Обработка исключений

Литература

Основная:

Павловская Т.А., Щупако Ю.А. C/C++. Структурное программирование, Питер, 2005. – 402 с.

Дополнительная:

Гросс К. C# 2008: Пер. с. анг. – СПб. БХВ-Петербург, 2009. – 576 с.

Троелсен Э. Язык программирования C# 2005 и платформа /NET 2.0:пер. с анг. – М:. ООО «И.Д. Вильямс, 2007. – 1168 с.

Вопросы:

1. Ошибки и исключения.

2. Стандартная обработка исключений.

3. Программные средства обработки исключений.

1. Ошибки и исключения

Ошибки в программе:

- ***синтаксические ошибки;***
- ***семантические ошибки;***
- ***некорректность чужого кода;***
- ***ошибки пользователя;***
- ***нарушения технологии обработки;***
- ***некорректная работа оборудования.***

Обнаруживаются в ходе выполнения программы

Ошибки в программе

Для предотвращения или выявления ошибочных ситуаций *в ходе выполнения программы* применяются:

- **средства разработчика;**
- **средства ОС. Например, Windows API содержит сотни кодов ошибок.**

Указанные средства не обеспечивают единства в представлении сообщений об ошибках.

Последствия проявления ошибок

- невозможность решения задачи;
- утечка ресурсов;
- наличие «мусора» после выполнения программы;
- искажения хранимых данных, программ и т.д.

Последствия проявления ошибок

Часть ошибок система обходит, например, для вещественных чисел:

- при вычислении квадратного корня из отрицательной величины результатом является величина NaN;

- при делении на 0 выдается результат бесконечность;

- при вычислении логарифма нуля выдается результат отрицательная бесконечность.

Последствия проявления ошибок

При стандартных настройках проекта и работе с целыми числами слишком большое целочисленное значение урезается путем отбрасывания «лишних» старших цифр.

Но при этом понять где произошла ошибка не удается.

Обработка исключений

нестандартные ситуации называют исключениями.

Обработка исключений

- средства языка программирования, предназначенные для описания реакции программы на нестандартные ситуации (ошибки…), возникающие при исполнении программы и приводящие к невозможности или нецелесообразности дальнейшей работы в соответствии с основным алгоритмом программы.

Исключительные ситуации

Исключительные ситуации, возникающие при работе программы, можно разделить на две группы:

- синхронные и
- асинхронные

Исключительные ситуации

Синхронные исключения могут возникнуть только в определённых, заранее известных точках программы.

Ошибка деления на нуль, ошибка чтения файла — типичные синхронные исключения, так как возникают они только при выполнении соответствующих операций (соответственно, деления, чтения из файла и т.п.).

Исключительные ситуации

Асинхронные исключения могут возникать в любой момент времени и не зависят от того, какая конкретно инструкция программы выполняется

Типичные примеры таких исключений: аварийный отказ питания или поступление новых данных.

Обработка исключений

простейший формат защищенного блока, который имеет вид

- **try {*операторызащищенногоблока*}**
- **catch(…) {*обработчикошибочнойситуации*}**

Обработка исключений

пример перехвата ошибочной ситуации.

```
float k;
int i, j;
…
try {
    i=j/k;
    }
catch(…) {
 cout << “Ошибка (деление на ноль)” << endl;}
```

Обработка исключений

Для возбуждения **собственных исключений** используется оператор
throw *[выражение]*

Тип выражения, указанного в операторе throw, определяет тип исключительной ситуации, а значение может быть передано в блок обработки исключительных ситуаций.

Этот механизм, заявленный как стандартный, представляется весьма экзотическим без использования механизма классов. Лишь использование стандартных классов-исключений или разработка собственных классов позволяют в полной мере оценить все возможности такого подхода.

