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

**лекция 12**

**Тема лекции:** Необработанные исключительные ситуации. Замена функций unexpected () И terminate (). Исключительные ситуации и локальные объекты. исключительные ситуации и конструкторы. Исключительные ситуации и распределение памяти

***Необработанные исключительные ситуации***

Необработанные исключительные ситуации передаются вверх по цепочке вызовов одних функций другими, пока не встретится подходящий им оператор *catch* или пока больше не останется непроверенных обработчиков исключений. Если произойдет последнее, для обработки исключительной ситуации вызывается одна из трех специальных функций, которые автоматически присоединяются к каждой программе на С ++, использующий исключительные ситуации. Эти функции имеют имена *unexpected () terminate (), abort ().*

Они вызываются в соответствии с правилами:

Если в программе возникли исключительные ситуации, не обрабатываются оператором *catch,* то вызывается функция unexpected *().* Исключительные ситуации, которые не обрабатываются одним оператором *catch,* называются непредвиденными исключительными ситуациями. По умолчанию *unexpected ()* вызывает функцию *terminate ().* Непредвиденные исключительные ситуации могут возбуждаться программой при обнаружении повреждения стека дескриптором класса, в результате чего вызывается функция *terminate().* По умолчанию функция *terminate ()* вызывает функцию *abort ().*

Функция *abort()* немедленно прерывает выполнение программы. Она никогда не вызывается непосредственно. Если не обработаны все возможные исключительные ситуации и и не будут приняты меры для перепрограммирования функций *unexpected ()* и *terminate()* необработанная исключительная ситуация аварийно прервет выполнение программы путем обращения к *abort ().*

Можно заменить функции *unexpected ()* и *terminate ()* своим кодом для обработки необработанных исключений. Например, имеет смысл заменить функцию *unexpected ()* для сообщения о любых необработанные исключительные ситуации на этапе разработки программы.

Это может помочь выявить отсутствуют обработчики ошибок в программе. В других случаях, можно заменить *terminate ()* Диагностируя кодом для просмотра памяти, чтобы выявить операторы, которые разрушают кучу. Заменить функцию *abort ()* нельзя. Ее вызов всегда приводит к завершению программы.

Для задания адреса собственной функции-обработчика непредвиденных исключений надо использовать функцию *set\_unexpeсted ()* объявленную в заголовочный файл EXCEPT.H. Функция должна иметь тип *unexpected\_function,* не иметь аргументов и ничего не возвращать. Обработчик пользователя непредвиденных исключений может возбуждать исключительную ситуацию. В этом случае поиски оператора *catch* начинаются с того места, откуда сначала было вызвано обработчик. Для задания собственной функции-обработчика завершения программы надо использовать функцию *set\_terminate ()* также объявленную в заголовочном файле EXCEPT.H. Функция должна иметь тип *terminate\_function,* не иметь параметров, ничего не возвращать. Обе функции *set\_unexpected ()* и *set\_terminate ()* возвращают адрес текущей функции-обработчика. можно сохранить и затем восстановить существующие обработчики, запомнив их адреса в переменных, а затем передавая их обратно функциям.

Например, сначала объявляется прототип функции-обработчика:

void unexpectedHandler ();

Затем устанавливается обработчик:

set\_unexpected (unexpectedHandler)

Хранение и восстановление адреса старой функции-обработчика:

unexpected\_function savedAddress ();

savedAddress = set\_unexpecred (unexpectedHandler)

// новый обработчик

set\_unexpected (saveAddress) // восстановления

// новый обработчик больше не используется

Функция пользователя *terminate ()* устанавливается аналогично, только вместо *set\_unexpected ()* вызывается функция *set\_terminate (),* который возвращает адрес типа *terminate\_function.*

Пример программы установки обработчиков непредвиденных исключительных ситуаций. В программе также демонстрируется приемлемый способ обработки непредвиденных исключений неизвестных типов, которые могут возбуждаться в плохо документированных библиотечных функциях.

