Programozási paradigmák (C++)

Sergyán Szabolcs
sergyan.szabolcs@nik.uni-obuda.hu

Óbudai Egyetem
Neumann János Informatikai Kar
Alkalmazott Informatikai Intézet

2019. február 25.

Tömbök

- Tömböt az elemek típusának és a változónév után a méret megadásával deklarálhatunk.
- A tömb méretét egyszerűen nem lehet lekérdezni.
- A tömbön kívüli indexelés futás idejű hibát eredményez.

```
int a[5];
for (int i = 0; i < 5; i++)
{
  a[i] = i * i;
  cout << a[i] << endl;
}
```

Tömbök

- A tömb elemeit mutatón keresztül is elérhetjük. A tömb neve, mint mutató, a tömb 0 indexű elemére mutat.
- Az *i* indexű elemre a tömb nevéhez *i*-t adva hivatkozhatunk.
- Ilyen esetben akár ki is indexelhetünk a tömbből, viszont ekkor nem lefoglalt memória területre hivatkozunk.

```
int a[5];
for (int i = 0; i < 6; i++) //index out of range
{
   *(a + i) = i * i;
   cout << *(a + i) << endl;
}
```

Tömbök a szabad táron

- Tömböket a szabad tárban a `new` operátorral hozhatunk létre.
- A tömbelemekre a változónéven keresztül direktben hivatkozhatunk.
- A tömböt fel kell szabadítanunk a memóriában a `delete[]` operátorral.

```
int* a = new int[5];
for (int i = 0; i < 5; i++)
{
  a[i] = i * i;
  cout << *(a + i) << endl;
}
delete[] a;
```

A vector gyűjtemény

- A vector egy dinamikus méretű tömb. Használatához be kell tölteni a vector fejállományt.
- Deklarációnál meg kell adni, hogy milyen típusú elemeket tárolunk benne.
- A push_back metódus a vector végére beszúr egy új elemet.
- A size metódus megadja a vector aktuális méretét.

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

using namespace std;

int main( )
{
  vector< int > v;
  for ( int i = 0; i < 5; i++ )
  {
    v.push_back( i * i );
  }
  for ( int i = 0; i < v.size( ); i++ )
  {
    cout << v[i] << endl;
  }
}
```

Vector a szabad tárban

- A változó mutató típusú, így a dereferencia (*) operátort kell használni a mutatott érték eléréséhez.
- `(*v).` helyett `v->` használata ajánlott.
- A `vector` az `at` metódussal is indexelhető.

```
vector< int >* v = new vector< int >();
for ( int i = 0; i < 5; i++ )
{
  (*v).push_back( i * i ); // v->push_back( i * i );
}
for ( int i = 0; i < (*v).size( ); i++ ) // v->size( );
{
  cout << (*v)[i] << endl; // v->at(i)
}
delete v;
```

Vector bejárása iterátorral

- Az iterátorral egy `vector` aktuális elemein lehet végighaladni.
- Az iterátor típusú változó egy pointer (mutató).
- `v.begin()` a `vector` első elemére mutat rá.
- `v.end()` a `vector` utolsó eleme utánra mutat.
- Az iterátort növelve a `vector` soron következő elemére mutat rá.

```
vector< int > v;
for ( int i = 0; i < 5; i++ )
{
  v.push_back( i * i );
}
for ( vector< int >::iterator it = v.begin(); it != v.end(); it++ )
{
  cout << *it << endl;
}
```

A const kulcsszó

- Ha egy változó értéke nem változtatható meg a kezdeti értékadást követően, akkor érdemes konstansként deklarálni.
- const változónak mindig kell kezdeti értéket adni.
- Egy mutató esetén a mutatott érték és a mutató maga is lehet const.

```
char s[] = "Stroustrup";
const char* pc = s;
pc[3] = 'g'; //error
char* p = s;
pc = p;

char *const cp = s;
cp[3] = 'a';
cp = p; //error
```

```
int a = 1;
const int c = 2;
const int* p1 = &c;
const int* p2 = &a;
int* p3 = &c; //Error
int* p3 = 7;
```

Referenciák

- A referencia egy alias, azaz egy másik névvel tudunk egy adott változóra hivatkozni.
- Referencia változót a & operátort a változó típus mögé írva tudunk deklarálni.
- A referenciának mindig kell kezdeti értéket kapnia.
- A referencia egy változóra hivatkozhat, viszont a konstans referencia egy literálra is hivatkozhat.
- Konstans referenciának olyan függvény visszatérési értéke is átadható, amely a függvényben létrehozott lokális változóval tér vissza.
- Referenciákat használunk a függvények címszerinti paraméterátadásához is.

```
int a = 39;
int& r = a;
double& dr = 1; //error
const double& cdr = 1;
```

```
double f()
{
  double a = 7.0;
  return a;
}

//...

const double& rr = f();
cout << rr << endl;
```

Véletlenszám generátor

- A `cstdlib` fejállományban definiált `rand()` előállít egy pszeudo random pozitív egész számot.
- A véletlenszám generálás seed-jét az `srand()` függvénnyel lehet beállítani.

```
#include <cstdlib>
#include <ctime>

//...

srand( time(NULL) );
int a = rand() % 100 + 1;
```

Feladatok

1. Egy véletlenszámokból álló tömbben meg kívánjuk számolni a prímszámok darabszámát.
2. Az előző feladatot valósítsuk meg `vector` használatával is.
3. El kívánjuk tárolni a napi kiadásainkat. Ehhez egy kétdimenziós (fűrészfogas) tömböt használunk, melynek sorai a napok oszlopai pedig az egyes költések.
   A kiadások ismeretében lehessen meghatározni a napi összkiadást, a legnagyobb kiadást, stb.
4. Valósítsuk meg az előző feladatot `vectorok` használatával is.