

Algoritmi i strukture podataka

2. Čas, Uvod u C++

Aleksandar Veljković

2017/2018

1 Uvod

Jezik C++ je jezik koji pripada objektno orijentisanoj paradigmi, ipak, u okviru ovog kursa naglasak neće biti na objektno orijentisanoj strani jezika već na stukturama koje se nalaze u okviru standardne biblioteke i njihovim primenama u implementacijama složenijih algoritama.

Datoteka koja sadrži C++ kod ima ekstenziju **.cpp**.

Primer:

```
program.cpp
```

Kompajler koji se koristi za prevodjenje C++ koda je **g++**. Za prevodjenje koda smeštenog u datoteci sa nazivom **program.cpp** izvršava se naredba:

Primer:

```
g++ program.cpp
```

Nakon uspešnog prevodjenja koda, dobija se izvršna datoteka **a.out**.

Za potrebe kursa, biće korišćen C++ standard iz 2011. godine, koji se navodi opcijom **-std=c++11**.

Primer:

```
g++ program.cpp -std=c++11
```

2 Zaglavljiva biblioteka

- **<iostream>** - Zaglavljiva biblioteka koja obezbeđuje rad sa ulaznim i izlaznim tokom (C++ "paralela" za **stdio.h** u C-u)
- **<cstdio>** - C Zaglavljiva **stdio.h**
- **<vector>** - Zaglavljiva biblioteka u kojoj je definisana struktura vektor
- **<string>** - Zaglavljiva biblioteka za rad sa niskama
- **<cmath>** - Zaglavljiva biblioteka za rad sa matematičkim operacijama (C zaglavljiva **math.h**)
- **<cstdlib>** - C Zaglavljiva **stdlib.h**

3 Ispis i učitavanje podataka

Podaci se ispisuju na standardni izlaz usmeravanjem podataka, operatorom `<<`, na izlazni tok **cout**, koji je definisan u okviru **iostream** biblioteke. Ispis više podataka vrši se ulančavanjem više operatora `<<i` podataka za ispis. Prelazak u novi red označava se pomoću **endl**;

Primer:

```
#include <iostream>

int main(
    std::cout << "Trenutno zivimo u " << 21 << ". veku" << std::endl;

    return 0;
}
```

Kako bi se izbeglo ponavljanje naziva imenskog prostora **std**, može se navesti da je podrazumevani imenski prostor **std** naredbom **using namespace std;**.

Primer:

```
#include <iostream>

using namespace std;

int main()
{
    cout << "Trenutno zivimo u " << 21 << ". veku" << endl;

    return 0;
}
```

Učitavanje podataka vrši se usmeravanjem ulaznog toka **cin**, operatorom `>>`, ka promenljivama u kojima će ulazni podaci biti smešteni. Ulagani tok **cin** je, takodje, definisan u okviru biblioteke **iostream**

Primer:

```
#include <iostream>

using namespace std;

int main()
{
    int dan, mesec, godina;

    /* Ucitavanje danasnog datuma */

    cin >> dan >> mesec >> godina;

    return 0;
}
```

4 Vector

Struktura **vector** predstavlja linearno uredjen niz, u memoriji susednih, elemenata. Struktura je definisana u okviru biblioteke **vector**. Pri deklaraciji, tip elemenata koje će vektor sadržati navodi se unutar zagrade *<i>*. Elementima vektora se pristupa operatorom [].

Primer:

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

int main()
{
    /* Promenljiva niz tipa vektor koji sadrzi 5 elemenata
       koji su celi brojevi */

    vector<int> niz(5);

    return 0;
}
```

Moguća je dinamička inicijalizacija broja elemenata u toku izvršavanja.

Primer:

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

int main()
{
    int n;

    cin >> n;

    /* Inicijalizacija broja elemenata vektora na vrednost n,
       ucitanu u fazi izvrsavanja */

    vector<int> niz(n);

    /* Ucitavanje elemenata vektora */

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

    /* Ispisivanje elemenata vektora */

    for(int i = 0; i < n; i++)
        cout << niz[i] << " ";
    cout << endl;

    return 0;
}
```

Nakon deklaracije vektora, nova veličina vektora se može zadati funkcijom **resize**.

