Обработка исключений

C++ обеспечивает встроенный механизм обработки ошибок, называемый обработкой исключительных ситуаций (exception handling), который позволяет правильно реагировать па ошибки, возникающие в ходе выполнения программы. Используя механизм исключительных ситуаций, программа может автоматически вызывать процедуру обработки ошибок. Исключительная ситуация, или исключение – это возникновение непредвиденного или аварийного события, например, это деление на ноль или обращение по несуществующему адресу памяти. Обычно эти события приводят к завершению программы с системным сообщением об ошибке.

В C++ можно устранить ошибку и продолжать выполнение программы.

Исключения C++ не поддерживают обработку асинхронных событий, таких, как ошибки оборудования или обработку прерываний, например, нажатие клавиш Ctrl+C. Механизм исключений предназначен только для событий, которые происходят в результате работы самой программы и указываются явным образом.

В C++ исключение – это объект, при возникновении исключительной ситуации программа генерирует объект-исключение. Это удобно, так как с объектами, в отличие от ситуаций, можно, например, объект-исключение объявить как обычную переменную, передать его как параметр любым из возможных способов или возвратить в качестве результата. Можно объявлять массивы исключений или включать объекты-исключения в качестве полей в другие классы. В дальнейшем будет использоваться термин "исключение", понимая под этим объект-исключение.

Общая схема обработки исключений такова: в одной части программы, где обнаружена аварийная ситуация, исключение порождается; другая часть программы контролирует возникновение исключения, перехватывает и обрабатывает его.

В C++ есть три зарезервированных слова: try (контролировать), catch (перехватывать), throw (порождать), – которые и используются для организации процесса обработки исключений.

Обработка исключений

Исключения позволяют логически разделить вычислительный процесс на две части – обнаружение аварийной ситуации и ее обработка.

Это важно не только для лучшей структуризации программы. Главной причиной является то, что функция, обнаружившая ошибку, может не знать, что предпринимать для ее исправления, а использующий эту функцию код может знать, что делать, но не уметь определить место возникновения, что особенно актуально при использовании библиотечных функций и программ, состоящих из многих модулей.

Другое достоинство исключений состоит в том, что для передачи информации об ошибке в вызывающую функцию не требуется применять возвращаемое значение, параметры или глобальные переменные, поэтому интерфейс функций не раздувается.

Это особенно важно, например, для конструкторов, которые по синтаксису не могут возвращать значение.

Обработка исключений

Общий механизм обработки исключений

Место, в котором может произойти ошибка, должно входить в контролируемый блок – составной оператор, перед которым записано ключевое слово try.

Процесс обработки исключительных situаций:

- **Обработка исключения начинается с появления ошибки. Функция, в которой она возникла, генерирует исключение. Для этого используется ключевое слово throw с параметром, определяющим вид исключения. Параметр может быть константой, переменной или объектом и используется для передачи информации об исключении его обработчику.**
- **Отыскивается соответствующий обработчик исключения и ему передается управление.**
- **Если обработчик исключения не найден, вызывается стандартная функция terminate, которая вызывает функцию abort, аварийно завершающую текущий процесс. Можно установить собственную функцию завершения процесса.**

Обработка исключений

Синтаксис исключений

Ключевое слово try служит для обозначения контролируемого блока – кода, в котором может генерироваться исключение. Блок заключается в фигурные скобки:

try { <контролируемый_блок_программы> }

Все функции, прямо или косвенно вызываемые из try-блока, также считаются ему принадлежащими.

Размер блока try может варьироваться. Он может содержать как несколько операторов, так и целую программу (в этом случае функция main() целиком помещается в блок try).

Генерация (порождение) исключения происходит по ключевому слову throw, которое употребляется либо с параметром, либо без него:

throw [<выражение>];

Тип выражения, стоящего после throw, определяет тип порождаемого исключения. При генерации исключения выполнение текущего блока прекращается, и происходит поиск соответствующего обработчика и передача ему управления. Как правило, исключение генерируется не непосредственно в try-блоке, а в функциях, прямо или косвенно в него вложенных.

Выражение генерации исключения на практике означает либо константу, либо переменную некоторого типа. Тип объекта-исключения может быть любым, как встроенным, так и определяемым программистом.

