Объектно-ориентированное программирование

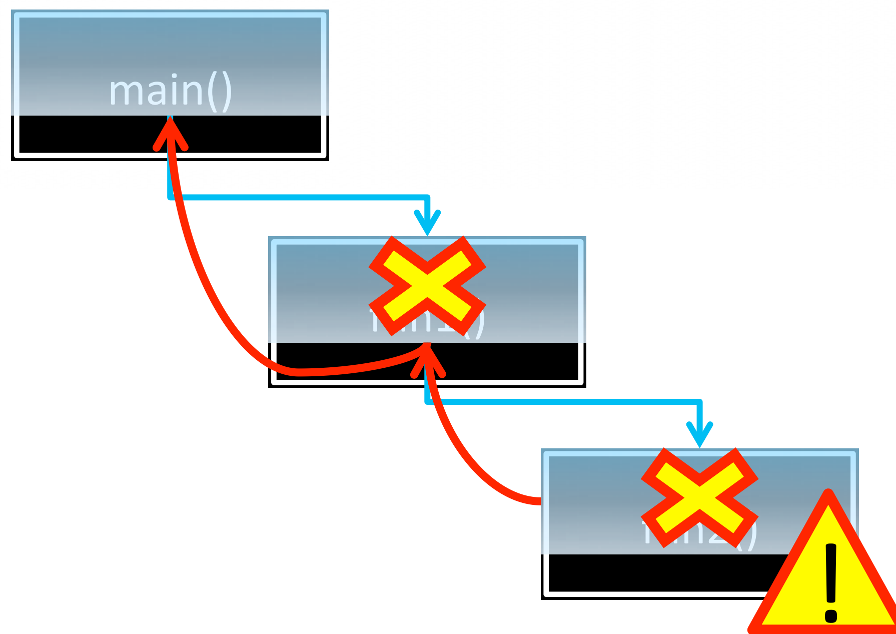
Раскрутка стека исключения. RAII.

Раскрутка стека исключения

Обработка возникающего исключения

- Когда выражение `throw` находится в `try`-блоке, все ассоциированные с ним предложения `catch` исследуются с точки зрения того, могут ли они обработать исключение.
- Если подходящее предложение `catch` найдено, то исключение обрабатывается. В противном случае поиск продолжается в вызывающей функции.
- Этот поиск последовательно проводится во всей цепочке вложенных вызовов.

Раскрутка стека



Раскрутка стека

- Процесс, при котором приложение, при возникновении исключительной ситуации, последовательно покидает составные блоки (вложенные блоки инструкций и вызовы функций) в поисках блока `catch`, способного обработать возникшее исключение, называется *раскруткой стека*.
- По мере раскрутки прекращают существование локальные объекты, объявленные в блоках, из которых произошел выход.
- **Вызываются деструкторы локальных объектов**.

Повторное возбуждение исключений

- Передать исключение другому `catch`-обработчику можно с помощью *повторного возбуждения исключения*.
- Для этой цели предусмотрена конструкция `throw` , которая может быть вызвана только `catch`-обработчика.
- Если объект-исключение передается по ссылке, то можно изменить состояние объекта-исключения и передать обновленный объект при инициализации `throw.`

TROW

```
void calculate( int op ) {
try {
      // исключение, возбужденное mathFunc(), имеет значение zeroOp
      mathFunc( op );
   }
   catch ( EHstate &eObj ) {
      // что-то исправить
      // модифицируем объект-исключение
      eObj = severeErr;
      throw;
   }
}
```

Обработка без catch

- Если обработчик исключения не находится, вызывается функция terminate() из стандартной библиотеки C++.
- По умолчанию terminate() активизирует функцию abort(), которая аномально завершает программу.
- Можно переопределить функцию terminate() если требуется отдать последние почести погибающему приложению.

RAII - Resource Acquisition Is Initialization

Resource Acquisition Is Initialization

- Получение ресурса есть инициализация (англ. Resource Acquisition Is Initialization (RAII)) — шаблон проектирования объектно-ориентированного программирования, смысл которого заключается в том, что получение некоторого ресурса совмещается с инициализацией, а освобождение — с уничтожением объекта.

Пример работы без RAII

- В Java значениями переменных являются не сами объекты, а ссылки на них.
- Поэтому для всех объектов память выделяется динамически, а сами объекты имеют неопределённое время жизни.
- В связи с этим, освободить ресурс в деструкторе невозможно, так как неизвестно, когда объект будет удалён (и будет ли удалён вообще). Поэтому в Java нет деструкторов, которые бы гарантированно вызывались.

