Računske vježbe 8

Programiranje II

Uvod

Za programski jezik `C++` definisana je bogata standardna biblioteka gotovih klasa i funkcija za često korišćene složene strukture podataka (nizovi, liste, skupovi itd.) kao i za postupke (pretraživanja, uređivanja itd.). Pokazuje se da su većina tih klasa i funkcija generičke klase, odnosno funkcije. Zbog toga se često i koristi pojam standardna biblioteka šablona (engl. *STL - Standard Template Library*). Svrha ove biblioteke jeste da pruži efikasne realizacije za često korišćene strukture podataka i postupke, a koje ne zavise od tipova podataka od kojih se struktura sastoji odnosno koji se obrađuju. Jedan jednostavan primjer:

```
#include <iostream>
#include <algorithm>

using namespace std;

class Actor
{
public:
  char* name;
  Actor(char* newName)
  {
    name = new char[strlen(newName) + 1];
    strcpy(name, newName);
  }
};

int main()
{
  char brad[] = "BradPitt";
  Actor BradPitt(brad);
  char mima[] = "MimaKaradzic";
  Actor MimaKaradzic(mima);
  swap(BradPitt, MimaKaradzic);
  cout << "Brad Pitt se sada zove: " << BradPitt.name << endl;
  cout << "Mima Karadzic se sada zove: " << MimaKaradzic.name << endl;
}
```

nam pokazuje koliko je ova biblioteka moćna. Naime, upotrebom generičke funkcije `swap` zamijenili smo sadržaj dva objekta. Što je bilo u jednom sada je u drugom i obratno. Ovu funkciju nije teško realizovati za jednu klasu. Ukoliko bismo ih imali na desetine u projektu morali bismo značajno vrijeme da alociramo samo za realizaciju zamjene vrijednosti. Vrši se duboko kopiranje, o tome ne moramo da brinemo.

1. Napisati funkciju `insertMean` koja proširuje niz elemenata tako što između svaka dva elementa niza umeće aritmetičku sredinu susjeda. Korišćenje pomoćnog niza nije dozvoljeno. Iz glavnog programa pozvati funkciju i štampati novo stanje niza.

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

using namespace std;

void insertMean(vector<double>& vec);

int main()
{
    vector<double> vec = { 1.0, 3.0, 5.0, 7.0 };
    cout << "Broj elemenata vektora je: " << vec.size() << endl;
    if (vec.empty())
    {
        cout << "Vektor je prazan!" << endl;
        exit(1);
    }
    else
    {
        cout << "Vrsimo umetanje elemenata..." << endl;
        insertMean(vec);
        cout << "Sada su elementi vektora: ";
        for (auto it = vec.begin(); it != vec.end(); ++it)
        {
            cout << *it << " ";
        }
        cout << endl << "Velicina se promijenila i iznosi: " << vec.size() << endl;
    }
}

void insertMean(vector<double>& vec)
{
    vector<double>::iterator it = vec.begin();
    for (it = vec.begin(); it != prev(vec.end()); ++it)
    {
        it = vec.insert(it + 1, (*it + *(it + 1)) / 2);
    }
}
```

Kroz elemente iterabilnih kontejnera je najlakše proći upotrebom iteratora. Iterator je instanca iteratorske klase pomoću kojeg pristupamo elementima kontejnera. Primjer iteratora za vektor:

```cpp
vector<typename>::iterator it;
```

gdje `typename` može biti proizvoljni tip i zavisi od tipa elemenata vektora od interesa. Iterator se obično inicijalizuje na početak kontejnera sa:

```cpp
it = data.begin();
```

i inkrementira se sve dok se ne dođe do kraja kontejnera. Iteratori mogu da se zamisle kao pokazivači koji pokazuju na elemente nizova. Sve operacije nad iteratorima su ostvarene preklapanjem operatora tako da nema razlike između izraza s običnim pokazivačima i izraza s iteratorima. Vrijednosti tekućeg elementa pristupamo sa `*it`. Iteratori se uvijek prefiksno inkrementiraju. Zašto? Sjetimo se preklapanja postfiksnog i prefiksnog inkrementiranja i zaključaka koje smo tada donijeli. Uslov kojim se prekida iteriranje kroz kontejner se može pogrešno shvatiti:

```cpp
it != data.end();
```

na način što bismo pretpostavili da ćemo se zaustaviti na pretposljednjem elementu. To se neće dogoditi zato što metoda `data::end` vraća iterator koji referencira na hipotetički element koji bi se našao nakon poslednjeg i zapravo ne pokazuje ni na jedan element. Metoda `insert` vraća iterator koji pokazuje na umetnuti element. Pošto vektori u osnovi koriste nizove, umetanje elemenata na pozicijama koje nisu kraj vektora dovodi do toga da kontejner premješta sve elemente koji su bili poslije pozicije umetanja na njihove nove pozicije. Ovo je generalno neefikasna operacija u poređenju sa onom koju za istu operaciju obavljaju druge vrste sekvencijalnih kontejnera kao što su liste.

2. Kreirati algoritam kojim se unosi niz `X` cijelih brojeva i koji kreira niz `Y` na način što se iz niza X eliminišu ponavljanja elemenata. Na izlazu štampati elemente niz Y.
   Primjer: `X=[1, 2, 3, 2, 7, 3, 9, 8, 7, 9]` daje `Y=[1, 2, 3, 7, 8, 9]`.

```cpp
#include <iostream>
#include <vector>
#include <set>

using namespace std;

int main()
{
  int n;
  int temp;
  vector<int> x;
  vector<int> y;
  set<int> xUnique;

  cout << "Unesite duzinu niza X: ";
  cin >> n;

    cout << "Unesite elemente niza X: ";
  for (int i = 0; i < n; i++) {
    cin >> temp;
    x.push_back(temp);
  }

  for (auto it = x.begin(); it != x.end(); ++it)
    xUnique.insert(*it);

  for (auto it = xUnique.begin(); it != xUnique.end(); ++it)
    y.push_back(*it);

  cout << endl << "Elementi niza y su: ";
  for (auto it = y.begin(); it != y.end(); ++it)
    cout << *it << " ";
}
```

Set je asocijativni kontejner koji skladišti jedinstvene elemente istog tipa u sortiranom redosljedu. Ovu osobinu seta iskoristili smo da eliminišemo ponavljanja - u set se ne mogu upisati dva elementa iste vrijednosti. Jedinstveni elementi su se u niz `y` mogli upisati i pomoću metode `assign`:

```cpp
y.assign(xUnique.begin(), xUnique.end());
```

Da budemo krajnje iskreni, zadatak se mogao riješiti u dvije linije koda i bez upotrebe pomoćnog niza na sljedeći način:

```cpp
set<int> xUnique(x.begin(), x.end());
x.assign(xUnique.begin(), xUnique.end());
```