Обработка исключительных ситуаций

МГТУ им. Н.Э. Баумана

March 28, 2016

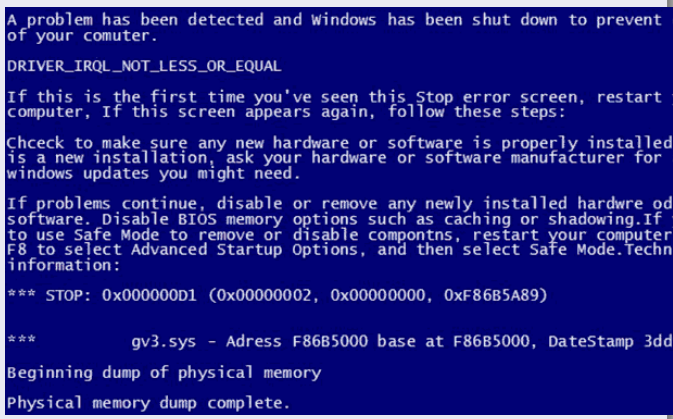
Обработка ошибок в C

Сообщение об ошибке из функции – стратегии

- Безусловное завершение работы
- Сообщение об ошибке через код возврата – возврат недопустимого значения
- Сообщение об ошибке через глобальную переменную и возврат допустимого значения
- Вызов функции, предназначенной для обработки ошибок

Безусловное завершение работы

```
1  class IntVector { /* ... */
2    size_t m_size;
3  public:
4    int operator[](size_t index) {
5      if (index < 0 || index >= m_size)
6        std::terminate();
7    }
8  };
9  int main() {
10   IntVector v = {1, 2, 3, 4};
11   if (v[5] == 2) // HERE MY PROGRAM
         TERMINATES!!!
12     std::cout << "v[5] == 2";
13   return 0;
14 }
```



Допустимо:

- нет других вариантов, дальнейшее функционирование не имеет смысла
- маленькие консольные приложения, которые проще перезапустить

Совпадает с ожиданиями пользователя?

Возврат недопустимого значения

```
1  int i; char c; double d;
2  int r;
3  const char* hello1 = "Welcome to my program";
4  const char* inputInt = "Please, input int";
5  // ...
6  r = printf("%s", hello1);
7  if(r != strlen(hello1)) exit(-1);
8  r = printf("%s", inputInt);
9  if(r != strlen(inputInt)) exit(-1);
10 r = scanf("%d", &i);
11 if(r <= 0) exit(-1);
12 r = printf("%s", inputChar);
13 if(r != strlen(inputChar)) exit(-1);
14 r = scanf("%c", &c);
15 if(r <= 0) exit(-1);
16 r = printf("%s", inputDouble);
17 if(r != strlen(inputDouble)) exit(-1);
18 r = scanf("%lf", &d);
19 if(r <= 0) exit(-1);
```

- Контроль возвращаемых ошибок часто игнорируется
- Многократное увеличение объема кода
- ЧТо, если все коды возврата корретные?

Глобальная переменная/функция обработки ошибок

```
1  class IntVector { /* ... */
2    size_t m_size;
3  public:
4    int operator[](size_t index) {
5      if (index < 0 || index >= m_size) {
6        errno = INDEX_OUT_OF_RANGE;
7        return 0;
8      }
9    }
10   int indexOf(int value) {
11     // ...
12     // index not found
13     SetLastError(INDEX_NOT_FOUND);
14   }
15 };
```

- Все проблемы, которые есть у метода возврата некорректного значения
- Что будет делать функция обработки ошибок?

Исключения

Механизм обработки ошибок, построенный на принципе разделения места обнаружения и места обработки ошибки.

- похож на метод с глобальной переменной
- отличается принудительностью обработки

```
1   class BadArgsException {};
2   int main(int argc, char* argv[]) {
3     // expecting ${0} hello world
4     try {
5       if(argc != 3)
6         throw BadArgsException();
7       if(std::string(argv[1]) != "hello")
8         throw BadArgsException();
9       if(std::string(argv[2]) != "world")
10        throw BadArgsException();
11    }
12    catch(const BadArgsException& e) {
13      std::cout << argv[0] << hello world expected" << std::endl;
14    }
15    return 0;
16  }
```

