Računarski praktikum 1
Vježbe 10

Zvonimir Bujanović
Vinko Petričević

Funkcijski objekti (funktori)

- objekt čija klasa koja ima predefiniran operator() zovemo funkcijski objekt ili funktor:

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

int main()
{
   int zz = 5; string oo = "AB";

   dupler<int> DI;
   DI( zz ); // zz=10

   dupler<string>()( oo ); // oo="ABAB"
}
```

Funkcijski objekti (funktori)

- dupler je unarni funktor (prima 1 parametar)
- možemo napraviti i npr. binarne funktore, pa i one bez ikakvih parametara:

```
template<class T>
struct max { // binarni funktor
   T operator()( T x, T y )
   { return (x < y) ? y : x; }
};

int main ()
{
   max<string> MS;
   string oo = MS( "ab", "aac" ); // oo="ab"

   int zz = max<int>()( 3, 7 ); // zz=7
}
```

Funkcijski objekti (funktori)

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

```
template<class Iter, class Functor>
void for_each( 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)

   ::for_each( L.begin(), L.end(), dupler<int>() );
   // L = (4, 12)
}
```

Funkcijski objekti (funktori)

- Iako možemo postići da for_each 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)

   ::for_each( L.begin(), L.end(), printer<int>() );
   // ispis na ekran: "2 6 "
}
```

Zadatak 1

- napišite generičku funkciju za sortiranje container-a, te pripadne binarne funktore tako da imamo sljedeću funkcionalnost:

```
int X[] = { 2, 6, 4, 7, 8 };
vector<int> V( X, X+5 );

::sort( V.begin(), V.end(), manji<int>() );
::for_each( V.begin(), V.end(), printer<int>() );
// ispise "2 4 6 7 8 "

::sort( X, X+5, veci<int>() );
::for_each( X, X+5, printer<int>() );
// ispise "8 7 6 4 2 "
```

- funktore koji vraćaju bool (poput veci, manji) zovemo predikati

Funktori i algoritmi u STL-u

- for_each i sort već dolaze u STL-u, koriste se na posve jednak način; sort je izuzetno efikasno implementiran
- `#include <algorithm>`
- postoje i predikati greater, less koji rade istu stvar kao redom predikati veci, manji koje smo napisali u Zadatku 1; dakle, možemo pisati:

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

- set i map mogu primati 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 containerima, 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
```

- 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
```

- 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);
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)
```

- umjesto zbrajaj možemo koristiti i već gotovi plus<int>

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.
```

next_permutation

- next_permutation – mijenja poredak elemenata u container-u tako da on bude iduća po redu pemutacija, vraća 1 ako iduća postoji, a 0 ako ne postoji

```
int X[]={1, 2, 3};
while (1) {
   for_each(X, X+3, printer<int>()); cout << endl;
   if (!next_permutation(X, X+3)) break;
}
```

- može primiti i dodatni parametar (binarni predikat) ako su elementi uređeni na drugačiji način:

```
int X[]={3, 2, 1};
while (1) {
   for_each(X, X+3, printer<int>()); cout << endl;
   if (!next_permutation(X, X+3, greater<int>()))
      break;
}
```

Zadatak 2

- odaberite bilo koja 3 opisana algoritma i napišite svoju implementaciju
- za popis svih dostupnih algoritama vidi
  `http://www.sgi.com/tech/stl/table_of_contents.html`

Datoteke

- #include <fstream>
- datoteku za čitanje otvaramo tako da deklariramo objekt tipa ifstream
- iz datoteke čitamo koristeći operator >>
- datoteku zatvaramo pozivom funkcije close()

```
int main() {
   // otvara postojecu datoteku "datoteka.txt"
   ifstream f( "datoteka.txt" );

   int a, b; string str;
   f >> a >> b >> str; // ucita 2 int-a i string

   f.close();
   return 0;
}
```

Datoteke

- datoteku za pisanje otvaramo tako da deklariramo objekt tipa ofstream
- iz datoteke čitamo koristeći operator <<
- datoteku zatvaramo pozivom funkcije close()

```
int main() {
   // stvara novu datoteku "datoteka.txt"
   ofstream f( "datoteka.txt" );

   int a=3, b; string str = "abc";
   f << a << " " << 5 << " " << str << endl;

   f.close();
   return 0;
}
```

- za nešto napredniju upotrebu datoteka vidi `http://www.cplusplus.com/doc/tutorial/files.html`