Лабораторная работа № 5-6

Анализ и обеспечение обработки исключительных ситуаций

Цели: получение навыков анализа и обеспечения обработки исключительных ситуаций.

Теоретические вопросы

- Исключения в C++/C#
- Установленные исключения
- Спецификация исключения

Теоретическая часть

Обработка исключений в C#. Оператор try-catch

Иногда при выполнении программы возникают ошибки, которые трудно предусмотреть или предвидеть, а иногда и вовсе невозможно. Например, при передачи файла по сети может неожиданно оборваться сетевое подключение. Или когда нам необходимо было ввести число, и если вместо числа мы ввели бы строку, то при конвертации этой строки в численный тип программа бы аварийно завершила работу, и мы получили бы ошибку. Такие ситуации называются **исключениями**.

Обработка исключений – это описание реакции программы на подобные события (исключения) во время выполнения программы. Реакцией программы может быть корректное завершение работы программы, вывод информации об ошибке и запрос повторения действия (при вводе данных).

Примерами исключений может быть:

- деление на ноль;
- конвертация некорректных данных из одного типа в другой;
- попытка открыть файл, которого не существует;
- доступ к элементу вне рамок массива;
- исчерпывание памяти программы;
- другое.

Для обработки исключений в C# используется оператор **try-catch**. Он имеет следующую структуру:

```
try
{
  //блок кода, в котором возможно исключение
}
catch ([тип исключения] [имя])
{
  //блок кода – обработка исключения
}
```

Работает это все очень просто. Выполняется код в блоке try, и, если в нем происходит исключение типа, соответствующего типу, указанному в catch, то управление передается блоку catch. При этом весь оставшийся код от момента выбрасывания исключения до конца блока try не будет выполнен. После выполнения блока catch, оператор try-catch завершает работу.

Указывать имя исключения не обязательно. Исключение представляет собою объект, и к нему мы имеем доступ через это имя. С этого объекта мы можем получить, например, стандартное сообщение об ошибке (Message), или трассировку стека (StackTrace), которая поможет узнать место возникновения ошибки. В этом объекте хранится детальная информации об исключении.

Если тип выброшенного исключения не будет соответствовать типу, указанному в catch – исключение не обработается, и программа завершит работу аварийно.

Ниже приведен пример программы, в которой используется обработка исключения некорректного формата данных:

```
static void Main(string[] args)
{
   string result = "";
   Console.WriteLine("Введите число:");
   try
   {
     int a = Convert.ToInt32(Console.ReadLine()); //вводим данные, и конвертируем в
целое число
     result = "Вы ввели число " + a;
   }
   catch (FormatException)
   {
     result = "Ошибка. Вы ввели не число";
   }
   Console.WriteLine(result);
   Console.ReadLine();
}
```

Типы исключений

Ниже приведены некоторые из часто встречаемых типов исключений.

Exception – базовый тип всех исключений. Блок catch, в котором указан тип Exception будет «ловить» все исключения.
FormatException – некорректный формат операнда или аргумента (при передаче в метод).
NullReferenceException - В экземпляре объекта не задана ссылка на объект, объект не создан
IndexOutOfRangeException – индекс вне рамок коллекции
FileNotFoundException – файл не найден.
DivideByZeroException – деление на ноль

Несколько блоков catch

Одному блоку try может соответствовать несколько блоков catch:

```
try
{
   //блок1
}
catch (FormatException)
{
  //блок-обработка исключения 1
}
catch (FileNotFoundException)
{
  //блок-обработка исключения 2
}
```

В зависимости от того или другого типа исключения в блоке try, выполнение будет передано соответствующему блоку catch.

Блок finally

Оператор try-catch также может содержать блок finally. Особенность блока finally в том, что код внутри этого блока выполнится в любом случае, в независимости от того, было ли исключение или нет.

```
try
{
   //блок1
```

```
}
catch (Exception)
{
   //обработка исключения
}
finally
{
  //блок кода, который выполнится обязательно
}
```

Выполнение кода программы в блоке finally происходит в последнюю очередь. Сначала try затем finally или catch-finally (если было исключение).

Обычно, он используется для освобождения ресурсов. Классическим примером использования блока finally является закрытие файла.

Finally **гарантирует** выполнение кода, несмотря ни на что. Даже если в блоках try или catch будет происходить выход из метода с помощью оператора return – finally выполнится.

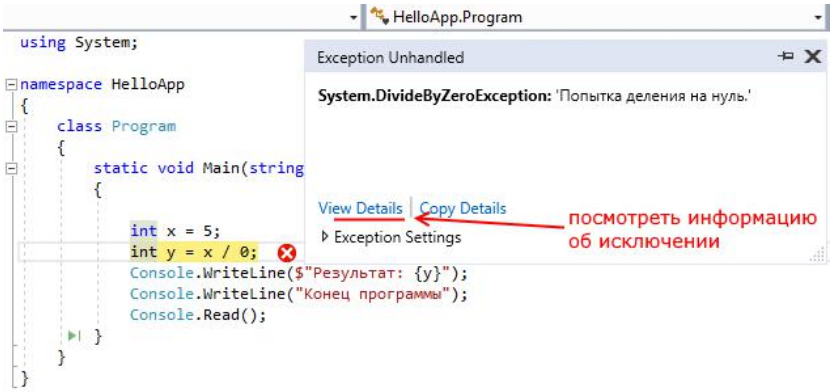
Операторы try-catch также могут быть вложенными. Внутри блока try либо catch может быть еще один try-catch.