Исключения

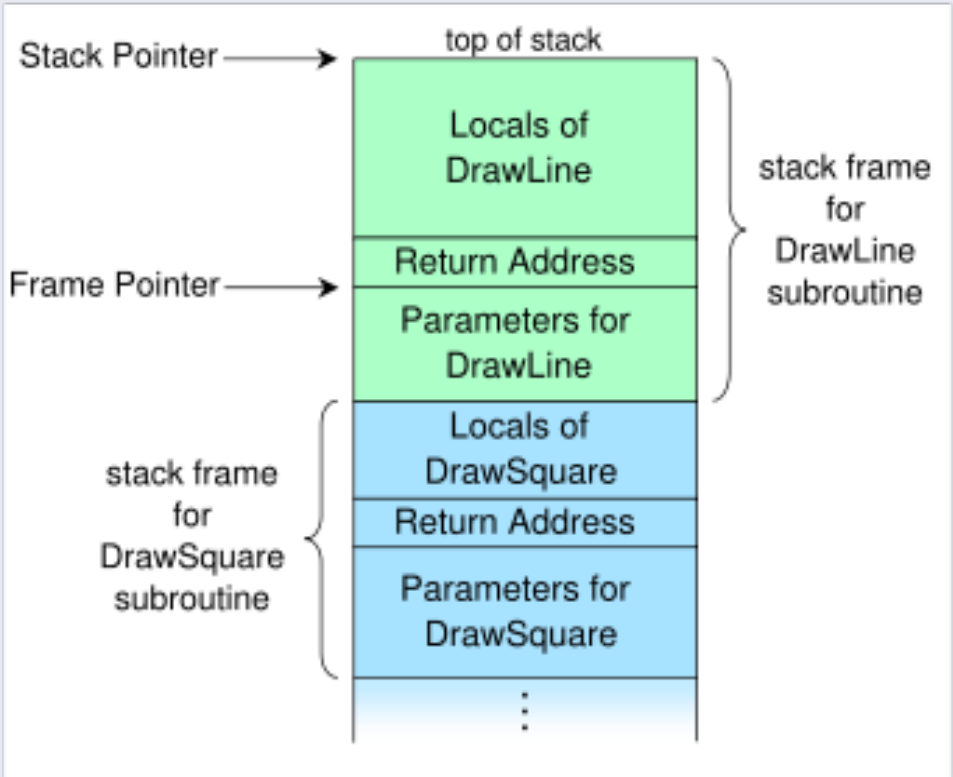
Механизм обработки ошибок, построенный на принципе разделения места обнаружения и места обработки ошибки.

- похож на метод с глобальной переменной
- отличается принудительностью обработки

```
1   class BadArgsException {};
2   int main(int argc, char* argv[]) {
3     // expecting ${0} hello world
4     try {
5       if(argc != 3)
6         throw BadArgsException();
7       if(std::string(argv[1]) != "hello")
8         throw BadArgsException();
9       if(std::string(argv[2]) != "world")
10        throw BadArgsException();
11    }
12    catch(const BadArgsException& e) {
13      std::cout << argv[0] << hello world expected" << std::endl;
14    }
15    return 0;
16  }
```

Стековый фрейм

Структуры в стеке процессора, которые содержат информацию о состоянии подпрограммы. Каждый стековый фрейм соответствует вызову какой-то подпрограммы, которая была вызвана, но ещё не завершилась.

