Тема 4.6 Реализация рассмотренных консольных приложений как оконных

Обработка исключений.

С ошибками работают в тех местах программы, где они могут появляться. Преимущество этого подхода в том, что программист, читающий текст программы, может видеть обработку ошибки в непосредственной близости от соответствующих операторов и определить, была ли осуществлена соответствующая проверка ошибки.

Проблема такого подхода состоит в том. Что основной код в этом случае «загрязняется» обработкой ошибок. Для программиста, работающего с прикладной программой, становится труднее читать этот код и определять, правильно ли он функционирует. Это делает код более трудным для восприятия и поддержки.

Вот некоторые распространенные примеры исключений: нехватка памяти, выход индекса за пределы массива, арифметическое переполнение, деление на нуль, недопустимые параметры функций.

Обработка исключений осуществляется для того, чтобы дать возможность программе перехватить и обработать ошибку прежде, чем ошибка произойдет и наступят ее неприятные последствия. Если программист не обеспечивает средства обработки неисправимой ошибки, то при ее возникновении программа прекращает свою работу.

Обработка исключений используется в случаях, при которых система может быть восстановлена для нормальной работы после ошибки, вызвавшей исключение. Процедура восстановления называется обработчиком исключения.

Исключения – возникновение непредвиденных ошибочных условий в момент работы программы. В то же время исключение – это более общее чем ошибка понятие, так как может возникать и тогда, когда в программе нет ошибок (например, сбой выделения памяти при создании объекта класса). В общем, исключение обозначает особую ситуацию, когда требуется безвозвратно переключить выполнение программы на блок обработки такой ситуации. Выявление особой ситуации производится только программным путем при помощи проверки нормального хода выполнения программы.

Средства обработки ошибочных ситуаций позволяют передать обработку исключений из кода, в котором возникло исключение, некоторому другому программному блоку, который выполнит в данном случае некоторые определенные действия. Таким образом, основная идея данного механизма состоит в том, что функция проекта, которая обнаружила непредвиденную ошибочную ситуацию, которую она не знает, как решить, генерирует сообщение об этом (бросок исключения). А система вызывает по этому сообщению программный модуль, который перехватит исключение и отреагирует на возникшее нештатное событие. Такой программный модуль называют «обработчик» или перехватчик исключительных ситуаций. И в случае возникновения исключения в его обработчик передается произвольное количество информации с контролем ее типа. Эта информация и является характеристикой возникшей внештатной ситуации.

Обработка исключения в C++ - это обработка с завершением. Это означает, что исключается невозможность возобновления выполнения программы в точке возникновения исключения.

Для обеспечения работы такого механизма были введены следующие ключевые слова:

try - проба испытания;

catch – перехватить (обработать);

throw – бросать.

Кратко рассмотрим их назначение.

Ключевое слово ***try*** открывает блок кода, в котором может произойти ошибка; это обычный составной оператор:

```
try {
        код
};
```

Код содержит набор операций и операторов, который и будет контролироваться на возникновение ошибки. В него могут входить вызовы функции пользователя, которые компилятор также возьмет на контроль.

Исключения, которые генерируются в блоке try, обычно перехватываются обработчиком, описанным в блоке catch, следующим непосредственно за этим блоком try.

```
try {
        код
}
catch () {
        код
}
```

За блоком try может не следовать ни одного блока catch, или может следовать несколько таких блоков. Если при выполнении блока try не генерируется ни одно исключение, все обработчики исключений пропускаются и управление передается первому оператору после последнего обработчика.

Среди данного набора операторов и операций обязательно указывают операцию броска исключения ***throw,*** которая имеет следующий формат:

```
throw выражение;
```

где *выражение* определяет тип информации, которая описывает исключение (например конкретные типы данных).

Это называется генерацией исключения или возбуждением исключения. После генерации исключение будет перехвачено ближайшим обработчиком исключений (ближайшим к блоку try, в котором было сгенерировано исключение), содержащим спецификацию соответствующего типа. Обработчики событий для блока try перечисляются сразу после него.

В процессе генерации исключения создается и инициализируется временная копия операнда throw. Этот временный объект затем инициализирует параметр в обработчике исключения. Временный объект уничтожается, когда завершается выполнение обработчика исключения и управление передается программе.

Исключение генерируется в функции блока try или в функции, вызываемой из из блока try. Точка, в которой выполняется генерация (throw), называется точкой генерации. После того, как, исключение сгенерировано, управление не может возвратиться в точку генерации.

Когда исключение сгенерировано, программное управление покидает текущий блок try и передается соответствующему обработчику catch (если он существует), расположенному после этого блока try.