Полный формат защищенного блока

try {*операторызащищенногоблока*}
{catch-блоки}…

Catch-блок имеет один из следующих форматов:
catch (*тип*)
{обработчикошибочнойситуации}
catch (*тип идентификатор*)
{обработчикошибочнойситуации}
catch (…) {*обработчикошибочнойситуации*}

Полный формат защищенного блока

Первый формат используется, если нам надо указать тип перехватываемого исключения, но не нужно обрабатывать связанное с этим исключением значение (это достигается при использовании второго формата оператора catch).

Наконец, третий формат оператора catch позволяет обработать все исключения (в том числе и ошибки выполнения, что было описано в предыдущем пункте).

Обработка исключений

1. Создается статическая переменная со значением, заданным в операторе throw. Она будет существовать до тех пор, пока исключение не будет обработано. Если переменная-исключение является объектом класса, при ее создании работает конструктор копирования.
2. Завершается выполнение защищенного try-блока: раскручивается стек подпрограмм, вызываются деструкторы для тех объектов, время жизни которых истекает и т.д.
3. Выполняется поиск первого из catch-блоков, который пригоден для обработки созданного исключения. Поиск ведется по следующим критериям:
если тип, указанный в catch-блоке, совпадает с типом созданного исключения, или является ссылкой на этот тип;
класс, заданный в catch-блоке, является предком класса, заданного в throw, и наследование выполнялось с ключом доступа public;
указатель, заданный в операторе throw, может быть преобразован по стандартным правилам к указателю, заданному в catch-блоке.
в операторе throw задано многоточие.

Обработка исключений

Если нужный обработчик найден, то ему передается управление и, при необходимости, значение оператора throw. Оставшиеся catch-блоки, относящиеся к защищенному блоку, в котором было создано исключение, игнорируются.
Из указанных правил поиска следует, что очень важен порядок расположения catch-блоков. Так, блок catch(…) должен стоять последним в списке, а блок catch (void *) – после всех блоков с указательными типами.

Если ни один из catch-блоков, указанных после защищенного блока, не сработал, по таким же правилам проводится выход из внешнего блока try (если он, конечно, есть!).

В конце оператора catch может стоять оператор throw без параметров. В этом случае работа catch-блока считается незавершенной, и происходит поиск соответствующего обработчика на более высоких уровнях.

Обработка исключений

```
try {                                          //Пример:
    …
    throw "Error!";
    …
    } // try
 catch (int) {…
    }

catch (float) {…
    }

catch (char * c) {…
    }

catch (...) { …
    }
```

- http://cppstudio.com/post/9773/

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

int main()
{
  setlocale(LC_ALL, "rus");

  int num1;
  int num2;
  int var = 2; //управляющая переменная для while

  while(var--) //пока var - истина (не равно 0)
  {
    cout << "Введите значение num1: ";
    cin >> num1;
    cout << "Введите значение num2: ";
    cin >> num2;

    cout << "num1 + num2 = " << num1 + num2 << endl;
    cout << "num1 / num2 = " << num1 / num2 << endl;
    cout << "num1 - num2 = " << num1 - num2 << endl;
    cout << "==================================" << endl << endl;
  }

  cout << "Программа завершила работу!" << endl << endl;

return 0;
}
```

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

int main()
{    setlocale(LC_ALL, "rus");
     int num1;
    int num2;
    int var = 2;

    while(var--)
    {
        cout << "Введите значение num1: ";
        cin >> num1;
        cout << "Введите значение num2: ";
        cin >> num2;

        cout << "num1 + num2 = " << num1 + num2 << endl;
        cout << "num1 / num2 = ";

       try //код, который может привести к ошибке,располагается тут
          {
               if (num2 == 0)
               {throw 123; //генерировать целое число 123
               }
cout << num1 / num2 << endl;
        }
       catch(int i)//сюда передастся число 123
          {
               cout << "Ошибка №" << i << " - на 0 делить нельзя!!!!" << endl;
          }

        cout << "num1 - num2 = " << num1 - num2 << endl;
        cout << "================================" << endl << endl;
    }

    cout << "Программа завершила работу!" << endl << endl;;

return 0;
}return 0;
}
```