Например:

```
throw 7;  // здесь объект-исключение – это целая константа, которая может быть условным номером-кодом ошибки, в общем случае этот код ошибки может вычисляться, например: throw 3*v-i;
throw "Ошибка: деление на нуль!";  // это символьная константа – сообщено об ошибке
throw Message[i];   // здесь объект-исключение – строка – сообщение  об ошибке.
```

Можно также определить собственный тип объекта-исключения, объявив новый класс, например: `class NegativeArgument{}; NegativeArgument exeption; if (x>0) double t = x/sqrt(x); else throw exeption;`

Обработка исключений

Синтаксис исключений

Обработчики исключений начинаются с ключевого слова catch, за которым в скобках следует тип обрабатываемого исключения.

Они должны располагаться непосредственно за try-блоком. Можно записать один или несколько обработчиков в соответствии с типами обрабатываемых исключений. Синтаксис обработчиков напоминает определение функции с одним параметром – типом исключения.

Существует три формы записи:

```
catch(<тип> <имя>)  { < тело обработчика > }
catch(<тип>)  { < тело обработчика > }
catch(...)  { < тело обработчика > }
```

Первая форма применяется, когда имя параметра используется в теле обработчика для выполнения каких-либо действий – например, вывода информации об исключении. Вторая форма не предполагает использования информации об исключении, играет роль только его тип. В третьей форме многоточие вместо параметра обозначает, что обработчик перехватывает все исключения. Так как обработчики просматриваются в том порядке, в котором они записаны, обработчик третьего типа следует помещать после всех остальных.

Пример:

```
catch (int i)  { < Обработка исключений типа int > }
catch (const char *)  { < Обработка исключений типа const char* > }
catch (...)  { < Обработка всех необслуженных исключений > }
```

После обработки исключения управление передается первому оператору, находящемуся непосредственно за обработчиками исключений. Туда же, минуя код всех обработчиков, передается управление, если исключение в try-блоке не было сгенерировано.

Обработка исключений

Простой пример обработки исключительной ситуации

```
#include <iostream>
using namespace std;
int main ()
{
  cout << "Начало\n";
  try // Начало блока try
   {
    cout << "Внутри блока try\n";
    throw 100;  // Генерируем ошибку
    cout<<"Этот оператор не
          выполняется.";
   }
 catch (int i) // Перехват ошибки
   {
      cout<<перехват  исключительной
      ситуации - значение равно: ";
    cout << i << "\n";
   }
 cout << "Конец";
return 0;
}
```

Блок try содержит три оператора. С ним связан оператор catch (int i), выполняющий обработку целочисленной исключительной ситуации. Внутри блока try выполняются только два из трех операторов: первый оператор cout и оператор throw. При генерации исключительной ситуации управление передается оператору catch, а выполнение блока try прекращается. Иначе говоря, блок catch не вызывается. Просто программа переходит к его выполнению. (Для этого стек программы автоматически обновляется.) Таким образом, оператор cout, следующий за оператором throw, никогда не выполняется.

Обычно оператор catch пытается исправить ошибку, предпринимая соответствующие действия. Если это возможно, выполнение программы возобновляется с оператора, следующего за блоком catch. Однако часто ошибку исправить невозможно, и блок catch прекращает выполнение программы, вызывая функцию exit () или abort ().

Обработка исключений

Исключение может генерироваться вне блока try только в том случае, если оно генерируется функцией, которая вызывается внутри этого блока.

Пример:

```
#include <iostream>
using namespace std;
void Xtest (int  test)
 {
  cout<<"Внутри функции Xtest, test =:  "
                    <<test<<"\n";
  if (test)   throw test;
}

int main()
{
  cout << "Начало\n";
   try      // Начало блока try
    { cout << "Внутри блока try\n";
       Xtest(0); Xtest(1); Xtest(2);
    }
   catch (int i)   { // Перехват ошибки
      cout  <<  "Перехват  исключительной
ситуации – значение равно: "; cout << i
<<"\n";
}
```

Блок try может находиться внутри функции. В этом случае при каждом входе в функцию обработка исключительной ситуации выполняется заново.

Пример:

```
#include <iostream>
using  namespace std;
// Блоки try/catch находятся внутри функции.
void Xhandler (int  test)
{
   try
     {  if (test)  throw test; }
   catch (int i) {
     cout  <<  "Перехват  исключительной
ситуации #:  " << i << '\n';
     }
}

int main()
  {
    cout << "Начало\n";
    Xhandler(1); Xhandler(2);
    Xhandler(0); Xhandler(3);
   cout << "Конец";
   return 0;
}
```

Обработка исключений

Важно четко понимать, что код, связанный с оператором catch, выполняется только при перехвате исключительной ситуации.

