

# **Objektumorientált programozás**



## **Objektumalapú programozás a C++ programozási nyelvben**

**Kivételkezelés**

**Darvay Zsolt**

# **Osztályok, kivételkezelés és sablonok**



1. Az osztály fogalma
2. Kivételkezelés
3. Sablonok

## 12. Kivételkezelés



**throw 1;**

# Áttekintés



- ▶ 12.1. Kivételek kiváltása és kezelése
- ▶ 12.2. Különböző típusú kivételek
- ▶ 12.3. A new operátor és a kivételek
- ▶ 12.4. A try és catch blokkok elhelyezése
- ▶ 12.5. Kivételkezelők egymásba ágyazása

# 12.1. Kivételek kiváltása és kezelése



- ▶ Ha egy eseménynek (hibának) a lekezelését el szeretnénk halasztani, akkor egy kivételt válthatunk ki.
- ▶ A kivételek kiváltása (eldobása) a **throw** utasítással történhet.
- ▶ A **throw** kulcsszó után egy értéket kell megadjunk. Ez lehet egy konstans, vagy egy változó, esetleg egy objektum.
- ▶ A kivétel kezelését egy **try** és egy **catch** blokk segítségével végezzük.

# A try és catch blokkok

- ▶ A try - catch szerkezet:

`try összetett_utasítás kezelő_sorozat`

- ▶ A kezelő\_sorozat több egymás utáni kezelőből áll. Az egyes kezelők alakja:

`catch(kivétel_deklaráció)`

`összetett_utasítás`

- ▶ A kivétel\_deklaráció a függvények formális paramétereinek megadásához hasonló.

# Példa



- ▶ A **vektor** osztály esetén az **osszead** tagfüggvényt úgy módosítjuk, hogy különböző méret esetén egy kivételt váltson ki.

# Az osztálydeklaráció

```
class vektor {    int *elem;  
                  int dim;  
public:  
    vektor(int *e, int d);  
    vektor(const vektor &v); //masoló konstruktor  
    ~vektor() { delete [] elem; }  
    void negyzetreemel();  
    vektor osszead(vektor& v); // összeadás  
    void kiir();  
};
```



# A konstruktor



```
vektor::vektor(int *e, int d)
{
    elem = new int[d];
    dim = d;
    for(int i=0; i < dim; i++)
        elem[i] = e[i];
}
```

# A másoló konstruktor



```
vektor::vektor(const vektor &v)
{
    dim = v.dim;
    elem = new int[dim];
    for(int i=0; i < dim; i++)
        elem[i] = v.elem[i];
}
```

# A negyzetreemel tagfüggvény és a destruktork

```
void vektor::negyzetreemel()
{
    for(int i = 0; i < dim; i++)
        elem[i] *= elem[i];
}
```

- ▶ A destruktort a struktúrán belül definiáltuk, ezért ez inline függvény lesz.

# Az összead tagfüggvény

```
vektor vektor::osszead(vektor& v) {  
    if (dim != v.dim)  
        throw "Kulonbozo meret";  
    int* x = new int[dim];  
    for(int i = 0; i < dim; i++)  
        x[i] = elem[i] + v.elem[i];  
    vektor t(x, dim);  
    delete [] x;  
    return t;  
}
```

# A kiír függvény



```
void vektor::kiir()
{
    for(int i = 0; i < dim; i++)
        cout << elem[i] << '\t';
    cout << endl;
}
```

# Fő függvény (VektX)

```
int main() {  
    int x[20];  
    int meretX;  
    cout << "az x tomb merete = ";  
    cin >> meretX;  
    for(int i = 0; i < meretX; i++)  
        x[i] = 2 * i + 1;  
    vektor VektX(x, meretX);  
    cout << "az x vektor : ";  
    VektX.kiir();  
}
```

# Fő függvény (VektY)

```
int y[20];  
int meretY;  
cout << "az y tomb merete = ";  
cin >> meretY;  
for(int i = 0; i < meretY; i++)  
    y[i] = 2 * i + 2;  
vektor VektY(y, meretY);  
cout << "az y vektor : ";  
VektY.kiir();
```

# Fő függvény (try és catch)

```
try {  
    cout << "az osszeg: ";  
    VektX.osszead(VektY).kiir();  
    cout << "Nincs hiba\n";  
}  
catch(const char *s) {  
    cout << "Hiba! " << s << endl;  
}  
}
```



# Kimenet (azonos méret)



az x tomb merete = 5

az x vektor : 1 3 5 7 9

az y tomb merete = 5

az y vektor : 2 4 6 8 10

az osszeg: 3 7 11 15 19

Nincs hiba

# Kimenet (különböző méret)



az x tomb merete = 9

az x vektor : 1 3 5 7 9 11 13 15 17

az y tomb merete = 7

az y vektor : 2 4 6 8 10 12 14

az osszeg: Hiba! Kulonbozo meret

## 12.2. Különböző típusú kivételek

- ▶ Egy **try** blokkhoz több **catch** blokk is tartozhat.
- ▶ Ekkor az első olyan **catch** blokkhoz kerül a vezérlés, amely el tudja kapni a kivételt.
- ▶ Minden kivételt elkap a **catch ( ... ) összetett\_utasítás**
- ▶ A **...** a szintaxis része.
- ▶ A **catch** blokkok sorrendje fontos!