Пример работы без RAII

```
try {
    File file = new File("/path/to/file");
    // Do stuff with file
} finally {
    file.close();
}
```

Пример работы без RAII

- Та же проблема в C++ решается по-другому: посредством закрытия файла в деструкторе объекта. Так как при раскручивании стека всегда вызываются деструкторы локальных объектов, файл закроется в любом случае.

```
try {
    File file = new File("/path/to/file");
    // Do stuff with file
}
//whatever
```

Исключения в конструкторах и деструкторах

Исключительные ситуации в конструкторах

- Конструкторы не могут возвращать коды ошибок, соответственно исключения – это единственный метод, чтобы понять, что в конструкторе что-то пошло не так.
- Но необходимо правильно обрабатывать исключительные ситуации в таких случаях.
- Особенно если в конструкторе формируются объекты в динамической памяти.

Пример исключительной ситуации в конструкторе

```
class Test {
Public:
   Test() {
        std::cout << "Test::Test()" << std::endl;
        // Здесь, в соответствии с RAII, захватили ресурсы
        if ( 1 ) { throw std::runtime_error( "AAAAAAAA" );
        } else {}
    }
    ~Test() {std::cout << "Test::~Test()" << std::endl;
        // А здесь мы освобождаем те самые важные ресурсы...}
};

int main() {
    Test* t = 0;
    try { t = new Test();    // Вроде бы создали...
    } catch ( const std::exception& exc ) {
        std::cout << exc.what() << std::endl;
    }
    delete t;     // Удалили? Ну-ну...
    return 0;
}
```

Исключительные ситуации в конструкторах

- Не стоит пытаться передавать исключения за тело конструктора, т.к. иначе, даже если мы не забываем вызвать `delete`, он не будет вызывать деструктор неправильно созданного объекта.
- В этом случае, никакие ресурсы, которые были инициализированы в конструкторе до вызова исключения (связи с БД, открытые файлы, и т.п.) уже никогда не будут освобождены (если они создавались в динамической памяти).

Решение проблемы?

- Вместо указателей на объекты в динамической памяти можно использовать один из множества типов «умных указателей», которые удаляются автоматически при выходе из области видимости (как локальные объекты)

```
class Cnt {
private:
  X *xa;
  X *xb;
public:
  Cnt(int a, int b) {
    cout << "Cnt::Cnt" << endl;
    xa = new X(a);
    xb = new X(b);
  }
  ~Cnt() {
    cout << "Cnt::~Cnt" << endl;
    delete xa;
    delete xb;
  }
};
```

```
class Cnt {
private:
  auto_ptr<X> ia;
  auto_ptr<X> ib;
public:
  Cnt(int a, int b) : ia(new X(a)),
ib(new X(b)) {
    cout << "Cnt::Cnt" << endl;
  }
  ~Cnt() {
    cout << "Cnt::~Cnt" << endl;
  }
};
```

Указатель AUTO_PTR

- std::auto_ptr<> - реализует семантику владения.
- Реализует так называемое *разрушающее копирование* - при присваивании, объект передается от одного указателя другому, удаляясь у первого, чтобы не удалять объект дважды при выходе из области видимости.

Исключительная ситуация в деструкторе

```
class test
{
public:
    test() { }
    ~test(){
        throw std::runtime_error("Game over!");
    }
};

int main() {
    try {
        test t;
                throw std::runtime_error("Error!");
    }
    catch(std::exception const&)
    { }
    return 0;
}
```

Исключительные ситуации в деструкторах

- Когда исключение покидает блок, все локальные объекты, созданные в этом блоке, уничтожаются.
- Если деструктор объекта, уничтожаемого во время развертки стека, генерирует исключение, то программа будет завершена досрочно, и ее уже ничего не спасет – вызывается функция terminate.

Что делать с ошибками в деструкторе?

- Запишите данные об ошибке в лог.
- Выведете сообщение на принтер, отправьте по SMS, позвоните и сообщите об этом бабушке…
- НО НИ В КОЕМ СЛУЧАЕ НЕ ИНИЦИАЛИЗИРУЙТЕ ИСКЛЮЧЕНИЕ
- Исключение при раскрутке стека всегда будет инициализировать функцию terminate().
- **Обрабатывайте все ошибки деструктора внутри деструктора.**

Итого

- Исключительные ситуации обрабатываются в блоке `catch`, который идет после `try`.
- После `try` может идти несколько блоков `catch`, каждый из которых настроен на перехват определенного типа исключений
- При возникновении исключения инициализируется раскрутка стека, заключающаяся в вызове деструкторов локальных объектов и передаче исключения вызывающей функции.
- Если обработчик не найден, вызывается функция terminate() которая вызывает функцию abort()
- Получение ресурса есть инициализация (англ. Resource Acquisition Is Initialization (RAII) – метод работы с ресурсами в C++
- С исключительными ситуациями в конструкторах надо разбираться очень тщательно, иначе можете потерять память или выделенные в конструкторе ресурсы (используйте умные указатели)
- НЕ ДАВАЙТЕ ИСКЛЮЧЕНИЯМ ВЫЙТИ ИЗ ДЕСТРУКТОРА