В противном случае оператор catch просто игнорируется. (Иначе говоря, поток управления никогда не проходит через тело оператора catch.)

Например, в следующей программе исключительные ситуации вообще не генерируются, и оператор catch не выполняется

```
#include <iostream>
using namespace std;
 int main()
 {
    cout<<"Начало\n";
    try // Начало блока try
     {
       cout<<"Внутри блока try\n";
       cout<< "Все еще внутри блока
                              try\n";
     }
    catch  (int i)    // Перехват ошибки
       {
         cout<<"Перехват исключитель-
        ной ситуации - значение равно: ";
          cout << i << "\n";
       }
      cout << "Конец";
  return 0;
}
```

Эта программа выводит на экран следующие сообщения:

Начало
Внутри блока try
Все еще внутри блока try
Конец

!! Как видим, поток управления обошел оператор catch стороной!

Обработка исключений

Перехват исключений

Когда с помощью throw генерируется исключение, функции исполнительной библиотеки C++ выполняют следующие действия:

1) создают копию параметра throw в виде статического объекта, который существует до тех пор, пока исключение не будет обработано;
2) в поисках подходящего обработчика раскручивают стек, вызывая деструкторы локальных объектов, выходящих из области действия;
3) передают объект и управление обработчику, имеющему параметр, совместимый по типу с этим объектом.

При раскручивании стека все обработчики на каждом уровне просматриваются последовательно, от внутреннего блока к внешнему, пока не будет найден подходящий обработчик.

Обработчик считается найденным, если тип объекта, указанного после throw:

- тот же, что и указанный в параметре catch (параметр может быть записан в форме T, const T, T& или const T&. где T – тип исключения);
- является производным от указанного в параметре catch (если наследование производилось с ключом доступа public);
- является указателем, который может быть преобразован по стандартным правилам преобразования указателей к типу указателя в параметре catch.

Отсюда следует, что обработчики производных классов следует размещать до обработчиков базовых, поскольку в противном случае им никогда не будет передано управление. Обработчик указателя типа void автоматически скрывает указатель любого другого типа, поэтому его также следует размещать после обработчиков указателей конкретного типа.

Обработка исключений

Перехват исключений. Пример:

```
#include <fstream. h>
  class Hello
   {
// Класс, информирующий о своем создании и уничтожении
    public:
      HelloO {  cout << "Hello!" << endl;  }
      Hello() { cout<<"Bye!"<<endl;  }
   };
void f1()
{
ifstream ifs("\\INVALID\\FILE\\NAME");
    // Открываем файл
 if (!ifs)
  {
  cout<<"Генерируем исключение"<<endl;
 throw "Ошибка при открытии файла";
  }
}
void f2()
{
Hello H:  // Создаем локальный объект
f1();      // Вызываем функцию, генерирующую исключение
}
```

//см. *продолжение*

```
int main()                    // продолжение
try
{ cout<<"Входим в try-блок"<<endl;
f 2();
cout<<"Выходим из try-блока"<endl;  }
catch(int i)
 { cout<<"Вызван обработчик int,
              исключение - "<<i<<endl;
  return -1; }
catch (const char * p)
  {  cout<<"Вызван обработчик const char*, исключение – "<<p<<endl;
 return -1; }
catch(...)
  {  cout<<"Вызван обработчик всех исключений" << endl;
 return -1;  }
return 0;  // Все обошлось благополучно
}
```

Результаты выполнения программы:

Входим в try-блок Hello!
Генерируем исключение
Bye!
Вызван обработчик const char *, исключение - Ошибка при открытии файла

Обработка исключений

Перехват исключений

Анализ примера:

После порождения исключения был вызван деструктор локального объекта, хотя управление из функции f1 было передано обработчику, находящемуся в функции main. Сообщение "Выходим из try-блока" не было выведено. Для работы с файлом в программе использовались потоки.

Таким образом, механизм исключений позволяет корректно уничтожать объекты при возникновении ошибочных ситуаций. Поэтому выделение и освобождение ресурсов полезно оформлять в виде классов, конструктор которых выделяет ресурс, а деструктор освобождает.