# A kivétel elkapása

- ▶ Jelölések:
  - ▶ T – a kiváltott kivétel típusa
  - ▶ C – a kezelő kivétel\_deklarációjában szereplő típus
- ▶ A kezelő el tudja kapni a kivételt, ha az alábbiak közül az egyik teljesül:
  - I. T és C megegyezik, vagy C alaposztálya T-nek
  - II. T és C mutató típus és a hivatkozott típusokra az I. fennáll.
  - III. C egy referencia és a típusára I. fennáll.

# Példa (verem)



```
#include <iostream>
```

```
#define MAX_ELEM 15
```

```
using namespace std;
```

```
// legtöbb MAX_ELEM darabot tárolhatunk
```

```
// a verem elemei int típusúak
```

# A verem osztály



```
class verem {  
    int t[MAX_ELEM];  
    int n;  
public:  
    class Betelt {}; // kivétel  
    class Ures{};    // kivétel  
    verem() { n = 0; }  
    void betesz(int x);  
    int kivesz();  
};
```

# A betesz tagfüggvény

```
void verem::betesz(int x)
{
    // Betelt b;
    // if ( n == MAX_ELEM ) throw b;
    if ( n == MAX_ELEM ) throw Betelt();
    t[n++] = x;
}
```

# A kivesz tagfüggvény



```
int verem::kivesz()  
{  
    if ( n == 0 ) throw Ures();  
    return t[--n];  
}
```



# Fő függvény



```
int main()
{
    int meret;
    cout << "meret = ";
    cin >> meret;
    verem v;
    cout << "Betesz: ";
```

# Betesz (try blokk)



```
try {  
    for(int i = 0; i < meret; i++)  
    {  
        v.betesz(i);  
        cout << i << " ";  
    }  
    cout << "\nNem telt be\n";  
}
```

# Betesz (catch blokk)



```
catch( verem::Betelt )  
{  
    cout << "\nBetelt.\n";  
}
```

# Kivesz (try blokk)



```
cout << "Kivesz: ";  
try {  
    while( true )  
        cout << v.kivesz() << " ";  
    cout << "\nNem ures.\n"; // nem  
                                // hajtja  
    vegre  
}
```

# Kivesz (catch blokk)



```
catch( verem::Ures )  
{  
    cout << "\nUres.\n";  
}  
} // fő függvény vége
```

# Kimenet (nem telik be)



meret = 10


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

Nem telt be

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

Ures.

# Kimenet (betelik)



meret = 20

Betesz: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

Betelt.

Kivesz: 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Ures.

## 12.3. A new operátor és a kivételek

- ▶ Ha nincs elegendő memória
  - ▶ a C++ régebbi változataiban (a Visual C++ 6.0-ban is) a **new** operátor zérust térített vissza.
  - ▶ a C++ szabványnak megfelelően egy **std::bad\_alloc** kivételt vált ki (Visual C++ .NET 2002-től kezdődően). Ha mégsem akarunk kivételt kiváltani, akkor a **new(std::nothrow)** alakot használhatjuk.



# 12.4. A try és catch blokkok elhelyezése

- ▶ Mekkora legyen a try blokk nagysága?
  - ▶ sok kód egyetlen try blokkban – kevesebb kezelő;
  - ▶ kevesebb kód több try blokkban – több kezelő.
- ▶ Ha egy kivétel hatására több blokkból lépünk ki, hogyan szabadulnak fel az automatikus objektumok?
  - ▶ Verem visszagombolyítása (a destruktorok meghívásának sorrendje ellentétes a létrehozás sorrendjével);
  - ▶ a catch blokk végrehajtása előtt kerül rá sor.

## 12.5. Kivételkezelők egymásba ágyazása

- ▶ Először a legbelsőbb try blokknak megfelelő kezelőket vizsgáljuk meg.
- ▶ Továbbdobás (catch blokkban):  
    throw;
- ▶ Példa

# Kivétel kiváltása



```
#include <iostream>
using namespace std;
class ZerusHiba{};
double f(double x) {
    if ( x == 0.0 )
        throw ZerusHiba();
    return 1.0 / x;
}
```

# Függvényben: try blokk

```
double g(char *s, double x) {  
    double y = 0.0;  
    cout << s;  
    try { y = f(x); }  
    catch( ZerusHiba ) {  
        cout << "Hiba g-ben: ";  
        throw;  
    }  
    return y;  
}
```

# A g függvény meghívása

```
int main() {  
    double szam;  
    cout << "A szam = ";  
    cin >> szam;  
    try {  
        cout << g("Inverz: ", szam) << endl;  
    }  
    catch( ZerusHiba ) {  
        cout << "Zerussal valo osztas.\n";  
    }  
}
```

# Az f függvény meghívása

```
try {  
    cout << "Inverz: ";  
    cout << f(szam) << endl;  
}  
catch( ZerusHiba )  
{  
    cout << "Zerussal valo osztas.\n";  
}  
}
```

# Kimenet (továbbdobással)



A szám = **0**

Inverz: Hiba g-ben: Zerussal valo osztas.

Inverz: Zerussal valo osztas.

# Kimenet (továbbdobás nélkül)

- ▶ Ha a **g** függvényből a **throw;**
- ▶ utasítást töröljük, akkor a kimenet:  
A szam = **0**  
Inverz: Hiba g-ben: 0  
Inverz: Zerussal valo osztas.