *vector*

Vektor je struktura koja omogućava olakšan rad sa nizovima. Za upotrebu vektora potrebno je uključiti zaglavlje *vector*.

```cpp
#include <vector>
```

Vektor celih brojeva dužine n moguće je formirati na sledeći način. Svi elementi vektora se automatski inicijalizuju na podrazumevane vrednosti. Podrazumevana vrednost za *int* je 0.

```cpp
vector<int> vec(n);
```

Vrednost na koju se elementi inicijalizuju moguće je zadati prilikom formiranja vektora. Na primer, naredna linija formira vektor dužine 4, pritom postavljajući vrednost svakog elementa vektora na 5.

```cpp
vector<int> vec(4, 5);
```

Elementima vektora moguće je pristupati indeksiranjem, na isti način na koji se to radi sa nizovima. Naredni primer prikazuje kako je moguće učitati vektor čija je dužina zadata sa standardnog ulaza.

```
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int n;
    cin >> n;

    vector<int> vec(n);
    for(int i = 0; i < n; i++)
        cin >> vec[i];

    return 0;
}
```

Izostavljanjem broja elemenata moguće je formirati prazan vektor.

```
vector<int> vec;
```

Elemente vektora moguće je zadati i direktno u kodu. Broj elemenata vektora je moguće dobiti funkcijom *size*.

```
vector<int> vec = {1, 4, 5, 2};

for(int i = 0; i < vec.size(); i++)
    cout << vec[i] << ' ';
```

Vektor je moguće proširivati dodavanjem novog elementa na kraj vektora upotrebom funkcije *push_back*. Naredni primer prikazuje učitavanje celih brojeva do kraja standardnog ulaza i njihovo smeštanje u vektor.

```
vector<int> vec;
int x;
while(cin >> x)
    vec.push_back(x);
```

Vektor podržava uklanjanje poslednjeg elementa upotrebom funkcije *pop_back*.

```
vec.pop_back();
```

Matricu celih brojeva je moguće realizovati kao vektor vektora celih brojeva. Naredni primer prikazuje formiranje i učitavanje matrice dimenzija $rows \times cols$.

```
vector< vector<int> > mat(rows, vector<int>(cols));

for(int i = 0; i < rows; i++)
    for(int j = 0; j < cols; j++)
        cin >> mat[i][j];
```

5 Struktura *string*

String je struktura koja omogućava olakšan rad sa niskama. Za upotrebu stringova potrebno je uključiti zaglavlje *string*.

```
#include <string>
```

Naredni primer prikazuje čitanje i pisanje niske. Niska se čita do prve pojave blanko karaktera.

```
string s;
cin >> s;
cout << s;
```

Nisku je moguće zadati direktno u kodu. Dužinu niske je moguće dobiti funkcijom *size*. Karakterima niske moguće je pristupati po indeksu. Naredni primer prikazuje ispis svakog karaktera niske u zasebnom redu.

```
string s = "niska";

for(int i = 0; i < s.size(); i++)
    cout << s[i] << '\n';
```

Moguće je vršiti i izmene pojedinačnih elemenata niske. Naredni primer postavlja vrednost početnog karaktera niske na *N*.

```
string s = "niska";
s[0] = 'N';
```

Moguće je vršiti konkatenaciju dve niske operatorom $+$. Vrednost niske c u narednom primeru je *"algoritmi"*.

```
string a = "algo";
string b = "ritmi";
string c = a + b;
```

Moguće je vršiti leksikografsko poređenje niski operatorima $<$, $>$ i $==$. Naredni primer će ispisati *a je manje od b*.

```
string a = "abcd", b = "abxyz";
if(a < b)  cout << "a je manje od b";
if(a > b)  cout << "a je vece od b";
if(a == b) cout << "a je jednako b";
```

6 Struktura *pair*

Struktura *pair* je implementacija uređenog para vrednosti ne nužno istog tipa. Naredni primer prikazuje formiranje uređenog para čiji je prvi element tipa *int*, a drugi element tipa string.

```
pair<int, string> par = {4, "niska"};
```

Elementima uređenog para moguće je pristupiti upotrebom promenljivih *first* i *second*.

```
par.first = 7;
par.second = "Niska";
```

Parove je moguće porediti leksikografski, odnosno prvo po prvom elementu, a u slučaju jednakih prvih elemenata, onda po drugom. Naredni primer će ispisati *a je manje od b*.

```
pair<int, int> a = {3, 5};
pair<int, int> b = {3, 7};
if(a < b)  cout << "a je manje od b";
if(a > b)  cout << "a je vece od b";
if(a == b) cout << "a je jednako b";
```

