Informatika 3

Výnimky

9

Čo je to výnimka

- Reprezentuje nejakú ***zlú*** udalosť
- Predstavuje nezvyčajný a neočakávaný stav
- Môžu byť spôsobené externými faktormi a faktormi prostredia

Ako ošetrovať chyby

- Program sa ukončí.
- Funkcie, detekujúce výnimku návratovou hodnotou - stavová hodnota, indikujúca nežiadúci stav; globálny indikátor errno a perror
- Vytvorenie spätne volanej funkcie na ošetrenie chýb, ktorú budú pri výskyte chyby volať nízko-úrovňové funkcie.
- Vykonanie vzdialeného skoku – setjmp, longjmp zo štandardnej C knižnice
  - setjmp – uloží sa dobrý stav programu
  - longjmp – obnoví sa tento stav
- Signály - funkcie signal a raise so štandardnej C knižnice (asynchrónne)

- Ani jeden nevie volať deštruktory
- Pre C++ sa používajú výnimky

Ukončenie programu – únik pamäti

```
#pragma once
#include <crtdbg.h>

#ifdef _DEBUG
  #define DBGNEW new (_NORMAL_BLOCK, __FILE__, __LINE__)
#else
  #define DBGNEW new
#endif

class MLeak
{
public:
    ~MLeak()
    {
        _CrtDumpMemoryLeaks();
    }
};

const MLeak dummy;
```

Ukončenie programu – únik pamäti

```
#pragma once
#include <crtdbg.h>

#ifdef _DEBUG
  #define DBGNEW new (_NORMAL_BLOCK, __FILE__, __LINE__)
#else
  #define DBGNEW new
#endif

class MLeak
{
public:
    ~MLeak()
    {
        _CrtDumpMemoryLeaks();
    }
};

const MLeak dummy;
```

Ukončenie programu – protokol

```cpp
#pragma once
#include <ctime>
#include <fstream>
#include <iostream>

using namespace std;

class OFStream
{
private:
    ofstream logSubor;
public:
    OFStream(const char* meno)
    {
        logSubor.open(meno);
    }
    void Write(const char* text)
    {
        time_t szClock;
        time(&szClock);
        string cas(asctime(localtime(&szClock)));
        cas = cas.substr(0, cas.length() - 1);
        logSubor << cas << ": " << text << endl;
    }
    bool IsOpen() const
    {
        return logSubor.is_open();
    }
    ~OFStream()
    {
        logSubor << "KONIEC" << endl;
        logSubor.close();
        cout << "Protokol je zatvoreny: " <<
                (IsOpen() ? "Nie" : "Ano") << endl;
    }
};

extern OFStream* LogSubor;
```

- Len deklarácia OFStream * LogSubor

Code	Description
LNK2001	unresolved external symbol "class OFStream * LogSubor" (?LogSubor@@3PAVOFStream@@A)
LNK1120	1 unresolved externals

Ukončenie programu – Tovar

```
class Tovar
{
private:
    string nazov = "";
    int hmotnost = 0;
    static int celkovaHmotnost;
    static int celkovyPocet;
public:
    Tovar(const char* pnazov, int hmotn)
        : nazov(pnazov ? pnazov : ""), hmotnost(hmotn)
    {
        celkovaHmotnost += hmotn;
        celkovyPocet++;
    };

    ~Tovar() {
        celkovaHmotnost -= hmotnost;
        celkovyPocet--;
        cout << "Dealokujem '" << nazov << "'" << endl;
        nazov.clear();
    };
```

Ukončenie programu – štandardný main

```
void Info()
{
    Tovar* tovar = DBGNEW Tovar("auto", 1000);
    Tovar kladivo("kladivo", 10);
    delete tovar;
}


OFStream* LogSubor = nullptr;

int main(void)
{
    LogSubor = new OFStream("MojaApp.log");
    LogSubor->Write("Start programu");
    Info();
    LogSubor->Write("Koniec programu");
    delete LogSubor;
}
```

Microsoft Visual Studio Debug Console

```
Dealokujem 'auto'
Dealokujem 'kladivo'
Protokol je zatvoreny: Ano
```