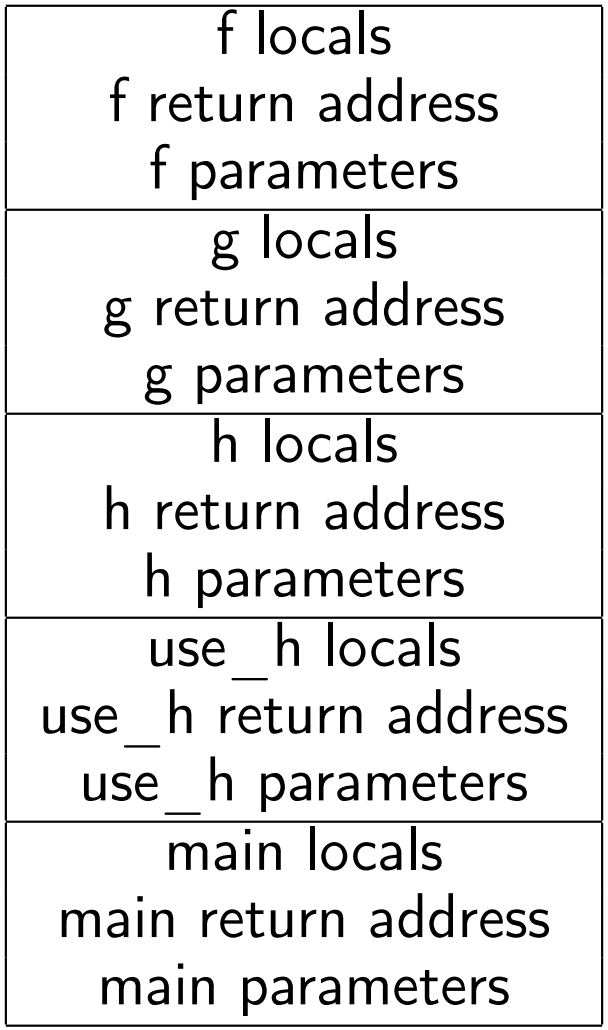


В x86-архитектуре стек растет от больших адресов к меньшим.

Раскрутка стека при исключении

```
1  void f() {
2    throw MyException("bad thing");
3    // ...
4  }
5  void g() {
6    return f();
7  }
8  void h() {
9    g();
10 }
11 void use_h() {
12   try {
13     h();
14   }
15   catch( const MyException& e) {
16     cout << e.what();
17   }
18 }
19 int main() {
20   // ...
21   use_h();
22   // ...
23   return 0;
24 }
```

Произошел throw в функции f()

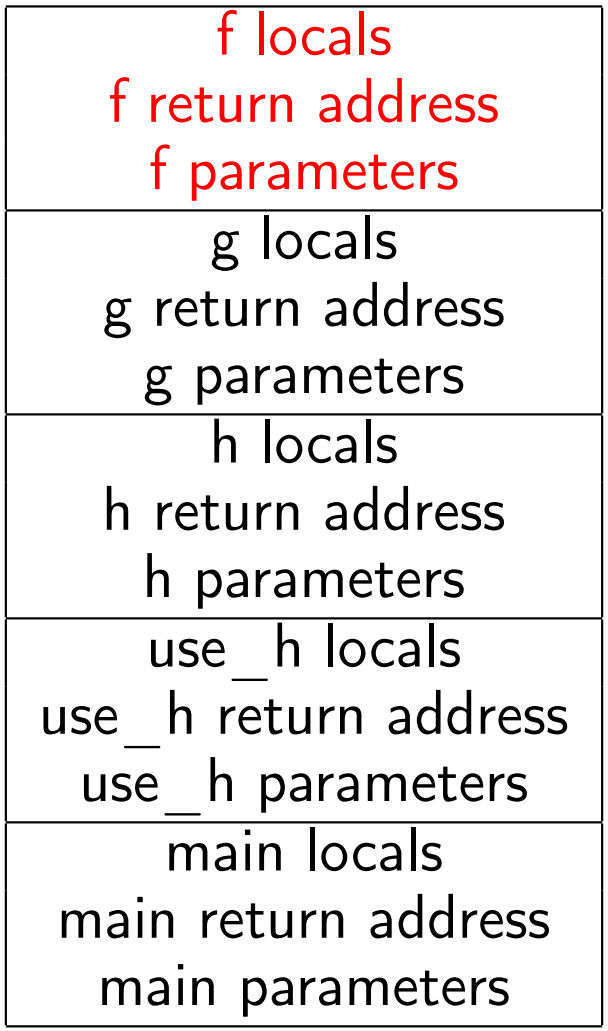


Раскрутка стека при исключении

```
1  void f() {
2    throw MyException("bad thing");
3    // ...
4  }
5  void g() {
6    return f();
7  }
8  void h() {
9    g();
10 }
11 void use_h() {
12   try {
13     h();
14   }
15   catch( const MyException& e) {
16     cout << e.what();
17   }
18 }
19 int main() {
20   // ...
21   use_h();
22   // ...
23   return 0;
24 }
```