В качестве примера можно привести класс для работы с файлом:

Конструктор класса открывает файл, а деструктор – закрывает. В этом случае есть гарантия, что при возникновении ошибки файл будет корректно закрыт, и информация не будет утеряна.

Исключение может быть как стандартного, так и определенного пользователем типа. При этом нет необходимости определять этот тип глобально – достаточно, чтобы он был известен в точке порождения исключения и в точке его обработки.

Класс для представления исключения можно описать внутри класса, при работе с которым оно может возникать. Конструктор копирования этого класса должен быть объявлен как public, поскольку иначе будет невозможно создать копию объекта при генерации исключения (конструктор копирования, создаваемый по умолчанию, имеет спецификатор public).

ФУНКЦИИ

C / C++

Функции

Формат описания Функции:

```
[класс] <возвращаемый_тип> <имя_функции>
        ([<тип1> <имя_формального_параметра1>, …,
          <типN> <имя_формального_параметраN>])
      [throw (исключения)]
{
 <тело_функции >
 return <возвращаемое_значение>;
}
```

где - класс – extern *или* static – *явно задает область видимости функции: глобальная (умолчание) или в пределах модуля;*

- исключения – обрабатываемые функцией исключения.

Обработка исключений

Список исключений функции

В заголовке функции можно задать список исключений, которые она может прямо или косвенно порождать. Поскольку заголовок является интерфейсом функции, указание в нем списка исключений дает пользователям функции необходимую информацию для ее использования, а также гарантию, что при возникновении непредвиденного исключения эта ситуация будет обнаружена.

Типы исключений перечисляются в скобках через запятую после ключевого слова throw, расположенного за списком параметров функции, например:

```
void f1() throw (int, const char*)  { <Тело функции> }
void f2() throw (Oops*){  <Тело функции> }
```

Функция f1 должна генерировать исключения только типов int и const char*.
Функция f2 должна генерировать только исключения типа указателя на класс Oops или производных от него классов.
Если ключевое слово throw не указано, функция может генерировать любое исключение.
Пустой список означает, что функция не должна порождать исключений:

```
void f() throw () {  // Тело функции, не порождающей исключений }
```

Исключения не входят в прототип функции. При переопределении в производном классе виртуальной функции можно задавать список исключений, такой же или более ограниченный, чем в соответствующей функции базового класса.

Указание списка исключений ни к чему не обязывает – функция может прямо или косвенно породить исключение, которое она обещала не использовать. Эта ситуация обнаруживается во время исполнения программы и приводит к вызову стандартной функции unexpected, которая по умолчанию просто вызывает функцию terminate. С помощью функции set_unexpected можно установить собственную функцию, которая будет вызываться вместо terminate и определять действие программы при возникновении непредвиденной исключительной ситуации.
Функция terminate по умолчанию вызывает функцию abort, которая завершает выполнение программы. С помощью функции set_terminate можно установить собственную функцию, которая будет вызываться вместо abort и определять способ завершения программы.
Функции set_unexpected и set_terminate описаны в заголовочном файле <exception>.

Обработка исключений

Исключения в конструкторах и деструкторах

Язык C++ не позволяет возвращать значение из конструктора и деструктора.
Механизм исключений дает возможность сообщить об ошибке, возникшей в конструкторе или деструкторе объекта.
Для иллюстрации создадим класс Vector, в котором ограничивается количество запрашиваемой памяти:

```
class Vector  { public
   { class Size{};                    // Класс исключения
     enum {max = 32000}:              // Максимальная длина вектора
     Vector(int n)                    // Конструктор
       { if (n<0 || n>max) throw Size(); ... }
     …
   };
```

При использовании класса Vector можно предусмотреть перехват исключений типа Size:

```
try
 {
   Vector *p = new Vector(i):
   …
 }
 catch (Vector::Size)
   {  ...  // Обработка ошибки размера вектора }
```

В обработчике может использоваться стандартный набор основных способов выдачи сообщений об ошибке и восстановления. Внутри класса, определяющего исключение, может храниться информация об исключении, которая передается обработчику. Смысл этой техники заключается в том, чтобы обеспечить передачу информации об ошибке из точки ее обнаружения в место, где для обработки ошибки имеется достаточно возможностей.
Если в конструкторе объекта генерируется исключение, автоматически вызываются деструкторы для полностью созданных в этом блоке к текущему моменту объектов, а также для полей данных текущего объекта, являющихся объектами, и для его базовых классов.
Например, если исключение возникло при создании массива объектов, деструкторы будут вызваны только для успешно созданных элементов.
Если объект создается в динамической памяти с помощью операции new и в конструкторе возникнет исключение, память из-под объекта корректно освобождается.

