ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ

ЛЕКЦИЯ №4

Как обрабатывать алгоритмические ошибки?

1. Вернуть результат операции явно (операция успешна / операция не успешна)
2. Вернуть результат выполнения операции как один из параметров (по ссылке)

Пример

RETURN_ERROR.CPP

Exceptions

Для реализации механизма обработки исключений в язык Си++ введены следующие три ключевых (служебных) слова:

1. **try** (контролировать)
2. **catch** (ловить)
3. **throw** (генерировать, порождать, бросать, посылать, формировать).

Добавляем в квадратное уравнение проверку

Куда добавить генерацию исключения?

```
if((b*b-4*a*c)<0) throw WrongArgumentException();
```

1. **Конструктор**
2. **Методы вычисления решения**

```
SquareEquation se(a, b, c);
try {
    std::cout << "X1=" << se.FindX1() << "\n";
    std::cout << "X2=" << se.FindX2() << "\n";
} catch (WrongEquationException ex) {
    std::cout << std::endl << ex.what() << std::endl;
}
```

Пример

CUSTOM_EXCEPTION.CPP

Exception

Исключение это:

1. Объект, наследник класса std::exception (#include <exception>)
2. Событие, прерывающее обработку программы.

Под прерыванием мы понимаем, что срабатывание исключение, аналогично срабатыванию оператора **return**.

Но есть существенное отличие: **return** возвращает результат в то место, где вызвали функцию.

Исключение возвращает объект исключения только в те места, где его явно ловят (**catch**)!

И только если исключение сработало (**throw**) в месте где мы его контролируем (**try**)!

Если исключение не поймать (**catch**) то оно будет прерывать работу функций, поднимаясь вверх по стеку вызова, пока не остановит программу.

Try / Catch

Служебное слово try позволяет выделить в любом месте исполняемого текста программы так называемый контролируемый блок:

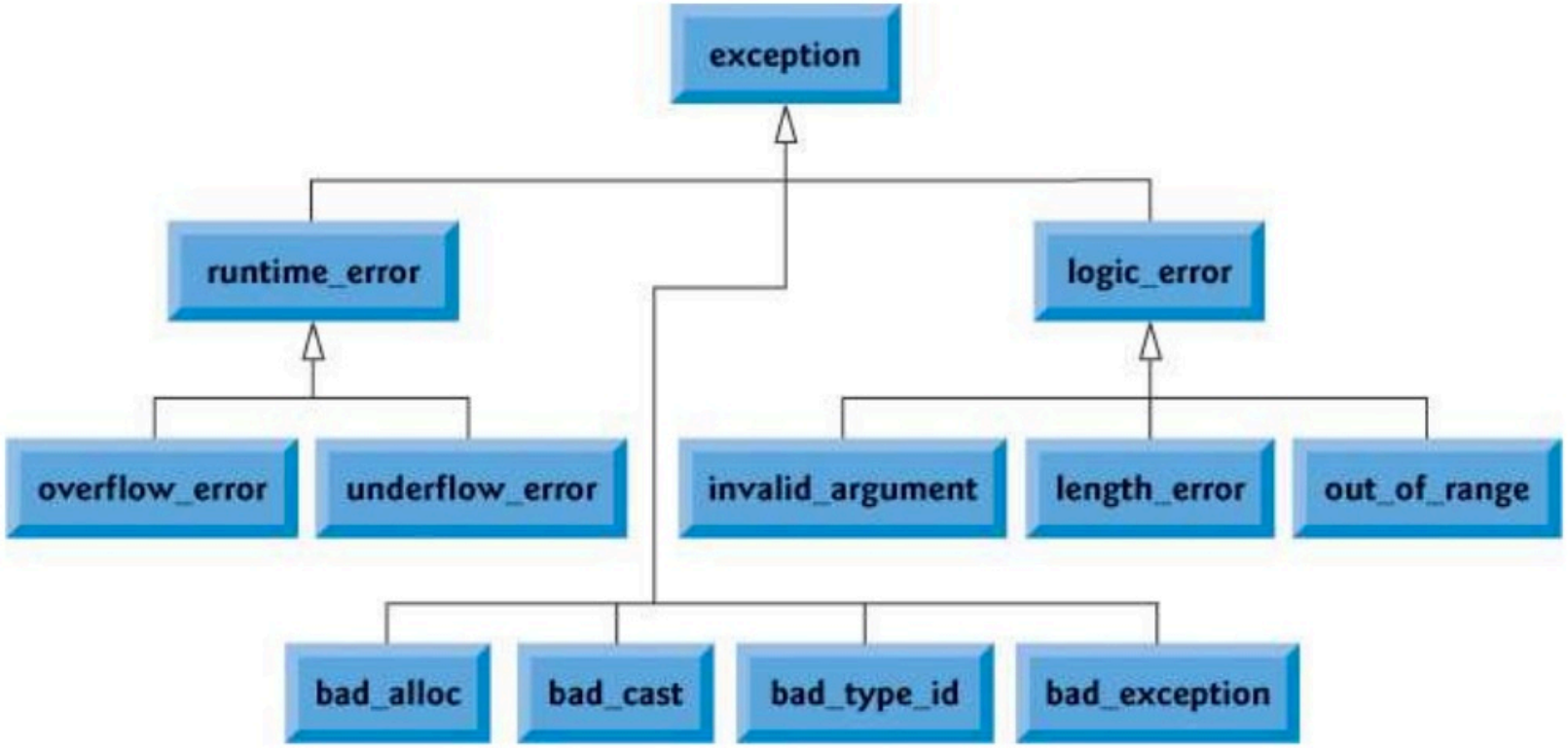
```
try {
//операторы
//операторы
} catch (Тип_исключения1 имя){
//операторы
} catch (Тип_исключения2 имя){
//операторы
}
```

