Razvoj softvera

Vežbe 1

Tokovi

- Tok – apstrakcija datoteka, ulaznih i izlaznih uređaja i komunikacionih linija
- `istream` – klasa koja definiše interfejs za izdvajanje podataka iz tokova. Osnova za sve ulazne tokove.
- `ostream` – klasa koja definiše interfejs za prosleđivanje podataka u tokove. Osnova za sve izlazne tokove.
- `iostream` – obuhvata oba interfejsa

Konzolni tokovi

- `cin` – objekat klase `istream`, vezan za standardni ulaz
- `cout` – objekat klase `ostream`, vezan za standardni izlaz
- `cerr` – objekat klase `ostream`, predstavlja izlaz za greške

Formatirano pisanje i čitanje

- operator prosleđivanja <<

```
cout << a;
cout << a << b << c;
```

- operator izdvajanja >>

```
cin >> a;
cin >> a >> b >> c;
```

Neformatirano čitaje i pisanje

- `ostream& put (char)` – ispisuje tačno jedan karakter u binarnom režimu
- `ostream& write(const char* data, int n)` – ispisuje u binarnom režimu niz on n karaktera, počev od karaktera na koji pokazuje pokazivač `data`
- `int get()` - čitanje tačno jednog karaktera u binarnom režimu
- `istream& get(char& c)` - čitanje tačno jednog karaktera u binarnom režimu

Neformatirano čitaje i pisanje

- `istream getline (char* s, streamsize n)` –čitanje reda karaktera. Čita se najviše n-1 znakova i upisuju se u niz karaktera na koji pokazuje `s`.
- `istream& read (char* data, int n)` –čita u binarnom režimu niz od n karaktera i upisuje ih u prostor na koji pokazuje pokazivač `data`

Prostori imena

- prostor imena – grupiše imena (klase, funkcije, objekte, ...) u logičku celinu
- rešava problem “zagađivanja” globalnog prostora imena

```
namespace <ime prostora imena> {
…
}
```

Prostori imena

- upotreba imena iz prostora imena:
  - `<ime prostora imena>::<ime> …`
  - `using <ime prostora imena>::<ime>;`
  - `using namespace <ime prostora imena>;`

- `std` –prostor imena u kome su definisana sva imena standardne biblioteke

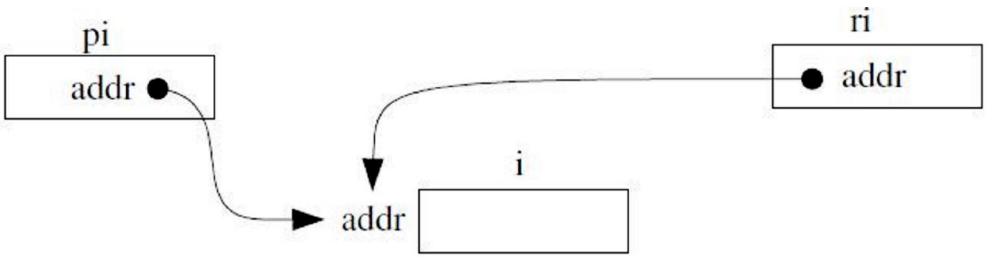
Reference

Pointers

```
int i;
int *pi = &i;
```

References

```
int i;
int &ri = i;
```




```
*pi = 4;
```

```
ri = 4;
```

Kolekcije podataka

- sekvencijalne kolekcije: lista (`list`), vektor (`vector`) i dek (`deque`)
  - elementi poređani u sekvencu i uređeni u nekom poretku
- uređene kolekcije: skup (`set`) i katalog (`map`)
- apstraktne : stek (`stack`), red (`queue`) i red sa prioritetom (`priority_queue`)

Vektor(`vector`)

- Elementi su poređani po svom rednom broju – indeksu
- Efikasan pristup elementima na osnovu indeksa i efikasno dodavanje i uklanjanje elemenata na kraj vektora
- Dodavanje i brisanje elemenata iz sredine vektora zahteva premeštanje velikog broja drugih elemenata i predstavljaju neefikasne operacije

Vektor(`vector`)

- `vector<TipElementa> imeVektora;`
- `size_type size() const` – izračunava veličinu vektora
- `void push_back(const T& x)` – dodavanje elementa na kraj vektora
- `void pop_back()` – brisanje elementa sa kraja vektora
- `T& operator[](size_type n)` – pristupanje elementu sa datim indeksom

  `imeVektora[i]`
- `T& back()` – pristup poslednjem elementu
- `T& front()` – pristup prvom elementu

Iteratori

- Jedan od osnovnih koncepata na kojima počivaju kolekcije standardne biblioteke
- Apstrakcija pokazivača, omogućavaju da se po istom principu obilaze elementi proizvoljne kolekcije
- Omogućavaju obradu elemenata neke kolekcije: poređenje, povećavanje i pristupanje elementima

Iteratori

- U okviru svake apstraktne kolekcije definiše se klasa `iterator` kao apstrakcija pokazivača na elemente kolekcije koja ima bar metode:
  - operator uvećava ++, prelazak na naredni element kolekcije
  - operator provere jednakosti ==, da li su dva iteratora jednaka
  - opeartor različitosti !=
  - operator dereferenciranja *, izračunava referencu na element kolekcije na koji se odnosi iterator
  - `begin()` - iterator koji se odnosi na prvi element kolekcije
  - `end()` - iterator koji se odnosi na prvi element iza poslednjeg elementa kolekcije, tj. vrednost koju iterator treba da ima nakon obrade svih elemenata kolekcije

Iteratori

- `niz[n]`
- Prosleđivanje elemenata u tok

```
for(unsigned i=0; i<velicina; i++)
   cout << niz[i] << endl;
```

- Ili

```
for(unsigned i=0; i<velicina; i++)
   cout << *(niz+i) << endl;
```

Iteratori

Å Ili

```
for(int* i=niz; i<niz+velicina; i++)
   cout << (*i) << endl;
```

Å Ili

```
int* pocetak= niz;
int* kraj= niz+velicina;
for(int* i=pocetak; i!=kraj; i++)
   cout << (*i) << endl;
```

Iteratori

- Niz –kolekcija K

```
K::iterator pocetak= niz.begin();
K::iterator kraj= niz.end();
for(K::iterator i=pocetak; i!=kraj; i++)
    cout << (*i) << endl;
```

Iteratori

Å vrste iteratora:

- ï običan - omogućava čitanje i menjanje elemenata kolekcije
- ï konstantan - omogućava samo čitanje elemenata kolekcije (`const_iterator`)
- ï obrnut – omogućava obilazak elemenata od poslednjeg prema prvom elementu (`reverse_iterator`)
- ï konstantan obrnut (`const_reverse_iterator`)