Обработка исключений

Обработка производных исключительных ситуаций

Если исключительные ситуации описываются с помощью базового и производных классов, при работе с операторами catch следует проявлять максимальную осторожность, поскольку оператор catch, соответствующий базовому классу, одновременно соответствует всем производным классам.
Таким образом, если необходимо перехватить исключительные ситуации базового и производных классов, в последовательности операторов catch производный класс следует обрабатывать первым. Если этого не сделать, оператор catch, соответствующий базовому классу исключительной ситуации, также будет перехватывать исключительные ситуации всех производных классов.

Рассмотрим следующую программу:

```
// Перехват производных классов
#include <iostream>
using namespace std;
    class B { };
    class D: public B { } ;
 int main() {
   D derived;
   try
     { throw derived; }
catch(B b) {
cout << "Перехват базового класса.\n";
}
catch(D d) {
cout << "Этот оператор не выполняется.\n";
}
return 0;
}
```

Поскольку объект derived является экземпляром класса, производного от класса B, он будет перехвачен первым оператором catch, а второй оператор catch никогда выполняться не будет.
Некоторые компиляторы в таких случаях выдают предупреждение. Другие компиляторы вообще считают это ошибкой. Так или иначе, чтобы исправить эту ситуацию, следует поменять порядок следования операторов catch.

Обработка исключений

Функции terminate() и unexpected()

Функции terminate() и unexpected() вызываются в крайних случаях, т.е. когда обработка исключительной ситуации выполняется неверно. Эти функции принадлежат стандартной библиотеке языка C++.

Их прототипы показаны ниже:

```
void terminate()
void unexpected()
```

Для вызова этих функций необходим заголовок <exception>.

Функция terminate() вызывается, если подсистема обработки исключительных ситуаций не может обнаружить подходящий оператор catch. Кроме того, она вызывается, если программа пытается повторно сгенерировать исключительную ситуацию, которая ранее никогда не генерировалась. Функция terminate() вызывается также во многих других, более запутанных ситуациях.

Например, деструктор уничтожаемого объекта генерирует исключительную ситуацию в процессе раскручивания стека, который выполняется при генерировании другой исключительной ситуации. Как правило, функция terminate () является последним средством обработки исключительной ситуации, если никакой другой обработчик не подходит.

По умолчанию функция terminate() вызывает функцию abort().

Функция unexpected() вызывается при попытке генерировать исключительную ситуацию, не указанную в разделе throw. По умолчанию функция unexpected() вызывает функцию terminate().

Обработка исключений

Применение обработки исключительных ситуаций

Система обработки исключительных ситуации позволяет реагировать на необычные события, возникающие в ходе выполнения программы.
Следовательно, обработчики исключительных ситуаций должны выполнять некие разумные действия, позволяющие исправить ошибку или смягчить ее последствия.
Рассмотрим в качестве примера следующую простую программу.
Она вводит два числа и делит первое из них на второе. Для предотвращения деления на нуль применяется обработка исключительной ситуации.

Этот очень простой пример иллюстрирует принцип обработки исключительных ситуаций. Если знаменатель равен нулю, возникает исключительная ситуация. Ее обработчик не предусматривает деления (это привело бы к аварийному завершению работы программы), а просто сообщает пользователю о возникшей ошибке.
Таким образом, деления на нуль можно избежать, продолжив выполнение программы. Эта схема работает и в более сложных случаях.

```
#include <iostream>
using namespace std;
  void divide(double a, double b);
int main()
 { double i,   j;
  do
    {
     cout<<"Введите числитель
              (0 означает выход): ";
     cin >> i;
 cout<<"Введите знаменатель: ";
 cin >> j;
  divide(i,  j);
   } while(i  != 0);
return 0;
}
 void divide(double a, double b)
 {
 try
  { if(!b)  throw b; //Проверка деления на нуль
    cout << "Результат: " << a/b << endl;
  }
  catch (double b)
  { cout <<"Делить на нуль нельзя.\n"; }
}
```