Лекция 9. Обработка исключительных ситуаций

**Исключительной ситуацией или исключением** (exception) называется прерывание нормального хода выполнения программы в случае возникновения непредвиденного или аварийного события. Исключительные ситуации могут генерироваться программно.

Существуют **два уровня обработки исключений**: средствами языка C++ и структурная обработка исключений (средствами языка операционной системы; structured exception handling).

**Обработка исключений средствами языка С++**

Для реализации механизма обработки исключительных ситуаций введены следующие **ключевые слова**:

1) **try** - начало блока исключения.

try

{операторы}

Позволяет определить контролируемый блок, внутри которого помещаются операторы, выполнение которых может вызвать ошибку, а также специальные операторы для генерации исключений.

2) **catch** - начало блока, ловящего исключения.

catch(тип\_исключения имя)

{операторы обработки исключения}

Если используются несколько обработчиков исключений, то они должны отличаться друг от друга типами исключений. Обработчик исключений имеет **три формы**:

1. сatch(тип\_параметра имя\_параметра)

Данная форма использует информацию об обрабатываемом исключении.

2. catch (тип\_параметра)

Для обработчика важен только тип исключения и факт его получения. Если в контролируемом блоке исключения не возникает, то обработчики исключений не проверяются. При возникновении исключения выполняются следующие действия:

-создаётся копия объекта, представляющего выражение оператора throw;

-вызываются деструкторы объектов, порождённых от начала контролируемого блока до точки выброса исключений;

-осуществляется поиск подходящего блока исключения, который выбирается по типу выражения в операторе throw, и происходит передача управления выбранному блоку;

-после выполнения оператором блока обработки исключения управление передаётся первому оператору, расположенному за последним блоком catch контролируемого блока;

-если поиск подходящего блока обработки исключения в контролируемом блоке окончился неудачно, то поиск осуществляется в блоках верхнего уровня вложенности (для перехода на предшествующий уровень может быть использован оператор throw без параметра);

-при неудачном поиске блока обработки исключений программа завершается аварийно.

3. catch (...)

Обработчик реагирует на любое исключение независимо от типа. Так как обработка исключений осуществляется последовательно, то данная форма обработчика ставится в конце блока обработки.

Объекты исключений, описывающие ошибку, создаются и затем генерируются с ключевым словом**throw**:

throw (выражение для генерации исключения)

Исключение создаётся как статический объект, тип которого определяется типом значения выражения для генерации исключения. Местоположение оператора throw называется **точкой выброса исключения**. В контролируемом блоке может быть любое количество операторов throw. Контролируемые блоки могут быть вложенными. После формирования исключения управление передаётся за пределы контролируемого блока, где обязательно должны находиться один или несколько обработчиков исключений.

Пример. Создать массив, содержащий не более пяти целых чисел из диапазона [10, 100].

#include <iostream>

using namespace std;

int main()

{

int a[10], i, n;

try

{

cout<<"Vvedite n"<<endl;

cin>>n;

if(n>5) throw "n is more than 5";

for(i=0; i<n; i++)

{

cout<<"Vvedite a["<<i<<"]:"<<endl;

cin>>a[i];

if(a[i]<10||a[i]>100)

throw"Value is not within the range";

}}

catch(char s[30])

{

cerr<<"Error: "<<s<<endl;

}

return 0;

}

Пример. Вычислить частное двух положительных чисел.

int main()

{

double a, b;

double s;

try{

cout<<"Vvedite a, b"<<endl;

cin>>a>>b;

if(a<0) throw"a<0";

if(b<0) throw"b<0";

if(b==0) throw 0;

s=a/b;

cout<<"Res="<<s<<endl;

}//Конец контролируемого блока

catch(int){

cerr<<"Error! Division by zero"<<endl;

}

catch(char s[20]){

cerr<<"Error!"<<s<<endl;

}

catch(…){

cerr<<"Error!"<<endl;}

return 0;

}

Если генерируются исключение, для которого отсутствует инструкция catch, то происходит аварийное завершение программы. В этом случае вызывается функция **terminate()**, которая по умолчанию вызывает функцию

abort() для завершения программы. Также функция terminate() может использовать функцию, назначенную в программе. Для этого используется следующая конструкция:

typedef void(\* тип\_функции)();

тип\_функции

set\_terminate(тип\_функции имя\_функции);//Устанавливает функцию для вызова в случае возникновения необработанной исключительной ситуации.