Пример:

Рассмотрим следующий пример:

```
class Program
{
    static void Main(string[] args)
    {
        int x = 5;
        int y = x / 0;
        Console.WriteLine($"Результат: {y}");
        Console.WriteLine("Конец программы");
        Console.Read();
    }
}
```

В данном случае происходит деление числа на 0, что приведет к генерации исключения. И при запуске приложения в режиме отладки мы увидим в Visual Studio окошко, которое информирует об исключении:



В этом окошке мы видим, что возникло исключение, которое представляет тип System.DivideByZeroException, то есть попытка деления на ноль. С помощью пункта View Details можно посмотреть более детальную информацию об исключении.

И в этом случае единственное, что нам остается, это завершить выполнение программы.

Чтобы избежать подобного аварийного завершения программы, следует использовать для обработки исключений конструкцию try...catch...finally. Так, перепишем пример следующим образом:

```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            int x = 5;
            int y = x / 0;
            Console.WriteLine($"Результат: {y}");
        }
        catch
        {
            Console.WriteLine("Возникло исключение!");
        }
        finally
        {
            Console.WriteLine("Блок finally");
        }
        Console.WriteLine("Конец программы");
        Console.Read();
    }
}
```

В данном случае у нас опять же возникнет исключение в блоке try, так как мы пытаемся разделить на ноль. И дойдя до строки

```
int y = x / 0;
```

выполнение программы остановится. CLR найдет блок catch и передаст управление этому блоку.

После блока catch будет выполняться блок finally.

```
Возникло исключение!
Блок finally
Конец программы
```

Таким образом, программа по-прежнему не будет выполнять деление на ноль и соответственно не будет выводить результат этого деления, но теперь она не будет аварийно завершаться, а исключение будет обрабатываться в блоке catch.

Задание № 1. Написать программу, в которой обрабатываются следующие исключительные ситуации: "отрицательное значение возраста" и "год рождения больше текущего":

```cpp
#include <iostream>
#include <ctime>
using namespace std;

int AgeCalc(int year)
{
    if (year <= 0)
    {
        throw "ERROR: negative value of the birth year!!!";
    }


    struct tm *CDate;
    time_t tt = time(NULL);
    CDate = localtime(&tt);

    if (year > (1900 + CDate->tm_year) )
    {
        throw "ERROR: The birth year value is greater than current year value!!!";
    }

    return 1900 + CDate->tm_year - year;
}

int main()
{
    int BYear = 1980;
    int PAge = 0;

    try{
        PAge = AgeCalc(BYear);
    }
    catch (const char * s)
    {
        cout << s << endl<<endl;
    }
    catch(...)
    {
        cout << "Unknown exception" << endl<<endl;
    }


    cout << "For birth year " << BYear << " the age is " << PAge << endl;

    return 0;
}
```

Задание № 2. Есть массив целых чисел размером 10. С клавиатуры вводится два числа - порядковые номера элементов массива, которые необходимо суммировать. Например, если ввели 3 и 5 - суммируются 3-й и 5-й элементы. Нужно предусмотреть случаи, когда были введены не числа, и когда одно из чисел, или оба больше размера массива.

Задание № 3. Составить программу циклического вычисления значений функций, определенных из таблицы вариантов заданий. Значения R должны вводиться с клавиатуры. R1 и R2 –вещественные, R3 –комплексное. Предусмотреть вывод подсказок в виде (например):

```
Функция sin(x)
Q - Выход из программы
Введите число или Q:
```

Для вычисления значений функции написать функцию, вычисляющую требуемые по заданию значения. При разработке функции разрешается использовать функции модуля math.h.

Предусмотреть анализ всей введенной информации на ошибки, обработку ошибок реализовать с использованием обработчиков try… в зависимости от варианта задания.

Предусмотреть вывод имени функции, в которой произошла ошибка. Вывод на экран и чтение с клавиатуры организовать при помощи стандартных потоков ввода/вывода/ошибки. Вывести исходные данные и результат в виде (например):

```
Sin(R) = rez;
```

Где rez –результаты вычисления (вещественный).

Вариант	Функция	Обработчики
1	Sin(R1)*(pi)/R2-R3	Потеря разряда Деление на 0
2	Sin(R2)/pi*R1+R3	Потеря разряда Переполнение
3	Tan(R1)/R3+Cmod(R3)	Потеря разряда Прерывание
4	Arctan(R1)*R2+ R3	Потеря разряда Переполнение
5	Ln(R1-R2)*R2-R3	Обл.опр.арг. Исчезновение порядка

Задание № 4. Реализуйте класс «очередь» из строк. Реализуйте методы для вставки в очередь и удаления. Породите и обработайте ошибки динамического выделения памяти, переполнения очереди.

Задание № 5. Оформите отчет.