Семинар 7
Исключения

Исключения

В языке Си++ почти любое состояние, достигнутое во время выполнения программы, можно заранее определить как особую ситуацию и предусмотреть действия, которые нужно выполнить при её возникновении. Делается это при помощи механизма исключений.

Исключения

Общая схема посылки и обработки исключений:

```
try{
   операторы
   throw выражение1;
   операторы
   throw выражение 2;
   операторы
}
catch (спецификация исключения 1)
{операторы обработки исключения 1}
…
```

Исключения

Служебное слово try позволяет выделить в любом месте программы блок контроля за исключениями:

try {операторы}

Оператор throw генерирует (посылает) исключения. С помощью этого оператора генерируется специальный объект, называемый исключением. Все исключения создаются как временные объекты, а тип и значение каждого исключения определяются формирующим его выражением.

Исключения

Каждый обработчик имеет следующий формат:

catch (спецификация исключения)

{операторы обработки исключения}

Спецификация исключения в заголовке обработчика подобна спецификации параметра функции и может иметь одну из трёх форм:

тип_исключения имя

тип_исключения

многоточие (реакция на любые исключения)

Исключения

Внешне и функционально обработчик исключений похож на определение функции с одним параметром, не возвращающей никакого значения. Когда вслед за блоком try размещены несколько ловушек, они должны отличаться друг от друга типами принимаемых исключений.

Исключения

Пример. Применение механизма исключений при определении НОД.

Алгоритм Евклида для вычисления НОД:

1. Если x==y, то ответ найден, НОД=x
2. Если x<y, то y заменяется значением y-x
3. Если x>y, то x заменяется значением x-y

Алгоритм Евклида применим, если:

- Оба числа неотрицательны
- Оба числа отличны от 0

Исключения

Пример №1

```
#include "stdafx.h"
#include<iostream>
#include<string>
using namespace std;
int GCD(int x, int y){
	if(x==0) throw y;
	if(y==0) throw x;
	if(x<0) throw string("\nNegative x");
	if(y<0) throw string("\nNegative y");
	while(x!=y){
		if(x>y) x=x-y;
		else y=y-x;
	}
	return x;
}
```

Исключения

```
int _tmain(int argc, _TCHAR* argv[])
{
    try{
        cout<<"GCD(66,44)="<<GCD(66,44)<<endl;
        cout<<"GCD(0,7)="<<GCD(0,7)<<endl;
        cout<<"GCD(-6,4)="<<GCD(-6,4)<<endl;
    }
    catch(const string report){
        cout<<report<<endl;
    }
    catch(const int ex){
        cout<<"One parameter is ZERO! ";
        cout<<"Another equals "<<ex;
    }
    return 0;
}
```

Исключения

Пример №2

```
struct DATA{
    int n,m; string s;
    DATA(int x, int y, string str) :n(x), m(y), s(str) {}
};
int GCD(int x, int y){
    if(x==0 || y==0) throw DATA(x,y,"ZERO!");
    if(x<0) throw DATA(x,y,"Negative parameter 1");
    if(y<0) throw DATA(x,y,"Negative parameter 2");
    while(x!=y){
        if(x>y) x=x-y;
        else y=y-x;
    }
    return x;
}
```

Чтобы объект-исключение мог передавать больше информации, возможно использование собственных типов данных. Исключения в этой программе формируются как объекты специально созданного для решения этой задачи класса DATA

Исключения

```
int _tmain(int argc, _TCHAR* argv[])
{
    try{
        cout<<"GCD(66,44)="<<GCD(66,44)<<endl;
        cout<<"GCD(0,7)="<<GCD(0,7)<<endl;
        cout<<"GCD(-6,4)="<<GCD(-6,4)<<endl;
    }
    catch(DATA d){
        cout<<d.s<<" x="<<d.n<<" y="<<d.m<<endl;
    }
    return 0;
}
```

Обратите внимание, что объекты класса DATA формируются внутри одной функции, но доступны в другой функции. Это особое свойство исключений. Они создаются как временные статические объекты в одном блоке, но доступны в другом.

