Napredne tehnike programiranja

Vježba 12

Funkcijski objekti (funktori)

- Funkcijski objekti (funktori) su klase koje imaju predefiniran operator ()
- Omogućavaju nam da kreiramo objekte koji se ponašaju poput funkcija

operator ()

- Operator za poziv funkcije () mora biti definiran kao funkcijski član klase
- Može imati niti jedan ili proizvoljan broj parametara
- Izraz `x(parametri)` se interpretira kao `x.operator()( parametri );`

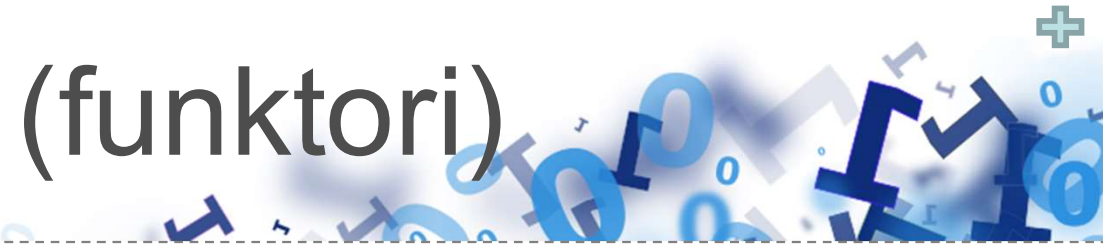
Funkcijski objekti (funktori)

- Primjer funktora:

```
template<class T>
struct dupler {
      void operator()( T &x ) { x = x+x; }
};

int main()
{
      int broj = 5; string rijec = „bla";
      dupler<int> DI;
      DI( broj ); // broj=10
      dupler<string> DS;
      DS( rijec ); // rijec =„blabla"
}
```

Funkcijski objekti (funktori)



- dupler je unarni funktor (prima 1 parametar)
- možemo napraviti i binarne funktore i one bez parametara:

```
template<class T>
class max { // binarni functor
 public:
   T operator()( T x, T y )
   { return (x < y) ? y : x; }
};
int main ()
{
     max<string> maxs;
     string rijec=maxs("ab","aa");// rijec="ab"
     int broj = max<int>()( 3, 7 ); // broj=7
}
```

Funkcijski objekti (funktori)

- Funktor možemo pozvati na dva načina:
  - pomoću objekta kreiranog od predloška
  - direktnim pozivom funkcije iz predloška

```
template<class T>
class max { // binarni functor
 public:
   T operator()( T x, T y )
   { return (x < y) ? y : x; }
};
int main ()
{
     max<string> MS;
     string rijec = MS("ab","aac");// rijec="ab"
     int broj = max<int>()( 3, 7 ); // broj=7
}
```

Funkcijski objekti (funktori)

- pomoću funktora dobijemo funkcije koje se ponašaju fleksibilno

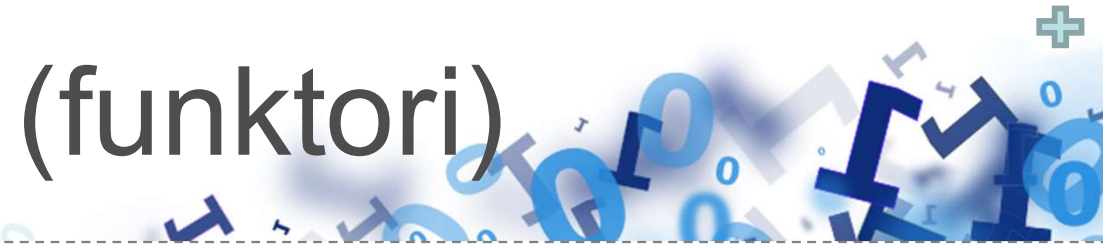
```
class add_x {
public:
   add_x(int x) : x(x) {}
   int operator()(int y) { return x + y; }
private:
   int x;
};

// sad možemo koristiti funktor
add_x add42(42), add100(100);
int i = add42(8);
i=add100(i);
```

Zadatak za vježbu

- Napišite funktor potencija koji ima jedan podatkovni član n tipa int . Kad kreiramo objekt tipa potencija on računa n-tu potenciju cijelog broja.