#include <iostream.h>

#include <exсeрt.h>

#define MAXERR 10

class MaxError {};

class Error

{

public:

Error ();

void Say ();

private:

static int count;

};

void Run () throw (Error)

void trapper ();

void zapper ();

int Error :: count;

void main ()

{

set\_unexpected (trapper)

set\_terminate (zapper)

for (;;)

{

try {Run ();

}

catch (Error e)

{

e.Say ();

}

}

void Run () throw (Error)

{

throw Error ();

// throw "Неизвестный тип объекта";

}

void trapper ()

{

cout << "Отделочник непредвиденных ситуаций. . ";

throw Error ();

}

void zapper ()

{

cout << "Отделочник завершения функции";

exit (-1)

}

Error :: Error ()

{

count ++;

if (count> MAXERR)

throw MaxError ();

}

void Error :: Say ()

{

cout << count << '\ n';

}

При запуске *unexpected.cpp* выведет 10 сообщений об ошибках перед завершением. В программе используются два класса исключительных ситуаций. Объект класса *MaxError,* что не располагает данными и функций, ссылается, когда число ошибок превышает константу *MAXERR,* заданную равной 10.

Конструктор класса *Error* инкрементирует статический член класса *count.* Член *count* - статический, поэтому существует только один экземпляр этого значения и он существует

до тех пор, пока программа не завершится. Член *count* - закрытый член класса, следовательно он не изменится в операторах программы вне класса. Функция-член *Say ()*

отражает значение счетчика ошибок программы. Если член *count> = MAXERR,* конструктор класса *Error* возбуждает исключительную ситуацию типа *MaxError.* При возбуждении исключительной ситуации объект класса не создается, поэтому создание объекта класса *Error* прерывается. В функции *main ()* устанавливаются функции-обработчики, замещающих за умолчанию функции *unexpected () и terminate ().* затем выполняется бесконечный цикл *for.* Внутри цикла в блоке *try* вызывается функция *Run ()* и оператор *catch* перехватывает все объекты класса *Error,* что посылает *Run ().* При возбуждении исключительной ситуации *catch* отражает текущее значение счетчика ошибок путем обращения к функции *Say ()* посланного объекта.

Функция *Run ()* всегда возбуждает исключительные ситуации, моделируя возникновения нескольких ошибок. Пользовательский обработчик непредвиденных исключений *trapper ()* выводит сообщение о вызове этой функции. Это происходит после возбуждения 10 исключительных ситуаций. Обработчик непредвиденных исключений может возбудить еще одно исключительное ситуацию для продолжения программы. В примере оператор

throw Error ();

посылает новый экземпляр класса *Error.* Создание объекта класса *Error* заставляет конструктор этого класса возбудить еще одну ситуацию типа *MaxError.* этот тип ошибки не поддерживается в операторе *catch,* поэтому вызывается обработчик завершения программы *zapper ().* Функция завершения не может возбудить исключительные ситуации, а также не может выполнить оператор *return.*

Перепрограммируем функцию *Run ()* так, чтобы в ней возбуждалась исключительные ситуации неизвестного типа. Например, в качестве неизвестного типа может выступить строку.

Когда программа скомпилирует и запустит обработчик непредвиденных ситуаций *trapper (),* то он будет вызываться для каждой исключительной ситуации возбуждается функцией *Run ().* Это происходит потому, что не существует оператора *catch* для объектов исключительной ситуации типа *char \** или *const char \*.* Функция *trapper ()* однако, транслирует неопознанные объекты исключительной ситуации, посылая объект известного типа, в данном случае, типа *Error.* Новая возбуждено исключительная ситуация продолжает выполнение программы в операторе *catch* внутри функции *main (),* который обрабатывает транслируемую исключительную ситуацию. В конце концов, в программе максимальное число ошибок будет превышен, и объект класса *MaxError* будет послан конструктором класса *Error.* В результате вызовется обработчик завершения программы путем обращения к библиотечной функции *exit ().*

Листинг результата в первом случае (в *Run ()* используется *throw*

*Error ();* ):

1

2

...

10

Обработчик непредвиденной ошибки

Обработчик завершения функции

Во втором случае (в *Run ()* - строка):

Обработчик неизвестной ошибки

1

Обработчик неизвестной ошибки

2

...