Exception – это все что угодно

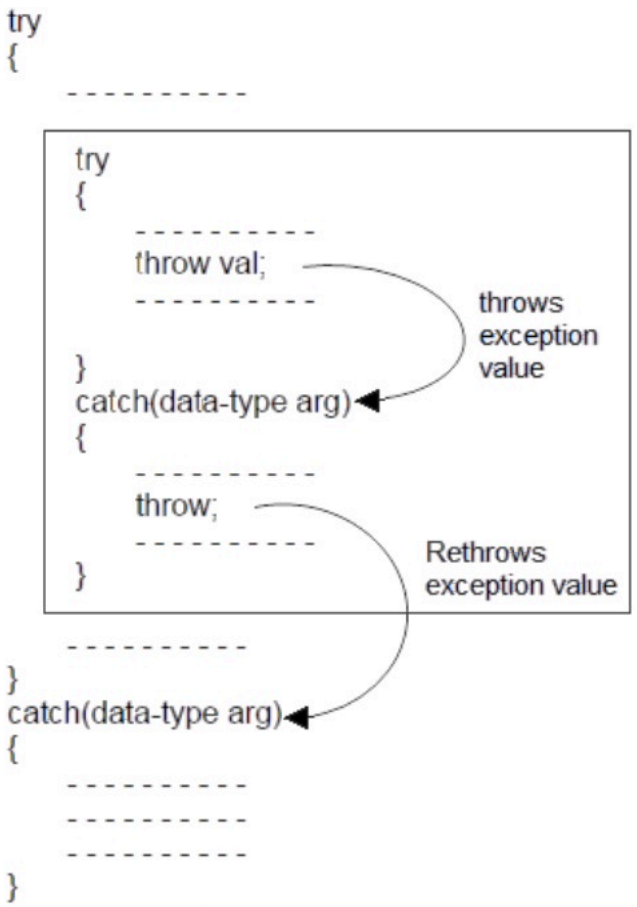
C++ позволяет создавать исключения любого типа, хотя обычно рекомендуется создавать типы, производные от **std::exception**. Исключение в C++ может быть перехвачено обработчиком **catch**, в котором определен тот же тип, что и у созданного исключения, или обработчиком, который способен перехватывать любой тип исключения.

Если созданное исключение имеет тип класса, у которого имеется один или несколько базовых классов, то его могут перехватывать обработчики, которые принимают базовые классы (и ссылки на базовые классы) этого типа исключения. Обратите внимание, что если исключение перехватывается по ссылке, то оно привязывается к самому объекту исключения; в противном случае обрабатывается его копия (как и в случае с аргументами функции).

Стандартные исключения C++ #include <exception>



Повторное «возбуждение» исключений



Exception Ptr

current_exception — данная функция возвращает exception_ptr. Если мы находимся внутри блока catch, то возвращает exception_ptr, который содержит обрабатываемое в данный момент текущим потоком исключение, если вызывать ее вне блока catch, то она вернет пустой объект exception_ptr

rethrow_exception — данная функция бросает исключение, которое содержится в exception_ptr. Если входной параметр не содержит исключения (пустой объект), то результат не определен.

make_exception_ptr — данная функция, может сконструировать exception_ptr без бросания исключения. Ее предназначение аналогично функции std::make_shared — конструирование объекта. Ее выполнение аналогично функции **throwExceptionAndCaptureExceptionPtr.** *В реализации от gcc-4.7.2 make_exception_ptr делает два копирования объекта some_exception.*

Пример

COMPLEX_EXCEPTION.CPP

Созданное исключение может перехватываться следующими типами обработчиков **catch**:

