Objektum orientált programozás C++ nyelven
7. Kivétel kezelés

PEKÁRDY MILÁN – PANNON EGYETEM

PEKARDY@DCS.UNI-PANNON.HU

Kivételek

- A kivételek segítségével a program futása során előforduló „kivételes körülményekre” (pl. futásidejű hiba) tudunk reagálni
- Egy kivétel bekövetkezése esetén a program vezérlése egy speciális kivétel kezelő függvényhez kerül
- Kivételek elkapásához a vizsgálandó kódrészletet egy ún. try-blokkba tesszük, amikor a vizsgált blokkban eldobunk egy kivételt, akkor a vezérlés a kivétel kezelőnek adódik át
- Ha nincs kivétel, akkor a kivétel kezelők nem futnak le, a normál ágon folytatódik a végrehajtás

Kivételek

- Egy kivétel eldobásához a throw kulcsszót használjuk egy try blokkon belül
- A kivétel kezelő blokkok a catch kulcsszóval vannak deklarálva, ami közvetlenül követi a try blokkot
- A throw kulcsszó után egy paramétert írhatunk, ami az „eldobandó” értéket reprezentálja, ez az érték kerül átadásra a kivétel kezelőnek
- A catch blokk szintaktikája hasonló egy függvényéhez, ami egy paramétert vár

Kivételek

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

int main () {
  try
  {
    throw 20;
  }
  catch (int e)
  {
    cout << "An exception occurred. Exception Nr. " << e << '\n';
  }
  return 0;
}
```

Kivételek

- A catch blokkban a paraméter típusa kiemelt jelentőséggel bír: a throw után megadott érték típusát hasonlítja össze a catch-ben lévő típussal és ha a típusok egyeznek, akkor fog az adott kivétel kezelő aktiválódni
- Több catch blokk is megadható egymás után, mindegyik különböző paraméter típussal. Ilyenkor az hívódik meg, amelyiknek egyezik a paraméter típusa az eldobott érték típusával
- Ha a catch paraméterben (…)-t adunk meg, akkor ez az ág bármilyen típusú paramétert képes elkapni (alapértelmezett kivétel kezelőként használható, ami minden eldobott értéket elkap, amit nem kezelt korábbi catch ág)

Kivételek

```
try {
  // code here
}
catch (int param) { cout << "int exception"; }
catch (char param) { cout << "char exception"; }
catch (...) { cout << "default exception"; }
```

- Egy kivétel elkapása és kezelése után a végrehajtás a try-catch blokk után folytatódik, nem pedig a throw után
- A try-catch blokkok egymásba is ágyazhatók: ilyenkor azt is megtehetjük, hogy egy belső blokkból tovább dobjuk a kivételt a külső blokk felé (throw;)

```
try {
  try {
      // code here
  }
  catch (int n) {
      throw;
  }
}
catch (...) {
  cout << "Exception occurred";
}
```

Sztenderd kivételek

- A C++ sztenderd könyvtár tartalmaz egy ősosztályt (std::exception) amiből származtatva olyan objektumokat hozhatunk létre, amiket kivételként eldobhatunk (<exception> fejlécben)
- Egy virtuális what() metódusa van, ami egy char*-al tér vissza. A függvény felülírható a származtatott osztályokban, hogy az adott hibáról egy megfelelő leírással térjen vissza
- Az ősosztályból való származtatással saját kivétel típusokat hozhatunk létre, amelyek jellemzik a programunkban előforduló kivételes eseményeket és azokról megfelelő információt tárolnak

Sztenderd kivételek

- A catch blokkban referenciaként kapjuk el a kivételt így az esetleges származtatott kivétel típusok esetén is meghívódik a blokk és a megfelelő what() verzió fog lefutni

```
// using standard exceptions
#include <iostream>
#include <exception>
using namespace std;

class myexception: public exception
{
  virtual const char* what() const throw()
  {
    return "My exception happened";
  }
} myex;

int main () {
  try
  {
    throw myex;
  }
  catch (exception& e)
  {
    cout << e.what() << '\n';
  }
  return 0;
}
```

Sztenderd kivételek

exception	description
`bad_alloc`	thrown by `new` on allocation failure
`bad_cast`	thrown by `dynamic_cast` when it fails in a dynamic cast
`bad_exception`	thrown by certain dynamic exception specifiers
`bad_typeid`	thrown by `typeid`
`bad_function_call`	thrown by empty `function` objects
`bad_weak_ptr`	thrown by `shared_ptr` when passed a bad `weak_ptr`

Példa

```
try {
    cout << "foo" << endl;
    try {
        cout << "foo1" << endl;
        throw "invalid argument";
    }
    catch (int e) {
        cout << e << endl;
    }
    catch (double e) {
        cout << e << endl;
    }
    cout << "bar" << endl;
}
catch (int a) {
    cout << a << endl;
}
catch (string e) {
    cout << e << "?" <<endl;
}
catch (char * e) {
    cout << e << "!" << endl;
}
```

Példa

```
int fact(int n)
{
    if (n < 0) throw string("Invalid argument");
    int num = 1;
    for(int i = 2; i <=n; i++) num*= i;
    return num;
}
```

```
int main()
{
    try {
        cout << fact(4) << endl;
        cout << fact(-3) << endl;
    } catch (string e)
    {
        cout << e << endl;
    }
    return 0;
}
```