Funkcijski objekti (funktori)



- pomoću funktora i iteratora možemo npr. za svaki
- element container-a napraviti neku operaciju:

```
template<class T>
struct dupler {
      void operator()( T &x ) { x = x+x; }
};
template<class Iter, class Functor>
void za_svaki( Iter st, Iter en, Functor f ) {
      for( Iter i = st; i != en; ++i )
      f( *i );
}
int main() {
      list<int> L;
      L.push_back(2);L.push_back( 6 );//L=(2, 6)
      za_svaki(L.begin(), L.end() ,dupler<int>());
      // L = (4, 12)
}
```

Funkcijski objekti (funktori)

- lako možemo postići da za_svaki radi nešto sasvim drugo npr. ispisuje na ekran

```
template<class T> struct printer {
   void operator()( T &x ) { cout << *x << " ";
}
};
int main() {
   list<int> L;
   L.push_back(2);L.push_back(6); // L = (2, 6)
   za_svaki(L.begin(),L.end(),printer<int>() );
   // ispis na ekran: "2 6 "
}
```

Predikati

- Predikati su funktori koji vraćaju true ili false

```
template<class T> struct veci{
     bool operator() (T &x, T&y){
          return  x>y ? true : false;}
};

template<class T> struct manji{
     bool operator() (T &x, T&y){
          return  x<y ? true : false;}
};
```

Zadatak

- napišite generičku funkciju za sortiranje spremnika, te pripadne binarne funktore tako da određivati smjer sortiranja spremnika:

```
int X[] = { 2, 6, 4, 7, 8 };
vector<int> V( X, X+5 );
sortiraj( V.begin(), V.end(), manji<int>() );
za_svaki( V.begin(), V.end(),printer<int>() );
// ispise "2 4 6 7 8 "
sortiraj( X, X+5, veci<int>() );
za_svaki( X, X+5, printer<int>() );
// ispise "8 7 6 4 2 "
```

Funktori i algoritmi u STL-u

- Unutar STL-a već postoje predlošci funkcijskih objekata
- for_each i sort funkcijski objekti već dolaze u STL-u, koriste se na posve jednak način;
- `#include <algorithm>`
- postoje i predikati `greater`, `less` koji rade istu stvar kao predikati veci, manji; dakle, možemo pisati:

```
sort( V.begin(), V.end(), greater<int>() );
```

Funktori i algoritmi u STL-u

- set i map mogu prilikom deklaracije primiti i dodatni funktor (default=less) koji kaže kako treba sortirati elemente:

```
set<int, greater<int> > S;
S.insert( 1 ); S.insert( 7 ); // S = (7, 1);
```

- postoji cijeli niz gotovih algoritama koji rade sa funktorima i spremnicima, slijedi kraći pregled

Pretraživanje

- find - vraća iterator prvog elementa jednakog 3. parametru

```
list<int> L;
L.push_back(3); L.push_back(1); L.push_back(7);
list<int>::iterator r= find(L.begin(), L.end(), 1);
// r = iterator na element jednak 1
```

Pretraživanje

- find_if - vraća iterator prvog elementa za kojeg vrijedi unarni predikat poslan kao treći parametar

```
struct pozitivan {
  bool operator()(int x) { return x > 0; }
};

list<int> L;
L.push_back(-3); L.push_back(-1); L.push_back(7);
list<int>::iterator r =
   find_if(L.begin(), L.end(), pozitivan());
// r = iterator na prvi pozitivni element (7)
```

Pretraživanje

- count - vraća broj elemenata jednakih 3. parametru

```
list<int> L;
L.push_back(1); L.push_back(1); L.push_back(7);
int r = count(L.begin(), L.end(), 1);
// r = 2
```

Pretraživanje

- count_if - vraća broj elemenata za koje vrijedi unarni predikat poslan kao treći parametar

```
list<int> L;
L.push_back(3); L.push_back(-1); L.push_back(7);
int r = count_if(L.begin(), L.end(), pozitivan());
// r = 2
```

copy / replace

- copy - kopira sve elemente iz intervala određenog sa prva dva parametra na mjesto koje počinje sa trećim parametrom. Uoči: mora biti dovoljno mjesta tamo gdje se kopira!

```
int X[] = {5, 2, 3, 6, 5};
list<int> L(5);
copy(X, X+5, L.begin());
```