Исключения

Как объект исключение создаётся выражением из оператора throw в контролируемом блоке, пересылается за пределы этого блока, и, наконец, исчезает после окончания обработки.

После обработки исключения нет возможности вернуться в то место, откуда исключение было послано.

Исключения

Пример №3.
Передача типов в качестве объектов-исключений.

```
#include "stdafx.h"
#include<iostream>
#include<string>
using namespace std;

class ZeroDivide{};  //Именно эти типы передаются
class Overflow{};

int division(double x, double y){
    if(x==0.0 && y==0.0) throw 0.0;
    if(y==0) throw ZeroDivide(); //Вызов конструктора
    double res = x/y;
    if(res>1e+30) throw Overflow(); //Вызов конструктора
    return x;
}
```

Исключения

```
double x=1e-20, z=1e+20, w=0.0;
void RR(void){
	try{ w=division(4.0,w);
		z=division(z,x);
		w=division(0.0,0.0);
	}
	catch(Overflow){cout << "Overflow" << endl; z=1e+30; x=1.0;}
	catch(ZeroDivide){cout << "ZeroDivide" << endl; w=1.0;}
	catch(...){cout << "Indeterminacy" << endl;}
}
int main()
{
	RR();
	RR();
	RR();
	return 0;
}
```

Исключения

Если исключение послано, но соответствующий ему обработчик не найден, то вызывается специальная библиотечная функция terminate() (в Visual Studio также – abort()). Обращение к этой функции завершает выполнение программы.

Исключения

Оператор, формирующий исключение, может иметь две формы:

throw выражение; //в этом случае формируется объект-исключение

throw; //используется только внутри обработчика исключений, его применение разумно в том случае, когда существует вложение блока контроля за исключениями. Его задача – ретрансляция исключения в блок более высокого уровня.

Исключения

В заголовке функции можно указывать, какие исключения эта функция порождает или ретранслирует:

тип имя_функции(спецификация параметров)
throw (список типов)
{операторы тела функции}

Исключения

Особенности спецификации исключений для функций:

- если в заголовке функции нет суффикса throw, то эта функция может посылать любые исключения
- суффикс throw() с пустым списком имён типов указывает, что за пределы функции не должны передаваться никакие исключения

Исключения

Примеры заголовков:

```
// Функция порождает исключения двух
// типов – A и B.
void f1() throw(A, B) {}

// Функция не порождает исключений
void f2() throw() {}
```

Исключения

Если функция порождает исключения, отличные от тех, что указаны в её спецификации исключений, то управление передаётся специальной функции unexpected(), которая вызывает функцию terminate().

Исключения

```
void EvenOrOdd(int k) throw(int, const char *) {
        if(k%2 != 0) throw k;
        else throw "even";
}
void f(int j) throw() {
        try{
                try{EvenOrOdd(j);}
                catch(int){
                        cout<<"Odd"<<endl;
                        throw; //Ретрансляция объекта-исключения
                }
                catch(const char *){
                        cout<<"Even"<<endl;
                }
        }
        catch(int i){
                cout<<"Result="<<i<<endl;
        }
}
int main()
{
        f(4); f(7);
        return 0;
}
```

Исключения

Стандартом введена специальная форма обработки исключений непосредственно в конструкторах – специальных методах класса:

имя_класса(спецификация параметров)
try : список_инициализаторов
{операторы тела конструктора}
последовательность обработчиков исключений

Исключения

Задания

1. Создать функцию для вывода на экран элементов массива. Функция принимает в качестве параметров указатель на первый элемент массива и количество элементов. Функция генерирует исключения: 1) исключение типа NoElements, если количество элементов равно 0, 2) исключение типа int, если переданное количество элементов отрицательно. Во втором случае в обработчике исключения напечатайте переданное в функцию количество элементов массива.
2. Создайте рекурсивную функцию поиска n-го члена Фибоначчи, генерирующую исключения: 1) если задан отрицательный аргумент для функции, 2) если задан нулевой аргумент для функции, 3) если количество вызовов рекурсивной функции превысило некоторый предел (число выбрать самостоятельно).