MemLeak.h | MojaApp.log | cDatum.h | mm.cpp

```
Thu Nov 23 08:51:45 2023: Start programu
Thu Nov 23 08:51:45 2023: Koniec programu
KONIEC
```

```
'Spec.exe' (Win32): Loaded 'D:\Vyuka\2023_24\Inf3\SpecialnePrvky\Spec\Debug\Spec.exe'. Symbols loaded.
'Spec.exe' (Win32): Loaded 'C:\Windows\SysWOW64\ntdll.dll'.
'Spec.exe' (Win32): Loaded 'C:\Windows\SysWOW64\kernel32.dll'.
'Spec.exe' (Win32): Loaded 'C:\Windows\SysWOW64\KernelBase.dll'.
'Spec.exe' (Win32): Loaded 'C:\Windows\SysWOW64\msvcp140d.dll'.
'Spec.exe' (Win32): Loaded 'C:\Windows\SysWOW64\vcruntime140d.dll'.
'Spec.exe' (Win32): Loaded 'C:\Windows\SysWOW64\ucrtbased.dll'.
The thread 0x5570 has exited with code 0 (0x0).
'Spec.exe' (Win32): Loaded 'C:\Windows\SysWOW64\kernel.appcore.dll'.
'Spec.exe' (Win32): Loaded 'C:\Windows\SysWOW64\msvcrt.dll'.
'Spec.exe' (Win32): Loaded 'C:\Windows\SysWOW64\rpcrt4.dll'.
The thread 0x26d4 has exited with code 0 (0x0).
The thread 0x1904 has exited with code 0 (0x0).
The program '[10176] Spec.exe' has exited with code 0 (0x0).
```

KST

Ukončenie programu – exit

```
void Info()
{
    Tovar* tovar = DBGNEW Tovar("auto", 1000);
    Tovar kladivo("kladivo", 10);
    delete tovar;
    exit(0);
}

OFStream* LogSubor = nullptr;

int main(void)
{
    LogSubor = new OFStream("MojaApp.log");
    LogSubor->Write("Start programu");
    Info();
    LogSubor->Write("Koniec programu");
    delete LogSubor;
}
```

Microsoft Visual Studio Debug Console

```
Dealokujem 'auto'
```

MemLeak.h | MojaApp.log | cDatum.h | mm.cpp

```
Thu Nov 23 08:57:11 2023: Start programu
```

```
The thread 0x6e78 has exited with code 0 (0x0).
Detected memory leaks!
Dumping objects ->
{252} normal block at 0x00891378, 8 bytes long.
 Data: <  V     > 08 FD 56 00 00 00 00 00
{159} normal block at 0x00890E90, 176 bytes long.
 Data: < q   p          > 14 71 DB 00 C0 70 DB 00 00 00 00 00 00 00 00 00
Object dump complete.
```

Ukončenie programu – atexit

```
void Info()
{
    Tovar* tovar = DBGNEW Tovar("auto", 1000);
    Tovar kladivo("kladivo", 10);
    delete tovar;
    exit(0);
}

OFStream* LogSubor = nullptr;

int main(void)
{
    atexit(ExitFun);
    LogSubor = new OFStream("MojaApp.log");
    LogSubor->Write("Start programu");
    Info();
}

void ExitFun()
{
    LogSubor->Write("Koniec programu");
    if (LogSubor != nullptr)
    {
        delete LogSubor;
        LogSubor = nullptr;
    }
}
```

Microsoft Visual Studio Debug Console

```
Dealokujem 'auto'
Protokol je zatvoreny: Ano
```

MemLeak.h | MojaApp.log | cDatum.h | mm.cpp

```
Thu Nov 23 09:00:05 2023: Start programu
Thu Nov 23 09:00:05 2023: Koniec programu
KONIEC
```

```
Detected memory leaks!
Dumping objects ->
{252} normal block at 0x00E81570, 8 bytes long.
 Data: <@ {     > 40 F6 7B 00 00 00 00 00
Object dump complete.
```

Návratová hodnota

```
bool Info2(Tovar* tovar);
bool Info3(Tovar* tovar);
bool Info4(Tovar* tovar);

bool Info()
{
    Tovar* tovar = DBGNEW Tovar("auto", 1000);
    Tovar kladivo("kladivo", 10);
    delete tovar;
    if (tovar->Hmotnost() > 500) {
        if (Info2(tovar)) {
            if (Info3(tovar)) {
                if (Info4(tovar)) {
                    return false;
                }
            }
        }
    }
    return true;
}

bool Info2(Tovar* tovar)
{
    return tovar->Hmotnost() > 400 ? false : true;
}

bool Info3(Tovar* tovar)
{
    return tovar->Hmotnost() > 300 ? false : true;
}

bool Info4(Tovar* tovar)
{
    return tovar->Hmotnost() > 200 ? false : true;
}
```