Обработчик исключения ***catch*** перехватывает информацию:

```
catch (тип и параметр) {
        код
}
```

Через параметр обработчику передаются данные определенного типа, описывающие обрабатываемое исключение. Код определяет те действия, которые надо выполнить при возникновении данной конкретной ситуации. В C++ используют несколько форм

обработчиков. Рассмотренный обработчик получил название параметризованный специальный перехватчик.

Перехватчик должен следовать сразу же после блока контроля, т.е. между обработчиком и блоком контроля не должно быть ни одного оператора. При этом в одном блоке контроля можно вызывать исключения разных типов для различных ситуаций, поэтому обработчиков может быть несколько. В этом случае их необходимо расположить сразу же за контролирующим блоком последовательно друг за другом.

Кроме того, запрещены переходы как из вне в обработчик, так и между обработчиками.

Можно воспользоваться универсальным или абсолютным обработчиком:

```
catch (...) {
                код
            }
```

где (…) – означают способность данного перехватчика обрабатывать информацию любого типа. Такой обработчик располагают последним в пакете специализированных обработчиков. Тогда, если исключение не будет перехвачено специализированными обработчиками, то будет выполнен последний – универсальный.

Если не возникнет исключение, набор обработчиков будет обойден, т.е. проигнорирован.

Если же исключение было «брошено» при возникновении критической ситуации, то будет вызван конкретной перехватчик при совпадении его параметра с выражением в операторе броска, т.е. управление будет передано найденному обработчику. После выполнения кода вызванного обработчика управление передается оператору, который расположен за последним перехватчиком, или проект корректно завершает работу.

Существенное отличие вызова конкретного обработчика от вызова обычной функции заключается в следующем: при возникновении исключения и передаче управления определенному обработчику система осуществляет вызов всех деструкторов для всех объектов классов, которые были созданы с момента начала контроля и до возникновения исключительной ситуации, с целью их уничтожения.

Блоки try как составные блоки могут быть вложены друг в друга. В случае возникновения исключения в некотором текущем блоке, поиск обработчика последовательно продолжается в блоках, предшествующих уровней вложенности с продолжением вызова деструктора.

```
#include <iostream.h>
int main()
{int donuts, milk;
    double dpg;
   try {
         cout << “Vvedite kolichestvo ponchikov:\n”;
          cin >> donuts;
         cout << “Vvedite kolichestvo ctakanov moloka”;
          cin >> milk;
        if (milk <= 0)
                throw donuts;
        dpg = donuts/double(milk);
        cout << donuts << “ ponchikov.\n”
              << milk << “ ctakanov moloka.\n”
              << “Na ctakan moloka y vas prixoditsya ”
              << dpg << “ ponchikov\n”;
```

```
}
catch (int e)
 {
     cout << e << " ponchikov, no net moloka!\n"
          << "Cxodite-ka za nim v magazin.\n";
   }
return 0;}
```

throw donuts; - передает на обработку значение типа int.

Такое переданное на обработку значение, в данном случае donuts, называется исключением, а сам процесс – генерацией исключения. Можно сгенерировать выражение любого типа. При генерации исключения код try-блока прекращает выполнение «поймавшего» исключения catch-блока. Такое выполнение кода из catch-блока называется перехватом или обработкой исключения. При генерации исключения оно обязано быть обработанным (перехваченным) некоторым catch-блоком. Этот catch-блок очень похож на определение функции с параметром типа int. На самом деле это не определение функции, но некоторыми чертами блок очень похож на него. Он представляет собой фрагмент кода, который выполняется, когда программа в предшествующем try-блоке встречает и выполняет инструкцию throw некоторое_значение_типа int;

Таким образом, эта инструкция throw подобна вызову функции, только вместо вызова функции выполняется код в catch-блоке, который часто называется обработчиком исключения.

Идентификатор e в строке catch (int e) похож на параметр при вызове функции, и действует аналогично параметру (так что и называть его будем параметром catch-блока).

Параметр catch-блока выполняет две функции:

1. Параметру блока предшествует имя типа, которое определяет, какого именно типа исключение обрабатывается данным блоком.
2. Параметр блока определяет имя перехваченного значения; таким образом, код блока может работать со значением сгенерированного исключения.

Пример обработки исключений: деление на нуль

Программа использует операторы try, throw, catch, чтобы обнаружить, указать и обработать исключение деления на нуль.

```
#include <iostream.h>

// Определение класса DivideByZeroError, используемого при обработке исключения, генерируемого делением на нуль.

class DivideByZeroError {
public:
  DivideByZeroError () : message ("Деление на нуль") { }
  void printMessage () const {cout << message;}
private:
  const char *message;
};
// Описание функции quotient. Используется, чтобы показать генерацию исключения  при ошибке деления на нуль.

float quotient (int num1, int num2)
{
    if (num2 == 0)
        throw DivideByZeroError ();
        return (float) num1 / num2;
```

```
// Управляющая программа

main ()
{
    cout << "Введите два целых числа для расчета их частного: ";
    int number1, number2;
    cin >> number1 >> number2;
    try {                    // включает код, который может сгенерировать исключение
        float result = quotient (number1, number2);
}
    catch (DivideByZeroError error) {    // обработчик ошибки
        cout << "ОШИБКА: ";
        error.printmessage ();
        cout << endl;
        return 1;                    // завершение при ошибке
}
return 0;                    // нормальное завершение
}
```