1. Обработчик, который может принимать любой тип данных (синтаксис с многоточием).
2. Обработчик, который принимает тот же тип, что и у объекта исключения; поскольку используется копия объекта, то модификаторы **const** и **volatile** игнорируются.
3. Обработчик, который принимает ссылку на тот же тип, что и у объекта исключения.
4. Обработчик, который принимает ссылку на форму **const** или **volatile** того же типа, что и у объекта исключения.
5. Обработчик, который принимает базовый класс того же типа, что и у объекта исключения; поскольку используется копия объекта, то модификаторы **const** и **volatile** игнорируются. Обработчик **catch** для базового класса не должен предшествовать обработчику **catch** для производного класса.
6. Обработчик, который принимает ссылку на базовый класс того же типа, что и у объекта исключения.
7. Обработчик, который принимает ссылку на форму **const** или **volatile** базового класса того же типа, что и у объекта исключения.
8. Обработчик, который принимает указатель, в который можно преобразовать созданный объект указателя при помощи стандартных правил преобразования указателей.

Очистка стека

В механизме исключений C++ элемент управления перемещается из оператора throw в первый оператор catch, который может обработать выданный тип. При достижении оператора catch все автоматические переменные, находящиеся в области между операторами throw и catch, удаляются в процессе, который называется очистка стека. При очистке стека выполнение продолжается следующим образом.

1. Элемент управления достигает оператора **try** в процессе нормального последовательного выполнения. Выполняется защищенный раздел в блоке **try**.

2. Если во время выполнения защищенного раздела исключение не создается, предложения **catch**, следующие за блоком **try**, не выполняются. Выполнение продолжается с оператора, расположенного после последнего предложения **catch**, следующего за связанным блоком **try**.

Очистка стека (продолжение)

3. Если исключение создается во время выполнения защищенного раздела или во время выполнения любой подпрограммы, прямо или косвенно вызываемой защищенным разделом, объект исключения создается из объекта, созданного операндом **throw**. (Это означает, что может быть задействован конструктор копии.) Обработчики **catch** проверяются в порядке их отображения после блока **try**. Если соответствующий обработчик не найден, проверяется следующий динамический внешний блок **try**. Этот процесс будет продолжаться до тех пор, пока не будет проверен последний внешний блок **try**.

4. Если соответствующий обработчик **catch** найден и он выполняет перехват по значению, его формальный параметр инициализируется путем копирования объекта исключения. Если обработчик выполняет перехват по ссылке, параметр инициализируется для ссылки на объект исключения. После инициализации формального параметра начинается процесс очистки стека. Выполняется обработчик **catch**, и выполнение программы продолжается после последнего обработчика, то есть с первого оператора или конструкции, которые не являются обработчиком **catch**.

Внимание!

Помни, что в блоке catch могут возникать свои исключения!

Если в блоке catch идет высвобождение ресурсов (удаление объектов, закрытие дескрипторов файлов) то их надо самих помещать в еще один вложенный блок try/catch.

Пример

EXCEPTION_IN_CATCH.CPP

Обработка всех исключений

```
1.try {
2.    throw CSomeOtherException();
3.}
4.catch(...) {
5.    // Catch all exceptions – dangerous!!!
6.    // Respond (perhaps only partially) to the exception, then
7.    // re-throw to pass the exception to some other handler
8.    // ...
9.    throw;
10.}
```

noexcept

В стандарте ISO C++11 был представлен оператор **noexcept**. Он поддерживается в Visual Studio 2015 и более поздних версий. Когда это возможно, используйте noexcept, чтобы задать возможность вызова функцией исключений.

В предыдущих версиях стандарта можно использовать **throw();**

Если метод с noexcept все таки сгенерирует исключение, то оно перехвачено уже не будет.

Пример

NOEXCEPT.CPP

Если исключение не обработали

Если для текущего исключения не удается найти подходящий обработчик (или обработчик **catch** для многоточия), то вызывается предопределенная функция времени выполнения terminate. (Функцию **terminate** также можно явным образом вызвать из любого обработчика.)

Действие по умолчанию для **terminate** заключается в том, что она вызывает функцию abort. Если вам необходимо, чтобы перед выходом из приложения функция **terminate** в вашей программе вызывала какую-то другую функцию, вызовите функцию **set_terminate**, указав в качестве ее единственного аргумента ту функцию, которую нужно вызвать.

Функцию **set_terminate** можно вызвать из любого места программы. Процедура **terminate** всегда вызывает последнюю функцию, заданную в качестве аргумента для **set_terminate**.

Пример

terminate.cpp

Exceptions итого

1. Помогает создать надежную программу;
2. Отделяет код обработки ошибок от основной логики программы;
3. Обработка исключений может быть реализована за пределами основного кода программы;
4. Существует возможность обрабатывать только выбранные типы исключений;
5. Программа, обрабатывающая исключения не остановится без объяснения причин (например, вывода на экран причины возникновения исключений);

Пример

auto.cpp

Пример

ENUM.CPP

Пример

FOR.CPP

Пример

NAMESPACES.CPP

Спасибо!

ВСЕ ИДЕМ НА ПЕРЕРЫВ