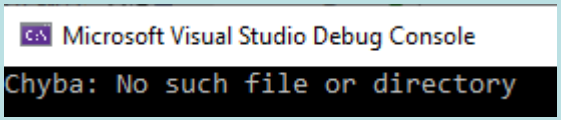
```
int main(void)
{
    if (Info()) {
    }
    return 0;
}
```

Nastavenie globálného indikátora - errno

https://learn.microsoft.com/en-us/cpp/c-runtime-library/errno-constants?view=msvc-170

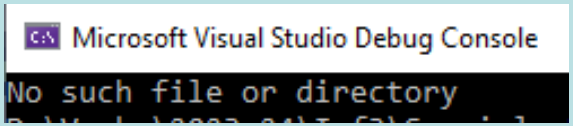
```c
int main(void)
{
    FILE* fh = nullptr;

    if (!(fh = fopen("NOSUCHF.ILE", "rb")) != 0)
    {
        perror("Chyba");
    }
    else
    {
        printf("open succeeded on input file\n");
        fclose(fh);
    }
}
```




```c
int main(void)
{
    FILE* fh = nullptr;

    if (!(fh = fopen("NOSUCHF.ILE", "rb")) != 0)
    {
        cout << strerror(errno);
    }
    else
    {
        printf("open succeeded on input file\n");
        fclose(fh);
    }
}
```



Spätne volaná funkcia - callback

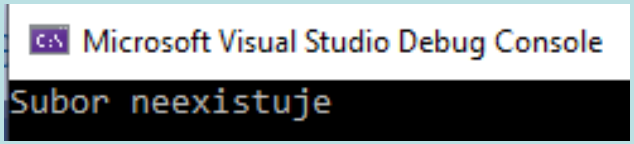
```
typedef void (ChybaFunPtr)(const char* oznam, ostream* of);

void ChybaFun(const char* text, ostream* of)
{
    if (of != nullptr)
        *of << text << endl;
    else
        cout << text << endl;
}

void Citaj(ChybaFunPtr callbackOnError)
{
    FILE* fh = nullptr;

    if (!(fh = fopen("NOSUCHF.ILE", "rb")) != 0)
    {
        callbackOnError("Subor neexistuje", &cout);
    }
    else
    {
        // ...Citam data...
        fclose(fh);
    }
}

int main(void)
{
    Citaj(ChybaFun);
}
```



Vzdialený skok

```
class A {
  public:
    A() { cout << "A()" << endl; }
   ~A() { cout << "~A()" << endl; }
};

jmp_buf jbuf;

void fun()
{
  A a;
  cout << "fun()\n";
  longjmp(jbuf,1);
}
```

```
int main()
{
  if(setjmp(jbuf) == 0) {
    cout << "setjmp=0...\n";
    fun();
  } else {
    cout << "setjmp!=0..." << endl;
  }
  return 0;
}
```

Signál

```cpp
#include <csignal>
#include <iostream>
#include <cstdlib>

using namespace std;
void CtrlBreak(int signal)
{
    cout << endl << "Zachyteny signal: " << signal << endl;
}

int main(void) {
    signal(SIGBREAK, CtrlBreak);
    int pocet = 1;
    while (1)
    {
        for (int i = 0; i < pocet - 1; i++)
            cout << 'A';
        pocet++;
        cout << endl;
    }
    return 0;
}
```

OO ošetrovanie výnimiek

- Ošetrenie výnimiek by malo byť vlastnosťou programovacieho jazyka
- Spôsob ošetrovania výnimiek by mal byť OOP orientovaný
- Ošetrovanie výnimiek musí byť flexibilné, musí podporovať čo najviac najbežnejších typov výnimiek a ich ošetrenie
- Mechanizmus ošetrovania výnimiek musí byť prekrývateľný programátorom

Základný problém

- Indikácia výnimky
- Ošetrenie výnimky

Vyslanie výnimky

- Výnimka sa vyvoláva kľúčovým slovom **throw**

```
#include <iostream.h>

const int CHYBA = -1;

void Fun(unsigned char *data,long size)
{
     if(size >1000)
     // nedokážem spracovať tak veľa dát
     throw CHYBA;
     // ... spracovanie dát
}
```

