Objektumorientált programozás

Objektumalapú programozás a C++ programozási nyelvben

Sablonok

Darvay Zsolt

Osztályok, kivételkezelés és sablonok

11. Az osztály fogalma
12. Kivételkezelés
13. Sablonok

13. Sablonok

Áttekintés

13.1. Függvénysablonok

- Függvénysablonok definiálása
- Függvénysablonok túlterhelése

13.2. Osztálysablonok

- Osztálysablonok definiálása
- Osztálysablonok objektumai
- Osztálysablonok konstruktora

13.1. Függvénysablonok

- Ha egy függvényt úgy szeretnénk definiálni, hogy különböző típusú paraméterekkel lehessen meghívni, akkor függvénysablont használhatunk.
- A függvénysablon típusparamétereket tartalmaz. Ezeket a meghíváskor a megfelelő típusokkal helyettesíti a rendszer.

Függvénysablonok definiálása

- A függvény fejléce elé a:

  template <típusparaméterek_listája>

kerül, ahol a típusparaméterek listáját a

  class név_1, class név_2, ..., class név_n

alakban adjuk meg.

- Ezt követően a név_i egy típust helyettesíthet.
- A függvény formális paramétereinek listája kell tartalmazza az összes típusparamétert.

Példa függvénysablonra

```
#include <iostream>
using namespace std;
template <class T>
void csere(T& a, T& b) {
  T x;
  x = a;
  a = b;
  b = x;
}
```

A „kiir_cserel” függvénysablon

```
template <class T>
void kiir_cserel(T& a, T& b) {
  cout << "Csere elott:\n";
  cout << "a =" << a << "\t";
  cout << "b =" << b << endl;
  csere(a, b);
  cout << "Csere utan:\n";
  cout << "a =" << a << "\t";
  cout << "b =" << b << endl;
}
```

A fő függvény

```
int main() {
  int u = 3;
  int v = 5;
  kiir_cserel(u, v);
  double y = 3.2;
  double z = 5.7;
  kiir_cserel(y, z);
}
```

Kimenet:

```
Csere elott:
a =3    b =5
Csere utan:
a =5    b =3
Csere elott:
a =3.2  b =5.7
Csere utan:
a =5.7  b =3.2
```

Példa kivételkezelésre (több kivétel)

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
using namespace std;
template <class T>
void kiir_nemzerus(T x) {
  if (x == static_cast<T>(0))
  throw static_cast<T>(0);
  cout << typeid(x).name() << " " << x << endl;
}
```

A fő függvény

```
int main()
{
  srand((unsigned)time(NULL));

  // try blokk
  // catch blokkok
```

A try blokk

```
try {
 switch (rand() % 5) {
 case 0:
  kiir_nemzerus(static_cast<long>(rand() % 2)); break;
 case 1:
  kiir_nemzerus(static_cast<float>(rand() % 2));
   break;
 case 2:
  kiir_nemzerus(static_cast<double>(rand() % 2));
   break;
 default:                    // esetleg más case is lehet
  kiir_nemzerus(rand() % 2);
 }
}
```

A catch blokkok

```
    catch (int) {
        cout << "Hiba: zerus (int)\n";
    }
    catch (long) {
        cout << "Hiba: zerus (long)\n";
    }
    catch (...) {
        cout << "Hiba: zerus (valos)\n";
    }

}// main
```

Lehetséges kimenetek

- Az alábbiak közül az egyik:
  - int 1
  - long 1
  - float 1
  - double 1
  - Hiba: zerus (int)
  - Hiba: zerus (long)
  - Hiba: zerus (valos)

A függvény létrehozása egy sablon alapján

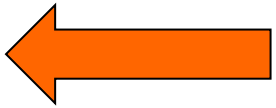
- Egy függvénysablon definícója nem vezet azonnali kódgeneráláshoz. A fordító csak abban az esetben hozza létre a sablon egy példányát, ha egy függvénymeghívással találkozik.
- Ha a következő függvénymeghíváskor az aktuális paraméterek listájának típusa megegyezik az előzővel, akkor a létrehozott függvényt hívja meg a rendszer. Ellenkező esetben a sablon alapján egy újabb példányt hoz létre. Nem történik típuskonverzió.

Függvénysablonok túlterhelése

- Ha azt szeretnénk, hogy bizonyos típusú formális paraméterekre ne legyen létrehozva a függvénysablon egy példánya, hanem egy másik függvény legyen végrehajtva, akkor túlterhelhetjük a függvénysablont.
- Ez azt jelenti, hogy egy hagyományos függvényt definiálunk, amely a kívánt típusú formális paraméterekből álló listával rendelkezik.

Példa a függvénysablon túlterhelésére

```
#include <iostream>
#include <cmath>
using namespace std;
template <class T>
void csere(T& a, T& b) {
  T x = a;
  a = b;
  b = x;
  cout << "Fuggvenysablonnal.\n";
}
```



A függvénysablon túlterhelése

```
void csere(int& a, double& b) {
  int x;
  x = a;
  a = static_cast<int>(b);
  b = x + fabs(b - a);
  cout << "Sablon nelkul.\n";
}
// Az egész részeket cseréli ki.
// A törtrész változatlan marad.
```

A „kiir_cserel” függvénysablon