Пример. Программа, использующая set\_terminate.

#include <iostream>

//#include <eh.h>

using namespace std;

void MyErr()

{

cerr<<"Error processing"<<endl;

exit(1);

}

int main()

{

set\_terminate (MyErr);

try{

throw 0;

}

catch(char)

{

cout<<"Error type - char";

}

return 0;}

**Структурированная обработка исключений (structured exception handling, SEH)**

При структурированной обработке исключений компилятор генерирует специальный код на входах и выходах блока исключений. Создаёт таблицы вспомогательных структур данных и предоставляет системе возможность прохода по блокам исключений. Структурированное управление исключениями позволяет на ряду с обработкой потока, явно порождённого программой исключений, обрабатывать также и исключения, порождаемые операционной системой в аварийный ситуациях.

**Формат обработчика исключений**

\_\_try

{

//защищённый участок программы

}

\_\_exept(фильтор исключений)

{

//обработчик исключений

}

\_\_finally

{

//операторы, выполняющиеся в любом случае

}

Блок **\_\_exept** активизируется только в случае возникновения исключительной ситуации (исключения), а блок **\_\_finally** активизируется при любом выходе из защищенного участка программы, что гарантирует обязательное выполнение расположенных в блоке операторов (вне зависимости от исключения). Блок исключений принимает решения по продолжению процесса на основании анализа, вычисляемого после порождения исключения константного выражения.

В качестве фильтра исключений могут использоваться следующие константные выражения (**фильтры исключений):**

1) EXСEPTION\_EXECUTE\_HANDLER - указывает на возможность обработки исключения (блока except()).

2) EXСEPTION\_CONTINUE\_EXECUTION - указывает на исправление ошибки; в этом случае система передаёт управление на инструкцию, которая вызвала исключение, так как считается, что теперь не возникнет исключительная ситуация.

3) EXСEPTION\_CONTINUE\_SEARCH - указывает на то, что ведётся поиск подходящего обработчика, что свидетельствует о том, что ошибка не обработана.

Пример. Вынести на экран строку, используя указатель.

#include <iostream>

#include <windows.h>

using namespace std;

int main(){

char \*a=NULL, s[10]="SEH";

\_\_try

{

cout<<a<<endl;//Ошибка!

}

\_\_exept(EXCEPTION\_EXECUTE\_HANDLER)

{

a=s;

cout<<a<<endl;

}

return 0;}

4) EXСEPTION\_ARRAY\_BIUNDS\_EXCEEDED - попытка обращения к элементу массива, индекс которого выходит за пределы массива, если поддерживается такой вид контроля.

5) EXСEPTION\_STACK\_OVERFLOW - переполнение стека.

6) EXСEPTION\_INT\_DIVIDE\_BY\_ZERO - попытка деления на ноль для операций с целыми числами.

7) EXСEPTION\_INT\_OVERFLOW - переполнение с операциями целого типа.

8) EXСEPTION\_FLT\_DIVIDE\_BY\_ZERO - попытка деления на ноль при операциях с вещественными числами.

9) EXСEPTION\_FLT\_INVALID\_OPERATION - ошибка при выполнении операций с плавающей точкой, для которых не предусмотрены другие коды исключений.

10) EXСEPTION\_FLT\_OVERFLOW - переполнение при выполнении операций с действительными числами.

Детальную информацию о причине возникновения ошибки можно получить, используя функцию **DWORD GetExceptionCode()**, которую можно вызвать только из фильтра или обработчика исключений.

Для явного порождения исключения используется функция VOID RaiseException (DWORD dwExceptionCode, DWORD dwExceptionFlags, DWORD nNumberOfArguments, CONST DWORD \*IpArguments);

DWORD dwExceptionCode - код исключения (можно получить, используя функцию GetExceptionCode()).

DWORD dwExceptionFlags - флаг возобновления исключения (указывает, можно ли возобновить выполнение программы с команды, стоящей после функции: 0 - можно, 1 - нельзя).

DWORD nNumberOfArguments - количество аргументов для детализации описания исключения в массиве.

Пример. Программа нахождения частного двух положительных чисел.