Zachytenie výnimky

```
try {
    /* riadený blok */
}
catch(Vynimka v) {
    /* ošetrenie výnimky */
}
catch(Vynimka2 v2) {
    /* ošetrenie inej výnimky */
}
```

- Ak výnimka nie je zachytená, prejde sa do ďalšej obaľujúcej sféry
- Porovnávanie
  - Skočí do catch bloku podľa typu výnimky
  - typ musí byť rovnaký alebo potomkom typu parametra catch bloku
  - Nevykonávajú sa konverzie jedného typu na iný
  - V catch môže byť odkaz na objekt (nevolá sa copy-konštruktor)

Príklad 1

```
// Výnimky
class A {
  public:
  A() { cout << "A()" << endl; }
 ~A() { cout << "~A()" << endl;
}
};

void Fun()
{
  A a;
  for(int i = 0; i < 3; i++)
    cout << "fun()\n";
  throw 47;
}
```

```
int main() {
  try {
    cout << "main()" << endl;
    Fun();
  }
  catch (int) {
    cout << "catch(int) "<< endl;
  }
}
```

Príklad 2

```
class Vynimka{
    char* text;
public:
    Vynimka(char* txt) { text=txt; }
    char* Preco() { return text; }
}
double vydel(double x, double y){
    if(y==0)
        throw Vynimka(“Delenie nulou”);
    return x / y;
}

main()
{   …
    try{
            int a=3, b=0;
            double x=vydel(a, b); // nastane výnimka a odvíjanie zásobníka
            …
    }
    catch(Vynimka& v){
            cout << v.Preco() << ‘\n’;
    }
}
```

Príklad 3

```
class Chyba {
};

class Varovanie : public Chyba {
};

class Fatal : public Chyba {
};

class A {
  public:
    void Fun() { throw Fatal(); }
};
```

```
int main() {
  A a;
  try {
    a. Fun ();
  }
  catch(Chyba &) {
    cout << "Chyba" << endl;
  }
  // Skryté
  catch(Varovanie &) {
    cout << "Varovanie" << endl;
  }
  catch(Fatal &) {
    cout << "Fatal" << endl;
  }
}
```

Modely ošetrenia výnimiek

- ukončenie – používa C++
- obnovenie – ak potrebujeme musíme si to naprogramovať

```
int main() {
  bool ok = true;
  while(ok==true) {
      try {
        throw 50;
      }
      catch(int x) {
         // OPRAVA
        if(Skoncit==ANO)
          ok=false;
      }
    }
}
```

Pravidlá používania výnimiek

- Každú výnimku treba zachytiť – neošetrená výnimka je programová chyba
- Nezachytená výnimka vyvoláva terminate() - volá abort()
  - dá sa zmeniť pomocou set_terminate()
- V deštruktore negenerovať výnimku (odporúčanie)
  - vo výnimke môže nastať ďalšia výnimka
  - V deštruktore lokálnej premennej pri odvíjaní zásobníka výnimka volá terminate()
- V konštruktore byť opatrný – ak nastala výnimka v konštruktore, deštruktor tohto objektu sa nevykoná
- Zachytávať výnimky odkazom
- Používať potomkov štandardných výnimiek
- Zachytenie akejkoľvek výnimky:

```
catch(…){
   cout << “Výnimočný stav\n”;
   throw; // prevyslanie výnimky
}
```

Obalová trieda

```
class A {
    int *Pole_d;
  public:
    A() {
      cout << "A()" << endl;
      Pole_d=new int[100];
      rob_nieco();
    };
    void rob_nieco() {
      Pole_d[0]=1;
      throw 47;
    };
   ~A() {
      cout << "~A()" << endl;
      delete []Pole_d;
    };
};
```

```
int main() {
  try {
    A ur;
  }
  catch(int) {
    cout<<"Obsluha vynimky"<<endl;
  }
}
```

A()
Obsluha vynimky

Obalová trieda - pokračovanie

```
class Obal {
    int *Pole_d;
  public:
    Obal(int vel) {
      cout << "Obal() - alokujem 'vel' int" << endl;
      Pole_d=new int[vel];
    };
   ~Obal() {
      cout << "~Obal() - dealokujem 'vel' int" << endl;
      delete [] Pole_d;
    };
    operator int *() { return Pole_d; };
};
```

