Исключения - некоторое событие, которое является неожиданным или прерывает нормальный процесс выполнения программы.

Исключительные ситуации можно разделить на 2 основных типа:

1) синхронные - могут возникать только в определенных, заранее известных точках программах.

2) асинхронные - возникают в любой момент времени и не зависят от программной реализации

Для обработки используются 2 механизма

1) с возвратом

2) без возврата

В любом случае программа должно определить аварийную точку, выявить причину ошибки и передать управление соответствующему обработчику ошибки.

1) - обработчик устраняет и приводит программу к состоянию, в котором она продолжает работу по основному алгоритму

2) - после устранения - управление передается в заранее заданное место программы.

Существует 2 варианта подключения обработчика исключительных ситуаций к программе: структурная и неструктурная обработка.

* Неструктурная – реализуется в виде механизма регистрации функции для каждого возникшего типа исключения. Системные библиотеки предоставляют процедуры: регистрация + удаление. Независимо от того какая программа работает, вызывается последний зарегистрированный обработчик. С помощью такой обработки – анализ асинхронных исключений, для синхронных – она неудобна.
* Структурная обработка – требует обязательной поддержки со стороны языка программирования.

Языковая конструкция содержит 2 блока:

1. защищаемый – блок, где может возникнуть ошибка.
2. Обработчик – операции при возникновении ошибки.

Если ошибка входит в исключения, то системная ошибка не генерируется. Обработчики исключений могут описываться по-разному, но работают одинаково. При ошибке находится 1 подходящий обработчик, работа контролируемого блока завершается, и управление передается обработчику.

Обработка SEH (Structured Exception Handling)

\_\_try – начало защищаемого блока

\_\_except – определяет выражение – фильтр, после которого реализуется блок обработки исключения.

Внутри защищаемого блока исключение может быть специально вызвано. Для этого используется вызов \_\_leave - передает управление в конец защищаемого блока, прекращает дальнейшее выполнение основного алгоритма.

При выполнении блока \_\_finally можно проанализировать код завершения защищаемого блока.

Если защищаемый блок завершился без завершения исключения, то возвращается результат функцией AbnormalTerminantion().

Если результат отличен от нуля, блок завершился без ошибок. Блок finally будет выполняться всегда кроме тех случаев, когда завершение программы вызвали системные функции: append и TerminateProccess.

Значения, возвращаемые функцией-фильтром.

EXCEPTION\_CONTINUE\_EXECUTION (-1) Возврат к месту вызова исключения и продолжение выполнения ««без обработки»».

EXCEPTION\_CONTINUE\_SEARCH (0) Исключение не распознано, будет продолжен поиск обработчика в стеке, в первую очередь среди выражений try- except.

EXCEPTION\_EXECUTE\_HANDLER (1) Исключение распознано, передача управления в блок \_\_except.

Использование SEH пример:

#include <stdio.h>

#include <windows.h>

#include <excpt.h>

int filter(unsigned int code, struct \_EXCEPTION\_POINTERS\* ep)

{

puts("in filter.");

if (code == EXCEPTION\_ACCESS\_VIOLATION)

{

puts("caught AV as expected.");

return EXCEPTION\_EXECUTE\_HANDLER;

}

else

{

puts("didn't catch AV, unexpected.");

return EXCEPTION\_CONTINUE\_SEARCH;

};

}

int main()

{

int\* p = 0x00000000; // Указатель на ноль

puts("hello");

\_\_try

{

puts("in try");

\_\_try

{

puts("in try");

\*p = 13;// Это выражение вызовет исключение, которое будет обработано

}

\_\_finally

{

puts("in finally. termination: ");

puts(AbnormalTermination() ? "\tabnormal" : "\tnormal");

}

}

\_\_except (filter(GetExceptionCode(), GetExceptionInformation()))

{

puts("in except");

}

puts("world");

}

Вывод в консоль после выполнения кода в примере :

hello

in try

in try

in filter.

caught AV as expected.

in finally.termination:

abnormal

in except

world

Плюсы SEH:

1) позволяет отлавливать широкий спектр исключений – деление на ноль, переполнение стека

2) обработка исключений ведется на уровне ядра ОС

3) возможно использование исключений без подключения внутренних механизмов С++, что позволяет сократить размер программы и использовать функции ядра.

Минусы:

1) плохая совместимость с С++, т.к. данный механизм реализован на уровне ядра он не воспринимает внутренних структур С++: не видит созданных классов и не освобождает выделенную им память

2) SEH не вызывает деструктуры пользовательских классов

3) невозможно использовать одновременно механизм SEH и стандартное исключение Си.

