|  |
| --- |
| **Page 1** |

**лекция 11**

**Тема лекции:** Использование исключительных ситуаций. возбуждение исключительных ситуаций. Синтаксис конструкций, используемых для обработки исключений

***исключительные ситуации***

Исключительная ситуация - это условие исключительной ситуации, что требует специальной обработки. Исключительные ситуации лучше всего использовать для обработки ошибок, возникающих при исполнении, но их применение на этом не ограничивается.

Для возбуждения исключительные ситуации оператор посылает объект, описывающий суть исключительной ситуации. Объект может быть литеральным значением, строкой, объектом класса или любого другого объекта.

Для обработки исключительной ситуации некоторый оператор захватывает условие,

послана другим процессом. Операторы, перехватывающие исключительные ситуации, называются обработчиками исключений. Программы готовятся к перехвату исключительных ситуаций, испытывая один или несколько процессов, возбуждающие исключительные ситуации. Для использование исключительных ситуаций испытывается один или несколько операторов и перехватывается любая исключительная ситуация, возбуждается этими операторами. Обнаружив состояние ошибки, функция может возбудить исключительные ситуации вызывает такие последствия:

• Функция объявляет, что возникла условие исключительной ситуации. Это может быть

как ошибкой, так и другим обстоятельством, что требует обработки.

• Функция спрашивает решения проблемы обработки исключительной ситуации.

Обработчик, если он существует, вызывается автоматически в ответ на посылку

объекта исключительной ситуации. В программе возбуждается исключительные ситуации посредством выполнения оператора *throw,* обычно внутри функции: throw "Overflow";

В другом месте программы обработчик строчных исключительных ситуаций может перехватить и отобразить на экране посланый объект. В обработчике задается тип объекта в выражении:

catch (const char \* message)

{

cout << "Error! "<< Message;

....

}

Оператор *catch* ловит посланы объекты строчных исключительных ситуаций и отображает их оператором вывода в поток. Если дальнейшие действия не определены в блоке *catch,* то программа продолжит свою работу после *catch.* можно также аварийно прервать выполнение программы, вызвать другую функцию или продолжить цикл для повторного выполнения действий, вызвавших проблему. Исключительные ситуации является механизмом для сообщений и принятие мер в случае возникновения условия исключительной ситуации. Исключительные ситуации не навязывают необходимых действий. Их обработка полностью зависит от программиста. Функции исключений не ограничены обработкой ошибок. К примеру, пустой учетный объект может сообщать о том, что он пуст, путем возбуждения исключительной ситуации. Произошла ошибка или что-то другое, зависит только от того, как программист назначит аварийное прерывание выполнения программ. Но, поскольку использование исключительных ситуаций может привести к двусмысленности в программах, их лучше использовать для принятия мер в случае возникновения реальных ошибок, которые могут заставить программу прервать процесс и привести к аварийному завершению или неверных результатов.

Исключительные ситуации представляются объектами, очень похожи на аргументы функций. Это могут быть объекты любого типа. Чаще всего – это экземпляры класса. Например, можно объявить класс *Overflow:*

class Overflow

{

...

};

Можно послать экземпляр класса для возбуждения исключительные ситуации:

throw Overflow;

В этом операторе создается объект класса *Overflow,* что потом ссылается назад, в место вызова функции. В другом месте программы можно перехватить исключительную ситуацию с помощью оператора *catch:*

catch (Overflow)

{

cout << "Сообщение";

}

Одна только присутствие объекта *Overflow* свидетельствует о возбуждении исключительной ситуации. Объект не обязан предпринимать каких действий (хотя может).

Возможен также следующая запись:

catch (Overflow overObject) {};

Объект в операторе *catch* получил имя *overObject.* В операторах внутри *catch* можно использовать объект *overObject* так же, как и параметр функции. Обычно в классе исключительные ситуации реализуются функциями-членами, которые можно вызвать в выражениях внутри *catch:*

class Overflow

{

void Report ()

{

cout << "Error";

}

};

В классе *Overflow* объявляется функция-член *Report (),* который отображает сообщение об ошибке. Функция может быть не только встроенной, но и функцией, вызывается. В операторе *catch* можно вызвать функцию *Report ()* объекта исключительной ситуации для отображения сообщения о ошибке

catch (Overflow overObject)

{

overObject.Report ();

}

В функциях можно возбуждать объекты исключительных ситуаций различных типов

для поддержки различных условий исключительных ситуаций:

int AnyFunction ()

{

if (condition1) throw "Big !!! ";

if (condition2) throw Overflow;

return 123;

}

Если условие *condition1* справедливая, функция возбуждает строчную исключительную

ситуацию с сообщением *"Big !!!".* Если выполняется условие *condition2,* функция посылает объект класса *Оverflow,* созданный, в данном случае, по помощью конструктора класса по умолчанию. Если же функция не выявила ошибок, она возвращает значение 123.