Primer:

```
vector<int> niz;  
niz.resize(10);
```

Matrica se može definisati kao vektor vektora, pri čemu broj kolona u svakom redu može (a i ne mora) biti različit.

Primer:

```
int n;  
cin >> n;  
vector<vector<int> > matrica(n); // Navodjenje broja redova  
for(int i = 0; i < n; i++)  
    matrica[i].resize(n); // Navodjenje broja kolona za svaki red;
```

Dodavanje novih elemenata na kraj vektora vrši se funkcijom **push_back()**, dok se izbacivanje elementa sa kraja vrši funkcijom **pop_back()**. Broj elemenata u vektoru dobija se pozivanjem funkcije **size()**.

Primer:

```
vector<int> niz;  
  
for(int i = 0; i < 10; i++)  
    niz.push_back(i); // Dodavanje vrednosti promenljive i na kraj vektora  
  
for(int i = 0; i < niz.size(); i++)  
    cout << niz[i] << " ";  
cout << endl;
```

5 String

Biblioteka **string** obezbedjuje tip i funkcije za rad sa niskama.

Primer:

```
#include <iostream>  
#include <string>  
  
using namespace std;  
  
int main()  
{  
    string niska1 = "abcd";  
    string niska2;  
  
    cin >> niska2;  
  
    cout << niska1 << ", " << niska2;  
  
    return 0;  
}
```

Niske se mogu nadovezivati (konkatenacija) operatorom **+**. Dužina niske se dobija pozivanjem funkcije **length()**. Funkcija **find()** pronalazi prvo pojavljivanje podniske u okviru niske počevši od pozicije u

niski koja je navedena kao drugi argument funkcije. Funkcija **replace()** zamenjuje podnisku koja počinje na poziciji koja je navedena kao prvi argument funkcije, dužina podniske koja se menja zadata je drugim argumentom, dok se trećim argumentom navodi niska koja dolazi na mesto podniske koja se menja. Jednakost niski se može proveriti operatorom **==** dok se leksikografsko poredjenje dve niske vrši funkcijom **compare()**, čiji je argument niska sa kojom se upoređuje.

Primer:

```
string niska = "abc";
cout << niska << endl;

niska = niska + "def";
cout << niska << endl;

cout << "Duzina niske je: " << niska.length() << endl;
cout << "Podniska 'de' se nalazi na poziciji: " << niska.find("de", 0) << endl;
cout << "Izmenjena niska: " << niska.replace(2, 3, "12345") << endl;

if(niska == "abcd")
    cout << "Jednake su" << endl;
else
    cout << "Nisu jednake" << endl;

cout << "Leksikografsko poredjenje: " << niska.compare("abcde") << endl;
```

6 Iteratori

Iteratori su posebna vrsta pokazivača koji se koriste za prolazak (iteraciju) kroz kolekciju elemenata. Svaka kolekcija elemenata (kao npr. **vector**) poseduje iterator koji pokazuje na početni element kolekcije (dobija se funkcijom **begin()**) i iterator koji pokazuje na kraj kolekcije (dobija se funkcijom **end()**). Prolazak kroz kolekciju obavlja se počevši od početnog iteratora, inkrementacijom iteratora koji prolazi kroz kolekciju, sve dok prolazeći iterator ne dostigne iterator za kraj. Promenljiva koja sadrži iterator ima tip **iteratora kolekcije kroz koju se prolazi**.

Primer:

```
vector<int> niz = {1, 2, 3, 4}; // Ovakva inicijaliza zahteva -std=c++11
vector<int>::iterator it; // Prolazeci iterator

for(it = niz.begin(); it != niz.end(); it++)
    cout << *it << " ";
cout << endl;
```

Kako bi se izbeglo pisanje dugog naziva tipa iteratora, od standardna iz 2011. godine, podržano je korišćenje ključne reči **auto**, kojom se kompjleru ostavlja da prepozna tip promenljive.

Primer:

```
vector<int> niz = {1, 2, 3, 4};

for(auto it = niz.begin(); it != niz.end(); it++)
    cout << *it << " ";
cout << endl;
```