Стандартное исключение С++ реализовано через механизм CRTL. Данный механизм принято считать окончательной моделью управления. После того, как исключение произошло, обработчик исключений не может потребовать, чтобы произошел возврат управления в точку возникновения ошибки. Это приводит к тому, что с помощью CRTL могут обрабатываться только синхронные исключения – исключения, происходящие внутри программы, аппаратные и асинхронные исключения не поддерживаются. В языке С++ практически любое состояние достижимое в ходе выполнения вычислительного процесса может быть определенно как исключение. Они заданны в языке для того чтобы дать возможность программисту динамически отрабатывать возникшие ситуации. Анализ и обработка исключений переносится из точки возникновения в специально предназначенные для обработки блоки, при этом возможна передача обработчику набора необходимых параметров, который характеризуют возникшую ситуацию. При CRTL выделяются 2 блока:

1. Защищенный блок после try
2. Блок обработчика после catch

Внутри блока обработчика может задаваться повторная генерация исключения с помощью вызова throw. При возникновении ошибки в любом из операторов защищаемого блока управление передается соответствующему обработчику блока catch при этом генерируется исключение конкретного типа и кода. После выполнения соответствующего обработчика управление может быть передано в завершающую точку программы. В случае если исключение не происходит управление передается оператору расположенному за блоком catch. Если внутри блока catch используется операция throw то происходит повторное возбуждение исключения. Текущая функция завершается и исключение передается обработчику верхнего уровня. Т.О. используется структурная вложенная обработка исключений, которая позволяет для каждого исключения, возникшего внутри контролируемого блока, найти обработчик либо внутри текущей функции либо выше по списку вызовов. В наихудшем случае обработку исключения выполнит ОС.

В С++ используется несколько форм обработчика исключений:

1. Полная форма, когда после ключевого слова catch (тип имя) {обработчик}

Данная форма используется для обработки исключений, если известно состояние переменных в момент генерации исключений.

1. Catch (тип) {обработчик}

Используется для обработки исключений, для которых известен только факт возникновения исключений (деление на 0)

1. Catch (…) {обработчик}

Перехватывает любое исключение возникающее в программе.

Одновременно в рамках одной функции может использоваться несколько обработчиков catch, при этом выбор соответствующего обработчика осуществляется путем последовательного просмотра. Следовательно, наиболее детализированные обработчики должны определяться раньше обработчика без параметров.

Генерация исключения может быть выполнена в любой точке программы. Для этого используют ключевое слово throw, после которого определяют тип возбуждаемого исключения. В этом случае исключение формируется как статический объект, значение которого определенно выражением генерации. Копия объекта передается за пределы контролируемого блока и инициирует переменную, используемую в спецификации обработчика исключений. Копия объекта сохраняется до тех пор, пока не будет начата обработка данного исключения. В ряде случаев используется вложение контролируемых блоков, тогда преимущественно обработка исключений должна выполняться на внешнем уровне. При этом желательно использовать ретрансляцию исключений (catch (…)).

Пример :

Предположим, GetNetworkResource() получает данные через сетевое подключение, а 2 типа исключений являются определенными пользователем классами, производными от std::exception. Обратите внимание, что исключения перехватываются по const ссылке в catch инструкции. Рекомендуется создавать исключения по значению и захватывать их ссылкой константы.

MyData md;

try {

// Код который может вызвать исключения

md = GetNetworkResource();

}

catch (const networkIOException& e) {

// Код, который выполнится после возникшего в блоке try

// исключения типа networkIOException

// Вывод сообщения об ошибке

cerr << e.what();

}

catch (const myDataFormatException& e) {

// Другое исключение

cerr << e.what();

}

// Генерируем исключения в функции для того чтобы их вызвать в блоке try и поймать в блоке catch

MyData GetNetworkResource()

{

// ...

if (IOSuccess == false)

throw networkIOException("Unable to connect");

// ...

if (readError)

throw myDataFormatException("Format error");

// ...

}

Иногда в программе приходится отслеживать непредвиденно или не специфицированное исключение. В этом случае используют функцию unexpected. Вызов terminate приводит к корректному завершению работы программы. Функция unexpected с помощью функции установки может определить собственный обработчик, который будет вызываться, если программа выбрасывает не специфицированные исключения. В качестве параметра функция set\_ unexpected определяет адрес функции обработчика. Если функция unexpected не определенна любое непредвиденное исключение будет передано в ОС.

#include <exception>

#include <iostream>

using namespace std;

void uefunction()

{

cout << "My unhandled exception function called." << endl;

terminate(); // Это то что программа вызовет по умолчанию

}

int main()

{

unexpected\_handler oldHandler = set\_unexpected(uefunction);

// Возвращаемое значение – старый обработчик

unexpected(); // Вызов функции которая выполнит

// обработку неизвестного исключения.

}

Функцию terminate тоже можно переопределить аналогичным способом.