try-catch не найден в функции f

f() завершается на строке throw

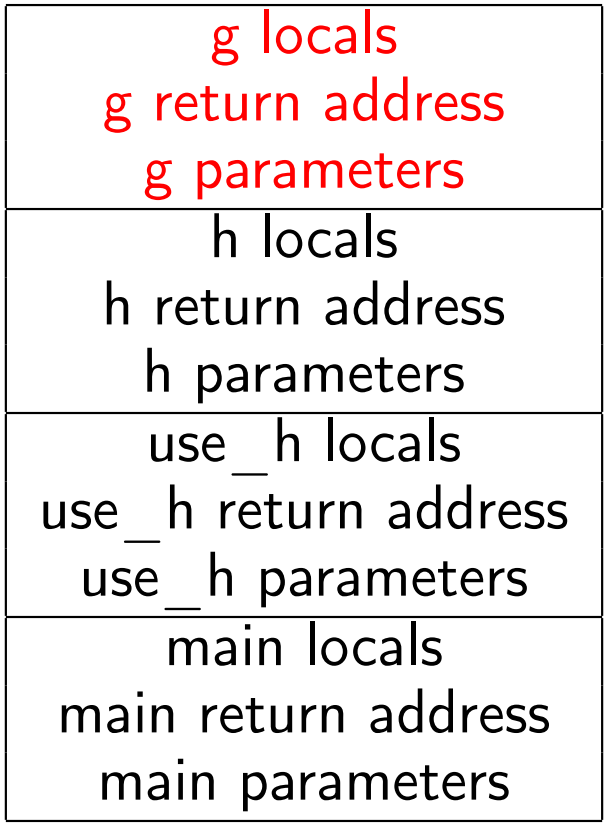


Раскрутка стека при исключении

```
1  void f() {
2    throw MyException("bad thing");
3    // ...
4  }
5  void g() {
6    return f();
7  }
8  void h() {
9    g();
10 }
11 void use_h() {
12   try {
13     h();
14   }
15   catch( const MyException& e) {
16     cout << e.what();
17   }
18 }
19 int main() {
20   // ...
21   use_h();
22   // ...
23   return 0;
24 }
```

try-catch не найден в функции g

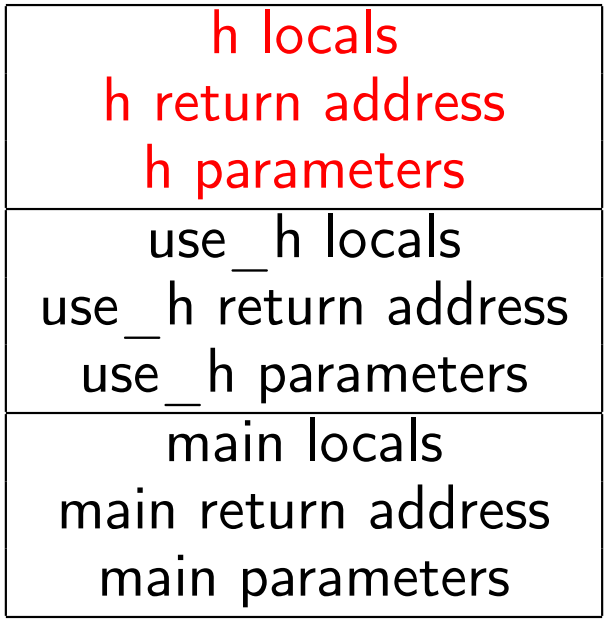
g() завершается на строке вызова f()



Раскрутка стека при исключении

```
1  void f() {
2    throw MyException("bad thing");
3    // ...
4  }
5  void g() {
6    return f();
7  }
8  void h() {
9    g();
10 }
11 void use_h() {
12   try {
13     h();
14   }
15   catch( const MyException& e) {
16     cout << e.what();
17   }
18 }
19 int main() {
20   // ...
21   use_h();
22   // ...
23   return 0;
24 }
```

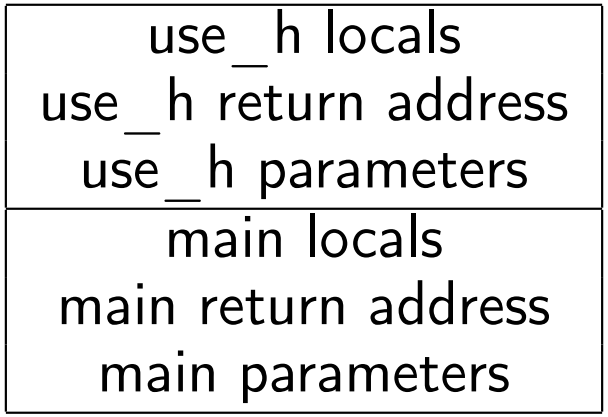
try-catch не найден в функции h
h() завершается на строке вызова g()