7 Prenos po referenci

Podrazumevan način prosleđivanja argumenata funkcijama u programskom jeziku C++ je po vrednosti. U narednom primeru se funkciji *uvecaj* prosleđuje vrednost $x = 10$. Vrednost promenljive x se uvećava, ali to ne utiče na vrednost promenljive a i program ispisuje vrednost 10.

```
1  #include <iostream>
2
3  using namespace std;
4
5  void uvecaj(int x) { x++; }
6
7  int main() {
8      int a = 10;
9      uvecaj(a);
10     cout << a;
11     return 0;
12 }
```

Funkciji *uvecaj* je moguće dozvoliti pristup promenljivoj a vršenjem prenosa po referenci. U narednom primeru promenljiva x ne predstavlja celobrojnu vrednost, već predstavlja referencu na celobrojnu promenljivu. Prilikom poziva *uvecaj(a)* se promenljiva x vezuje za promenljivu a i nakon toga promenljiva x predstavlja nadimak za promenljivu a. Dakle, sada je moguće vršiti izmenu promenljive a iz funkcije *uvecaj*, pa program ispisuje vrednost 11.

```
1  #include <iostream>
2
3  using namespace std;
4
5  void uvecaj(int &x) { x++; }
6
7  int main() {
8      int a = 10;
9      uvecaj(a);
10     cout << a;
11     return 0;
12 }
```

Pored mogućnosti izmene vrednosti promenljive prosleđene po referenci, prenos po referenci ima još jednu važnu osobinu. Ukoliko se koristi prenos po vrednosti prilikom prosleđivanja vektora, formira se kopija celog vektora, pa je u složenost takvog poziva potrebno uračunati dodatnih $O(n)$ koraka, gde je n broj elemenata prosleđenog vektora. Ukoliko se koristi prenos po referenci,

nema formiranja kopije vektora i dodatna složenost poziva je $O(1)$. Naredni primer prikazuje dva moguća potpisa funkcije koja implementira algoritam binarne pretrage.

```
1  void bpretraga(vector<int>  vec, int x); // O(n + log n)
2  void bpretraga(vector<int> &vec, int x); // O(log n)
```

8 Ključna reč *auto*

Upotrebom ključne reči *auto* moguće je izostaviti tip promenljive prilikom inicijalizacije i prepustiti određivanje tipa kompilatoru. U narednom primeru kompilator zaključuje da promenljiva x mora biti tipa *int* zato što joj se dodeljuje vrednost tog tipa.

```
    auto x = 8;
```

9 Iteratori

Iteratori su apstrakcija koncepta pokazivača u kontekstu određene strukture podataka (na primer vektora). Kako pokazivač pokazuje na određeno mesto u memoriji, tako iterator pokazuje na određen element u odgovarajućoj strukturi. Iterator na vektor celih brojeva moguće je deklarisati na sledeći način.

```
    vector<int>::iterator it_na_vektor;
```

Svaka struktura podržava dva specijalna iteratora. Iterator na prvi element u strukturi moguće je odrediti funkcijom *begin*. Funkcijom *end* moguće je odrediti iterator koji ne pokazuje ni na jedan element strukture, već konceptualno pokazuje na jednu poziciju nakon poslednjeg elementa strukture.

```
1  vector<int> vec(n);
2  vector<int>::iterator pocetak = begin(vec);
3  vector<int>::iterator kraj    = end(vec);
```

Kao i sa pokazivačima, iteratore je moguće dereferencirati operatorom *. Takođe, moguće je pomerati iteratore ka početku, odnosno kraju strukture upotrebom operatora ++ i −−. Naredni primer prikazuje upotrebu iteratora za

iteraciju kroz jedan vektor. Moguće je koristiti ključnu reč *auto* za skraćivanje zapisa.

```
1  for(auto it = begin(vec); it != end(vec); it++)
2      cout << *it << ' ';
```

Na raspolaganju su i iteratori za obilazak strukture unazad kojima se pristupa funkcijama *rbegin* i *rend*. Iterator *rbegin* pokazuje na poslednji element strukture, dok *rend* konceptualno pokazuje na poziciju ispred prvog elementa strukture. Uvećavanjem ovakvog iteratora operatorom ++ se iterator pomera ka početku strukture. Naredni primer prikazuje upotrebu ovih iteratora za iteraciju kroz vektor unazad.

```
1  for(auto it = rbegin(vec); it != rend(vec); it++)
2      cout << *it << ' ';
```

Moguće je odrediti indeks elementa na koji iterator *it* pokazuje. Jedan način je od iteratora *it* oduzeti iterator koji pokazuje na početak strukture.

```
int index = it - begin(vec);
```

Drugi način je upotrebom funkcije *distance* koja računa razdaljinu između dva iteratora.

```
int index = distance(begin(vec), it);
```

U slučaju strukture koja omogućava nasumični pristup elementima (indeksiranjem) poput vektora ili stringa, složenost određivanja indeksa je $O(1)$. U opštem slučaju to ne mora da važi i složenost može biti čak $O(n)$ gde je n broj elemenata u strukturi.