- replace - svaka pojava trećeg parametra zamjenjuje se četvrtim

```
list<int> L;
L.push_back(1); L.push_back(4); L.push_back(1);//L=1,4,1
replace(L.begin(), L.end(), 1, 5);
// L = (5, 4, 5)
```

fill / generate

- fill - popuni raspon određen prvim dvama parametrima sa trećim parametrom

```
list<int> L(5);
fill(L.begin(), L.end(), 3);
// L = (3, 3, 3, 3, 3)
```

- generate - kao fill, ali treći parametar je funktor bez parametara, u donjem slučaju pointer na C-ovu funkciju rand()

```
list<int> L(5);
generate(L.begin(), L.end(), rand);
// L = (234324, 823, 9162, 12312, 5685) npr.
```

transform

- ima dvije varijante
- transform (st1, en1, st2, f) nad svakim elementom raspona [st1, en1> provede unarni funktor f i rezultate sprema u raspon koji počinje sa st2

```
struct negiraj {
   int operator()(int x) { return -x; }
};

list<int> L; L.push_back(3); L.push_back(5);
vector<int> V(5);
transform(L.begin(), L.end(), V.begin(), negiraj());
// V = (-3, -5, 0, 0, 0)
```

transform

- transform (st1, en1, st2, st3, f) nad svakim elementom raspona [st1, en1> i odgovarajućim elementom raspona koji počinje na st2 provede binarni funktor f i rezultate sprema u raspon koji počinje sa st3

```
struct zbrajaj {
  int operator()(int x, int y) { return x+y; }
};
list<int> L; L.push_back(3); L.push_back(5);
list<int> M; M.push_back(6); M.push_back(7);
vector<int> V(5);
transform(L.begin(), L.end(), M.begin(), V.begin(),
zbrajaj());
// V = (9, 12, 0, 0, 0)
```

reverse / random_shuffle

- reverse - preokreće naopako redoslijed elementa u rasponu [st, en>

```
list<int> L;
L.push_back(3); L.push_back(5); L.push_back(7);
reverse(L.begin(), L.end());
// L = (7, 5, 3)
```

- random_shuffle - na slučajan način promjeni redoslijed elemenata u rasponu [st, en>; radi samo na vector-u ili polju

```
vector<int> V;
V.push_back(3); V.push_back(5); V.push_back(7);
random_shuffle(V.begin(), V.end());
// V = (5, 7, 3), npr.
```

Lista algoritama (bez primjera)

- `copy( p1, p2, p3)` Kopira elemente između pokazivača p1 i p2 na lokaciju određenu pokazivačem p3 (vraća kao rezultat pokazivač na poziciju točno iza odredišnog bloka).
- `copy_backward( p1, p2, p3)` Kopira elemente unazad odnosno redom od posljednjeg elementa prema prvom.
- `fill( p1, p2, v)` Popunjava sve elemente između pokazivača p1 i p2 sa vrijednošću v.
- `fill_n( p, n, v)` Popunjava n elemenata počevši od pokazivača p sa vrijednošću v.

Lista algoritama

- `swap_ranges( p1, p2, p3)` Zamjeni blok elemenata između pokazivača p1 i p2 sa blokom elemenata na koji pokazuje pokazivač p3 (kao rezultat vraća pokazivač na poziciju tačno iza bloka na koji pokazuje p3).
- `reverse( p1, p2)` Obrne redoslijed elemenata između pokazivača p1 i p2, tako da prvi element postane posljednji
- `reverse_copy( p1, p2, p3)` Kopira blok elemenata između pokazivača p1 i p2 u obrnutom redoslijedu na lokaciju određenu pokazivačem p3 (tako da će posljednji element u izvornom bloku biti prvi element u odredišnom bloku, itd.). Izvorni blok ostaje nepromjenjen.
- `replace( p1, p2, v1, v2)` Zamjenjuje sve elemente između pokazivača p1 i p2 koji imaju vrijednost v1 sa elementima sa vrijednošću v2.