Obalová trieda - použitie

```
class A {
    Obal Pole_d;
  public:
    A() : Pole_d(100) {
      cout << "A()" << endl;
      rob_nieco();
    };
    void rob_nieco() {
      Pole_d[0]=1;
      throw 47;
    };
   ~A() {
      cout << "~A()" << endl;
    };
};
```

```
int main() {
  try {
    A ur;
  }
  catch(int) {
    cout <<"Obsluha vynimky"<< endl;
  }
}
```

Obal() - alokujem 'vel' int
A()
~Obal() - dealokujem 'vel' int
Obsluha vynimky

Výnimky na úrovni funkcií

```
double vydelPole(double* pole, double y, int n)
try
{
  for(int i=0 ; i<n ; i++){
        pole[i]=vydel( pole[i], y);
  }
}
catch(Vynimka v)
{
  cout<<v.Preco()<<'\n';
}
```

- To isté platí aj pre metódy
- Často sa používa v konštruktoroch

Výnimka na úrovni funkcie

```
class cZakladnaTrieda {
  int i;
public:
  class cVynimka {};

  cZakladnaTrieda(int i) : i(i) {
    throw cVynimka();
  }
};
```

Výnimka na úrovni funkcie-pokr.

```
class cOdvodenaTrieda : public cZakladnaTrieda {
public:
  class cVynimkaOdv {
    const char* msg;
  public:
    cVynimkaOdv(const char* msg) : msg(msg) {}
    const char* Oznam() const {
      return msg;
    }
  };
  cOdvodenaTrieda(int j) // Konštruktor
  try
    : cZakladnaTrieda(j) {
    // Telo konštruktora
    cout << "Toto by sa nemalo vytlačit" << endl;
  }
  catch (cVynimka &) {
    throw cVynimkaOdv("cZakladnaTrieda subobjekt vyslal vynimku");;
  }
};
```

Špecifikácia výnimky

- Môžeme definovať, ktoré výnimky metóda generuje:

```
class complex {
   double r, i;
public:
   complex vydel(double delitel) throw(Vynimka, Vynimka2) { … }
};
```

- Ak throw(v1,v2,v3) nie je uvedené, metóda môže vyhodiť ľubovoľnú výnimku.
- throw( ) – metóda nevyhadzuje žiadnu výnmku
- unexpected() – funkcia {nastavujeme cez set_unexpected()}, ktorá sa volá ak metóda vyhodí inú výnimku ako je uvedená v hlavičke – štandardne volá terminate() {nastavujeme cez set_terminate()}
- U šablón je deklarácia výnimiek prakticky nepoužiteľná

Štandardné výnimky

- exception – základná trieda
- logic_error – chyby programovacej logiky (napr. nesprávny parameter)
- runtime_error – chyba hardvéru, vyčerpanie pamäte

Používanie výnimiek

- Nie pre asynchrónne udalosti
- Nie pre mierne chybové podmienky
- Nie pre riadenie priebehu programu

- Vyriešenie problému a opätovné zavolanie funkcie ktorá výnimku spôsobila.
- Zorganizovanie vecí a pokračovanie bez opätovného volania funkcie.
- Výpočet nejakého alternatívneho výsledku namiesto toho, čo mala poskytovať funkcia.
- Urobenie hocičoho v aktuálnom kontexte a znovu vyslanie ***tej istej*** výnimky do vyššieho kontextu.
- Urobenie hocičoho v aktuálnom kontexte a znovu vyslanie ***odlišnej*** výnimky do vyššieho kontextu.
- Ukončenie programu
- Obaľovacie funkcie (najmä C knižničné funkcie, ktoré používajú obyčajnú chybovú schému tak, aby poskytovali výnimky).
- Zjednodušenie. Ak schéma výnimiek robí veci komplikovanejšími, nie je vhodné používať ju.
- Vytvorenie bezpečnejšej knižnice a programu. Toto je dlhodobá investícia (odolnosť aplikácie).

Pravidlá

- Začnime so štandardnými výnimkami
- Vnárajme svoje vlastné výnimky
- Používajme hierarchie výnimiek
- Zachytávajme odkazom, nie hodnotou
- Vysielajme výnimky v konštruktoroch
- Negenerujme výnimky v deštruktoroch