10 *for*-petlja raspona

Kroz svaku strukturu koja podržava rad sa iteratorima može se iterirati *for*-petljom raspona. U svakom koraku iteracije promenljivoj x se dodeljuje jedna vrednost iz vektora, pri čemu se vrednosti dodeljuju onim redosledom kojim se obilaze iteratorima.

```
1  vector<int> vec(n);
2  for(int x : vec)
3      cout << x << ' ';
```

Ovo se može razumeti kao skraćeni zapis za sledeće.

```
1  vector<int> vec(n);
2  for(auto it = begin(v); it != end(v); it++) {
3      int x = *it;
4      cout << x << ' ';
5  }
```

Ukoliko je potrebno menjati sadržaj elemenata strukture moguće je koristiti referencu. Naredni primer uvećava vrednost svakog elementa vektora za jedan.

```
1  for(int &x : vec)
2      x++;
```

U cilju skraćivanja zapisa u slučaju dugačkog naziva tipa promenljive i na ovom mestu je moguće koristiti ključnu reč *auto*.

```
1  for(auto &x : vec)
2      x++;
```

11 Zaglavlje *algorithm*

Zaglavlje *algorithm* sadrži veliki broj korisnih funkcija koje implementiraju jednostavne algoritme koji se često javljaju prilikom programiranja. Ovde je navedeno nekoliko tih funkcija.

Funkcija *count* u složenosti $O(n)$ određuje broj pojavljivanja tražene vrednosti u strukturi.

```
int br = count(begin(vec), end(vec), 3);
```

Funkcija *find* u složenosti $O(n)$ određuje iterator na prvo pojavljivanje tražene vrednosti.

```
auto it = find(begin(vec), end(vec), 3);
```

Funkcije *min_element* i *max_element* u složenosti $O(n)$ određuju iterator na prvo pojavljivanje najmanje odnosno najveće vrednosti u strukturi.

```
auto it = min_element(begin(vec), end(vec));
```

Funkcija *binary_search* u složenosti $O(\log n)$ određuje da li je tražena vrednost u sortiranoj strukturi ili ne.

```
bool postoji = binary_search(begin(vec), end(vec), 3);
```

Funkcija *fill* u složenosti $O(n)$ postavlja vrednost svakog elementa strukture na zadatu.

```
fill(begin(vec), end(vec), 3);
```

Funkcija *reverse* u složenosti $O(n)$ obrće redosled elemenata strukture.

```
reverse(begin(vec), end(vec));
```

Funkcija *sort* u složenosti $O(n \log n)$ sortira elemente strukture.

```
sort(begin(vec), end(vec));
```

Funkcija *copy* kopira sadržaj jedne strukture u drugu. Naredni primer prikazuje upotrebu funkcije za kopiranje sadržaja vektora *vec1* na početak vektora *vec2* uz pretpostavku da je on dovoljno velik.

```
copy(begin(vec1), end(vec1), begin(vec2));
```

Funkcija *merge* izvršava algoritam spajanja dve sortirane strukture. Naredni primer prikazuje upotrebu funkcije za spajanje sortiranih vektora *vec1* i *vec2*. Rezultat se smešta u vektor *vec* uz pretpostavku da je on dovoljno velik.

```
merge(begin(vec1), end(vec1),
      begin(vec2), end(vec2),
      begin(vec));
```

12 Preklapanje operatora

Operatori u jeziku C++ su funkcije čiji je poziv sintaksno ulepšan. Pod određenim uslovima je moguće proširiti definicije nekih operatora. Naredni primer prikazuje definiciju operatora << za ispisivanje vektora.

```
1  ostream& operator<<(ostream& output, vector<int>& vec) {
2      for(int el : vec)
3          output << el << ' ';
4      return output;
5  }
```

Tip prvog argumenta, kao i povratne vrednosti je *ostream* za rad sa izlaznim tokovima. Jedan objekat tog tipa je *cout*.

```
1  vector<int> vec = {3, 4, 5};
2  cout << vec << '\n';
```