Результат выполнения программы:

Введите два целых числа для расчета их частного: 7 3
Частное равно 2.333333

Введите два целых числа для расчета их частного: 23 0
ОШИБКА: Деление на нуль

Программа выполняет блок try, включающий код, который может возбудить исключение. Вызывается функция quotient, содержащая операцию деления. Функция quotient создает объект исключение деления на нуль. В общем случае ошибка может появляться при выполнении операторов, явно упомянутых в блоке try, или в вызовах функций или даже в глубоко вложенных вызовах функций, инициализированных оператором из блока try.

После блока try записан блок catch, содержащий обработчик исключения для ошибки деления на нуль. Вообще, когда исключение генерируется внутри блока try, оно перехватывается блоком catch, в котором определен тип, соответствующий сгенерированному исключению. В программе определено, что блок catch ,будет перехватывать объекты исключения типа DivideByZeroError; этот тип соответствует типу объекта исключения, генерируемого в функции quotient. Тело данного обработчика исключения просто выдает сообщение об ошибке и возвращает 1, что указывает на прерывание выполнения из-за ошибки.

Если, в процессе выполнения код в блоке try не генерирует исключение, тогда все обработчики catch, следующие за блоком try, пропускаются и управление передается первой строке после блоков catch; в программе выполняется оператор возврата return, который возвращает 0, указывая на нормальное завершение работы.

В функции quotient, когда условный оператор if определяет, что знаменатель равен нулю, выполняется оператор throw, который специфицирует имя конструктора объекта исключения. В результате создается объект класса DivideByZeroError. Этот объект будет перехвачен оператором catch (в котором специфицирован тип DivideByZeroError), расположенным после блока try. Конструктор класса DivideByZeroError просто копирует строку «Деление на нуль» в закрытый элемент данных message. Сгенерированный объект принимается как параметр в обработчике catch (в данном случае, как параметр error) и сообщение печатается путем обращения к открытой функции printmessage.

Избегайте имени Exception для любого класса исключения. Весьма вероятно, что это имя используется библиотеками.

Множественная обработка исключений.

```
#include <iostrem.h>
#include <string.h>
class NegativNumber
{
   public:
NegativNumber(){}
     NegativNumber(string take_me_to_your_catch_blok):
message(take_me_to_your_catch_blok) {}
      string get_message() { return message; }
     private:
    string message;
};
class DivideByZero
{};
int main()
{
    int whiteRose, redRose;
     double portion;
      try{
        cout << "Vvedite kolichestvo boicov Beloi Rozi:\n";
        cin >> whiteRose;
            if (whiteRose <0)
          throw NegativNumber("Beloi Rozi");
        cout << "Vvedite kolichestvo boicov Krasnoi Rozi:\n";
        cin >> redRose;
         if (redRose<0)
          throw NegativNumber("Krasnoi Rozi");
          if (redRose!=0)
              portion = whiteRose/ double(redRose);
          else
                        throw   DivideByZero();
         cout << "Na odnogo boica Krasnoi Rozi prixoditsya "
               << portion << " boicov Beloi Rozi.\n";
      }
      catch (NegativNumber e)
       {
            cout << "y " << e.get_message()
                  << " ne moget bit otricatelnogo "
                  << "chisla boicov!\n";
}
          catch(DivideByZero)
         {
            cout << "Eto konec ...\n";
          }
      return 0;
}
```

При перехвате исключений разного типа порядок catch-блоков может оказаться весьма важен. Когда в try-блоке генерируется исключение, для поиска обрабатывающего его catch-блока все блоки проверяются в том порядке, в котором они расположены в тексте программы, и первый же подходящий блок выполняет обработку.

Рассмотрим определение класса DivideByZero. Этот класс не имеет ни переменных-членов, ни функций-членов. Он не имеет ничего, кроме имени; но для нас этого вполне достаточно. Генерация исключения DivideByZero активизирует необходимый catch-блок. При использовании тривиального класса исключения обычно нас интересует только тип; значение не представляет никакого интереса, и потому в блоке catch опускается переменная параметр. Параметр может быть опущен и в любых других случаях, когда значение исключения нас не интересует.

Throw-список в производных классах.

При переопределении функции в производном классе она должна иметь тот же throw-список, что и в базовом классе, т.е. вы не можете ничего добавлять в throw-список, но можете удалять из него имеющиеся там типы исключений.