Lista algoritama

- `replace_copy( p1, p2, p3, v1, v2)` Kopira blok elemenata između pokazivača p1 i p2 na lokaciju određenu pokazivačem p3 uz zamjenu svakog elementa koji ima vrijednost v1 sa elementom koji ima vrijednost v2.
- `rotate( p1, p2, p3)` Posmiče blok elemenata između pokazivača p1 i p3 ulijevo onoliko koliko je potrebno da element na koji pokazuje pokazivač p2 dođe na mjesto elementa na koji pokazuje pokazivač p1 (jedan korak posmicanja se obavlja tako da krajnji lijevi element prelazi na posljednju poziciju, a svi ostali se pomiču za jedno mjesto ulijevo).
- `rotate_copy( p1, p2, p3, p4)` Isto kao “rotate”, ali kopira rotiranu verziju bloka elemenata na poziciju određenu pokazivačem p4, dok sam izvorni blok ostaje neizmijenjen.

Lista algoritama

- `count( p1, p2, v)` Vraća kao rezultat broj elemenata između pokazivača p1 i p2 koji imaju vrijednost v.
- `equal( p1, p2, p3)` Vraća kao rezultat logičku vrijednost "true" ako je blok elemenata između pokazivača p1 i p2 identičan bloku elemenata na koji pokazuje pokazivač p3, a u suprotnom vraća logičku vrijednost "false".
- `min_element( p1, p2)` Vraća kao rezultat pokazivač na najmanji element u bloku između pokazivača p1 i p2.
- `max_element( p1, p2)` Vraća kao rezultat pokazivač na najveći element u bloku između pokazivača p1 i p2.
- `find( p1, p2, v)` Vraća kao rezultat pokazivač na prvi element u bloku između pokazivača p1 i p2 koji ima vrijednost v. Ukoliko takav element ne postoji, vraća p2 kao rezultat.

Lista algoritama

- `remove( p1, p2, v)` Reorganizira elemente bloka između pokazivača p1 i p2 tako da ukloni sve elemente koji imaju vrijednost v. Kao rezultat vraća pokazivač takav da u bloku između pokazivača p1 i vraćenog pokazivača niti jedan element neće imati vrijednost v. Ukoliko niti jedan element nije imao vrijednost v, funkcija vraća p2 kao rezultat. Sadržaj bloka između vraćenog pokazivača i pokazivača p2 na kraju je nedefiniran.
- `remove copy( p1, p2, p3, v)` Kopira blok elemenata između pokazivača p1 i p2 na lokaciju određenu pokazivačem p3, uz izbacivanje elemenata koji imaju vrijednost v. Kao rezultat, funkcija vraća pokazivač koji pokazuje iza posljednjeg elementa odredišnog bloka.

Lista algoritama

- `unique( p1, p2)` Reorganizira elemente bloka između pokazivača p1 i p2 uklanjanjem elementa koji su jednaki elementima koji im prethode. Kao rezultat vraća pokazivač p takav da u bloku između pokazivača p1 i pokazivača p neće biti susjednih elemenata koji su jednaki. Ukoliko u nizu nije bilo susjednih elemenata koji su jednaki, funkcija vraća p2 kao rezultat. Ova funkcija neće ukloniti nesusjedne elemente koji su jednaki. Također, ova funkcija ne mijenja broj elemenata niza nego samo premješta elemente. Sadržaj bloka između vraćenog pokazivača i p2 na kraju je nedefiniran.
- `unique_copy( p1, p2, p3)` Kopira blok elemenata između pokazivača p1 i p2 na lokaciju određenu pokazivačem p3, uz izbacivanje elemenata koji su jednaki elementu koji im prethodi. Kao rezultat, funkcija vraća pokazivač koji pokazuje tačno iza posljednjeg elementa odredišnog bloka.

Zadatak

Koristeći odgovarajuće funkcije iz biblioteke “algorithm”, napišite program koji će za listu od 10 cijelih brojeva unesenih sa tipkovnice ispisati:

- najveći element liste;
- koliko puta se u listi pojavljuje najmanji element;
- koliko u listi cijelih brojeva ima onih koji su potencije broja 2 (tj. brojevi poput 1, 2, 4, 8, 16, 32, itd.);
- element sa najmanjom sumom znamenki;
- Nakon toga, program treba prepisati u drugu listu sve elemente koji nisu dvoznamenkasti, i ispisati novu listu
- Napomena: U programu nije dozvoljeno koristiti petlje (osim za unos elemenata niza), već sve manipulacije treba ostvariti isključivo pozivima funkcija iz biblioteke “algorithm” i korištenjem funkcijskih objekata.