#include <iostream>

#include <windows.h>

using namespace std;

int main()

{

double a, b;

double s;

\_\_try{

cout<<"Vvedite a, b"<<endl;

cin>>a>>b;

s=a/b;

if(a<0||b<0)

RaiseException(EXCEPTION\_PRIV\_INSTRUCTION, 0, 0, NULL);

cout<<"Res = "<<s<<endl;

}

\_\_except(EXCEPTION\_EXECUTE\_HANDLER)

{

DWORD er=GetExceptionCode();

if(er==EXCEPTION\_INT\_DIVIDE\_BY\_ZERO)

cerr<<"Division by zero"<<endl;

else

if(er==EXCEPTION\_PRIV\_INSTRUCTION)

cerr<<"Error:a<0||b<0"<<endl;

else cerr<<"Some other exeption"<<endl;

}

return 0;

}

Выход из блока \_\_try может быть осуществлён следующими способами:

1) Завершение блока без ошибки.

2) Выход из блока при помощи управляющей инструкции \_\_leave.

3) Выход из блока при помощи операторов return, break, continue, goto.

4) Передача управления обработчику исключения.

Любой выход из контролируемого блока приводит к выполнению оператора блока \_\_finally. Проверить состояние контролируемого блока можно с помощью функции bool AbnormalTermination(void), которая возвращает истину, если контрольный блок завершён с исключением.

Пример. Использование вложенных блоков.

#include <iostream>

#include <string>

#include <windows.h>

using namespace std;

int main(){

int x=0;

\_\_try{

cerr<<"Begin block 1"<<endl;

\_\_try{

cerr<<"Begin block 2"<<endl;

x/=0;//Ошибка

}

\_\_finally{

cerr<<"End block 2"<<endl;

if(AbnormalTermination())

cerr<<"Error in block 2"<<endl;

else cerr<<"Block 2 is completed without error"<<endl;

}

cerr<<"End block 1"<<endl;

}

\_\_except(puts("Filter"), EXCEPTION\_EXECUTE\_HANDLER)

{

cerr<<"Handing exeptions"<<endl;

}

cout<<"Completion of the program"<<endl;

return 0;

}

**Обработка исключений в объектах классов**

Пример. Класс с обработкой ошибки.

#include <iostream>

using namespace std;

class Cls

{

int x;

public:

Cls(int a):x(a){}

class MyErr{};

void fdel()

{

if (x==0) throw MyErr();

x=1/x;

}

};

int main()

{

try

{

Cls a(1);

a.fdel();

}

catch(Cls::MyErr)

{

cerr<<"Division by zero";

}

return 0;

}

Пример. Класс для ввода в массив целых положительных чисел.

#include <iostream>

#include <string>

using namespace std;

const int nmas=10;

class Cls

{

int mas[nmas];

int n;

public:

class MyErr

{

public:

int m;

string st;

MyErr(string a, int b):m(b), st(a){}

};

Cls(int k)

{

if(k>nmas)

throw MyErr("Array index out of bounds", k);

n=k;

for(int i=0; i<n; i++) mas[i]=i\*10;

}

void add\_mas()

{

cout<<"Enter the array:";

for(int i=0; i<n; i++)

{

cin>>mas[i];

if(mas[i]<0)

throw MyErr("Negative value", mas [i]);

}

}

};

int main()

{

try

{

Cls a(5);

a.add\_mas();

}

catch(Cls::MyErr er)//Создаётся объект

{

cerr<<"Error: "<<er.st<<"Value = "<<er.m;

}

return 0;

}

Пример. Класс для ввода-вывода положительного ненулевого целого числа.

#include <iostream>

using namespace std;

class BCls{

protected:

int x;

public:

BCls():x(0){}

virtual void MyErr(){

cerr<<"Error BCls"<<endl;

}

};

class Cls01:public BCls{

public:

void vvod(){

cout<<"Input x: ";

cin>>x;

if(!cin.good()) throw \*this;//good проверяет правильность ввода данных

}

virtual void MyErr(){

cout<<"Input error"<<endl;

}

};

class Cls02:public Cls01{

public:

void vivod(){

if(x<0) throw \*this;

cout<<"Output x="<<x<<endl;

}

virtual void MyErr(){

cout<<"Error: x<0"<<endl;

}

};

class Cls03:public Cls02{

public:

operator int() {return x!=0;}

virtual void MyErr(){

cout<<"Error:x==0"<<endl;

}

};

int main(){

try{

Cls03 a;

a.vvod();

a.vivod();

if(!a) throw a;

}

catch (BCls& err)

{

err.MyErr();

}

return 0;

}

Вывод:

Input: x=0

Output: x=0

Error: x=0