10

Обработчик неизвестной ошибки

Обработчик завершения функции

***Исключительные ситуации и локальные функции***

С помощью специальной опции компилятора локальные объекты, остались в стеке, автоматически уничтожаются возбуждением исключительной ситуации, то есть вызываются деструкторы этих объектов. Для разрешения автоматического уничтожения нужно задать опцию *-xd* для автономного компилятора. Или интегрированной среды выбрать *Options / Project,* открыть пункт *С ++ Options* и выбрать пункт *Exception handling / RTTI. RTTI - Runtime Type Information* - информация о времени выполнения. Должны быть также установлены пункты *Enable exception,* *Enable destructor cleanup.* Для Object-Windows программ обязательно нужно устанавливать автоматическое очистки. Автоматическое уничтожение необходимо для функций, которые создают локальные объекты и возбуждают исключительные ситуации после их создания. Пример: эта функция может оставить объект класса *TAnyClass* в стеке:

int AnyFunction ()

{

AnyClass object (123)

if (condition) throw Error ();

return object.Value ();

}

Функция создает в классе объект класса *TAnyClass.* Если события развиваются нормально, при завершении функции вызывается деструктор объекта. но если функция завершается исключительной ситуацией, деструктор ещё не вызывается, что может привести к серьезным проблемам. Использование ключа *- xd* или аналогичных установок в интегрированной среде гарантирует, что деструктор для всех объектов будет вызван после завершения функции в результате возбуждения исключительной ситуации. Указатели на объекты не удалятся автоматически.

int AnyFunction ()

{

AnyClass \* p = new AnyClass (123)

if (condition)

{

delete p;

throw Error ();

}

delete p;

return 123;

}

Избежать дублирования операторов *delete* можно только тогда, когда

исключительная ситуация возбуждается после уничтожения динамического объекта:

int AnyFunction ()

{

AnyClass \* p = new AnyClass (123)

delete p;

if (condition) throw Error ();

return 123;

}

***Исключительные ситуации и конструкторы***

Конструкторы класса могут возбуждать исключительные ситуации для сигнализации о том, что они не могут создать объект. пример:

class AnyClass

{

public:

AnyClass ()

{

if (condition) throw Error ();

}

~ AnyClass () {}

};

Если конструтор возбуждает исключительную ситуацию, что означает ненормальное

завершения конструктора, то деструктор этого объекта вызывается. Рассмотрим случай:

class AnyClass

{

OtherClass x;

public:

AnyClass (): x (123)

{

if (condition) throw Error ();

}

~ AnyClass () {}

};

Объект *x* типа *OtherClass* создается конструктором *AnyClass ()* до того,

как будут выполняться операторы в теле конструктора. если конструктор

*OtherClass ()* возбуждает исключительную ситуацию, другие операторы конструктора НЕ

выполняются. Поскольку код конструктора НЕ исполнился, объект класса *AnyClass*

не создается и, следовательно, деструктор ~ *AnyClass ()* не вызывается.

Класс может возбудить исключительные ситуации своего же типа. Обычно это делается

с помощью функции-члена, которую часто называют *Raise ().* К примеру:

class Error

{

public:

void Raise ()

{

throw Error ();

}

};

Если *e* - объект класса *Error,* то оператор

e.Raise ();

возбуждает исключительную ситуацию, посылая еще один объект этого же класса. В зависимости от предоставляемого сервиса классом *Error,* создание нового объекта исключительной ситуации может лучше удовлетворить запросы пользователя, чем повторное возбуждение исключительной ситуации и с уже существующим объектом:

throw е;

***Исключительные ситуации и управления распределением памяти***

Оператор *new* возбуждает исключительную ситуацию, если не может удовлетворить

запрос выделения памяти. Тип исключительной ситуации - *xalloc,* определенный в

заголовочный файл EXCEPT.H следующим образом:

class xalloc: public xmsg

{

public:

xalloc (const string & msg, size\_t size)

size\_t requested () const;

void raise () throw (xalloc)

private:

size\_t siz;

};

Для использования исключительной ситуации вместе с *new* нужно заключить операторы, выделяют память, в блок *try,* за которым должен следовать оператор *catch* для объекта класса *xalloc.* К примеру:

try

{

\* p = new AnyClass ();

}

catch (хalloc x)

{

cout << "Mеmory error:" << x.why ();

exиt (-1)

}

Класс *xalloc* выводится из класса *xmsg,* объявленного в том же заголовочный файл. Класс-предок содержит текстовое сообщение типа *string.* Для доступа к этой строки используется функция-член *why ().* Для определение размера неудачного запроса выделения памяти в байтах надо использовать функцию *x.requested ().* Можно использовать один блок *try* в функции *main ()* для обработки всех возникающих ошибок.

void main ()

{

for (;;)

{

try

{

Run ();

}

catch (Хalloc x)

{

cout << "Out of memory";

exit (-1)

}

}

}

Здесь идея заключается в вызове единой функции *Run (),* что запускает программу.

Любые исключительные ситуации будут обрабатываться в функции *Run (),* или в функции

*main ().*