Итак:

• Функции могут возбуждать одну или несколько исключений различных типов, представляющие различные условия исключительных ситуаций

• Возбуждение исключительной ситуации немедленно завершает выполнение функции, в

которой выполняется оператор *throw.*

• Исключительные ситуации обеспечивают альтернативный механизм возврата для функций.

Функция *AnyFunction ()* обычно возвращает целое значение, однако, если возникает исключительная ситуация, она возвращает строку или объект класса *Overflow.* Только операторы *catch* могут воспринять эти значения. Можно поддерживать несколько типов объектов исключительных ситуаций с помощью серии операторов *catch.* Для обработки исключительной ситуации для класса *Error* и строк можно написать:

catch (Error e) {}

catch (const char \* message) {}

***Введение в блоки try***

Для того, чтобы принимать объекты исключительных ситуаций некоторой функции, ее

нужно вызвать внутри блокy *try:*

try

{

int x = AnyFunction ();

}

catch (Error e)

{

// Если функция возбудила исключительную ситуацию вызвать

e.Report (); // Report () объекта Еrror

exit (-1)

}

Блок *try* содержит один или несколько операторов, исключительные ситуации которых

нужно перехватывать. Один или более операторов *catch* должны следовать по блоку *try.*

Несколько блоков *try* и соответствующих им операторов *catch* могут быть объявлены вложенными, хотя синтаксис в этом случае может привести к запутанного кода.

try

{

Function ();

try

{

Function ();

}

catch (Error e)

{

e.Report ();

}

}

catch (Error e)

{

e.Report ();

}

Здесь если в функции *Function ()* возбуждается исключительная ситуация, программа

вполне пропускает вложенный блок *try.* В блоке *try* может быть несколько операторов:

try

{

cout << "Сообщение";

int x = AnyFunction ();

cout << x;

}

В блоке *try* сначала отображается сообщение. Затем в нем вызывается функция *AnyFunction (),* результат которой присваивается целой переменной *x.* Если внутри функции *AnyFunction ()* возникла исключительная ситуация, то блок *try* немедленно завершается. Таким образом, любое условие исключительной ситуации приводит к пропуску присвоение *x* и завершающего оператора вывода.

По блоку *try* обязательно должны следовать один или несколько операторов *catch:*

catch (char \* message)

{

cout << "Error !!! "<< Message;

exit (-1)

}

catch (Overflow)

{

cout << "Overflow! ";

exit (-2)

}

Сначала испытывается вызов функции *AnyFunction ().* если функция завершилась нормально, ее результат присваивается переменной *x,* и выводится. Если исключительной ситуации не было, оба оператора *catch* пропускаются, поэтому что нет объектов исключительных ситуаций объявленных типов, которые нужно было бы перехватывать. Если функция *AnyFunction ()* возбуждает исключительные ситуации, выполнения блока *try* немедленно прерывается, посланные объекты перехватываются соответствующими операторами *catch.* В примере в операторах *catch* вызывается библиотечная функция *exit ()* для аварийного завершения программы. Она необязательна. Два оператора *catch* обрабатывают исключительные ситуации для строк и

класса *Overflow.* Любые другие типы исключений, не обработанные оператором *catch,* передаются вверх по цепочке вызовов. К примеру, предположим, что функция *AnyFunction ()* также возбуждает исключительные ситуации типа *NewException.* Если функция *g ()* вызывает код функции *AnyFunction (),* то объект этой исключительной ситуации будет передаваться вверх, в *g (),* так как в этом коде обрабатываются только исключительные ситуации типов *char \** и *Overflow.* если ни из операторов *catch* нужного типа не было найдено в цепочке вызовов, ведут к исключительной ситуации, выполнение программы прерывается вызовом специального обработчика исполняющей системы.

*Объявления исключительных ситуаций.* С помощью альтернативной формы объявление функции можно задавать типы исключительных ситуаций, которые разрешается возбуждать в данной функции. Например, функция *AnyFunction ()* может задать типы своих исключительных ситуаций следующим образом:

void AnyFunction () throw (Error)

Выражение *throw (),* следующего за именем функции и списком ее параметров указывает, что функция *AnyFunction ()* может возбуждать исключительные ситуации типа *Error.* Это объявления указывает компилятору, что функции *AnyFunction ()* не допускается возбуждать исключительные ситуации других типов. Для задания функции, возбуждает исключительные ситуации нескольких типов, надо перечислить типы данных локальных

объектов исключительных ситуаций:

void AnyFunction () throw (Error, char \*, OtherType)