```
template <class T1, class T2>
void kiir_cserel(T1& a, T2& b) {
  cout << "Csere elott:\n";
  cout << "a =" << a << "\t";
  cout << "b =" << b << endl;
  csere(a, b);
  cout << "Csere utan:\n";
  cout << "a =" << a << "\t";
  cout << "b =" << b << endl;
} // T1 és T2 különböző lehet
```

A fő függvény

```
int main() {
 double y = 3.2;
 double z = 5.7;
 kiir_cserel(y, z);
 int t = 7;
 double w = 10.9;
 kiir_cserel(t, w);
}
```

Kimenet:

```
Csere elott:
a =3.2  b =5.7
Fuggvenysablonnal.
Csere utan:
a =5.7  b =3.2
Csere elott:
a =7    b =10.9
Sablon nelkul.
Csere utan:
a =10   b =7.9
```

13.2. Osztálysablonok

- Osztálysablonok definiálása
- Osztálysablonok objektumai
- Osztálysablonok konstruktora

Osztálysablonok definiálása

- Definíció:

```
template <paraméterek_listája>
class oszt {
    // ...
};
```

- A paraméterek listája típusparamétereket és konstansparamétereket tartalmazhat.
- A típusparamétert a class név alakban adjuk meg, a konstansparamétert pedig a típus név formában, egy hagyományos deklarációval.

A tagfüggvények definiálása

- Ha a tagfüggvényeket az osztályon kívül definiáljuk, akkor azokat függvénysablonként kell megadnunk, tehát a

  template <paraméterek_listája>

  el kell legyen helyezve a fejléc előtt.
- Az osztály neve után pedig <> jelek között meg kell adni a típusparaméterek és konstansparaméterek neveit.

Példa

```
template <class T, int I>
class verem {
  T t[I];
public:   // ...
  void betesz(T x);
};
template <class T, int I>
void verem<T, I>::betesz(T x) {
     // ...
}
```

Osztálysablonok objektumai

- Az osztálynévre mindig úgy hivatkozunk, hogy megadjuk a típusparamétereket és a konstansparamétereket.
- Egy objektum az

  osztálynév<paraméterek> objektum;

- alakban definiálható. Például:

```
verem<int, 10> v;
```

Példa osztálysablonra

- Egy veremre vonatkozó osztálysablont definiálunk, a betesz és kivesz tagfüggvényekkel. Ha a verem betelt, vagy üres, kivételt váltunk ki.

```
#include <iostream>
using namespace std;
```

A verem osztálysablon

```
template<class T, int I>
class verem {
  T t[I];
  int n;
public:
  class Betelt {}; // kivetel
  class Ures{};    // kivetel
  verem() { n = 0; }
  void betesz(T x);
  T kivesz();
  void kivesz_kiir();
};
```

A betesz tagfüggvény

```
template<class T, int I>
void verem<T,I>::betesz(T x)
{
  if ( n == I )
     throw Betelt();
  t[n++] = x;
}
```

A kivesz tagfüggvény

```
template<class T, int I>
T verem<T,I>::kivesz()
{
  if ( n == 0 )
     throw Ures();
  return t[--n];
}
```

A kivesz_kiir tagfüggvény

```
template<class T, int I>
void verem<T,I>::kivesz_kiir()
{
  while(n > 0)
     cout << t[--n] << " ";
  cout << endl;
}
```

A fő függvény (10 darab int típusú elem)

```
int main() {
  const int meret = 10;
  verem<int, meret> v;
  for(int i = 0; i < meret; i++)
     v.betesz(i);
  v.kivesz_kiir();
}
```

Kimenet: 9 8 7 6 5 4 3 2 1 0

Más fő függvény (5 darab double típusú elem)

```
int main() {
  const int meret2 = 5;
  verem<double, meret2> v2;
  for (int i = 0; i < meret2; i++)
    v2.betesz(i + static_cast<double>(i) / 10);
  v2.kivesz_kiir();
}
```

Kimenet:

4.4 3.3 2.2 1.1 0

Más fő függvény (kivételkezelés: betelt verem)

```
int main() {
  const int meret = 10;
  verem<int, meret> v;
  try {
     for (int i = 0; i <= meret; i++)
          v.betesz(i);
  }
  catch (verem<int, meret>::Betelt) {
     cout << "Betelt.\n";
  }
}
```

Kimenet:

Betelt.

Más fő függvény (kivételkezelés: üres verem)

```
int main() {
   const int meret = 10;
   verem<int, meret> v;
   for (int i = 0; i < meret; i++)
       v.betesz(i);
   try {
       while (true)cout << v.kivesz() << " ";
   }
   catch (verem<int, meret>::Ures) {
       cout << "Ures.\n";
   }
}
```

Kimenet: 9 8 7 6 5 4 3 2 1 0 Ures.

Osztálysablonok konstruktora

- Ha egy osztálysablon konstruktorát az osztálydeklaráción kívül definiáljuk, akkor a típusparaméterek és konstansparaméterek neveit csak az osztálynévben kell megadni. A tagfüggvénynévben nem kell ezeket megismételni. Például:

```
template <class T, int I>
verem<T, I>::verem()
{
    n = 0;
}
```