Раскрутка стека при исключении

```
1  void f() {
2    throw MyException("bad thing");
3    // ...
4  }
5  void g() {
6    return f();
7  }
8  void h() {
9    g();
10 }
11 void use_h() {
12   try {
13     h();
14   }
15   catch( const MyException& e) {
16     cout << e.what();
17   }
18 }
19 int main() {
20   // ...
21   use_h();
22   // ...
23   return 0;
24 }
```

try-catch окружает вызов h(), поэтому управление передается в блок catch use_h продолжает работу со строки, следующей за блоком catch



Синтаксис кода обработки исключения

```
try{
  // code in this block (or in functions called from here) may throw exception
}
catch (T e) {
  // process exception of type T
}
catch (T1& e) {
  // process exception of type T2
}
catch (const T3& e) {
  // process exception of type T3
}
catch (...) {
  // catch all exceptions

  // do cleanup and rethrow
  throw; // rethrow
}
```

Какие исключения ловит catch

```
1 try{
2   // something throw
3 }
4 catch (T e) {
5
6 }
```

- Исключение – объект типа T
- Исключение – публичный производный тип от T (срезка при копировании в e)
- Первые два пункта, если T указатель или ссылка (без срезки)

Какие исключения ловит catch

```
1  class BadThing {};
2  class BadThingWithVector : public BadThing {};
3  class BadIndex: public BadThingWithVector {};
4  class BadRange: public BadThingWithVector {};
5  try{
6    // something throw
7  }
8  catch (??? e) {
9
10 }catch (??? e) {
11
12 }catch (??? e) {
13
14 }catch (??? e) {
15
16 }
```

Какие исключения ловит catch

```
1  class BadThing {};
2  class BadThingWithVector : public BadThing {};
3  class BadIndex: public BadThingWithVector {};
4  class BadRange: public BadThingWithVector {};
5  try{
6    // something throw
7  }
8  catch (BadIndex e) {
9
10 }catch (BadRange e) {
11
12 }catch (BadThingWithVector e) {
13
14 }catch (BadThing e) {
15
16 }
```

try для тела функции

```
int f(uint32_t i, uint32_t n)
try {
  // ...
  if(i >= n)
    throw std::runtime_error("Bad index");
  return 0;
}
catch(const std::exception& e) {
  return -1;
}
```

Список инициализаторов

```
class A {
  int m_m, m_n;
public:
  A(int m, int n)
  try
    : m_m(m), m_n(n)
  {
    //todo
  } catch (...) {
    //TODO
  }
}
```

RAII

```
1 int doSmth(string filename, int i) {
2   FILE* f = fopen(filename.c_str(), "r");
3   // ...
4   if (i < 0)
5     throw runtime_error("Bad index");
6   // ...
7   fclose(f);
8 }
```

RAII

```
int doSmth(string filename, int i) {
  FILE* f = fopen(filename.c_str(), "r");
  // ...
  if (i < 0)
    throw runtime_error("Bad index");
  // ...
  fclose(f);
}
```

```
class FilePtr {
private: FILE* mFile;
public;
  FilePtr(string filename) {
    mFile = fopen(filename.c_str(), "r");
  }
  ~FilePtr() {
    fclose(mFile);
  }
}
int doSmth(string filename, int i) {
  FilePtr f(filename.c_str(), "r");
  if (i < 0)
    throw runtime_error("Bad index");
  // ...
}
```

Спецификация исключений throw() (deprecated)

```
int doSmth(string filename, int i) throw (BadThing, runtime_error) {
  // ...
  if (i < 0)
    throw runtime_error("Bad index");
  // ...
}
```

- Указание, какие исключения могут быть выброшены из функции
- Контроль на этапе выполнения
- Вызов std::unexpected(), если бросается исключение другого типа

```
int doSmthElse() throw () // does not throw exception
{
  // ...
  return 0;
}
```

Спецификация исключений noexcept (C++11)

noexcept-функция

```
1  int doSmth(string filename, int i) noexcept {
2    // ...
3    return 0;
4  }
```

- Указание, что функция не бросает исключение
- Контроль на этапе выполнения
- Вызов std::terminate(), если бросается исключение

noexcept-оператор

```
1  int doSmth(string filename, int i) noexcept(noexcept(openfile(filename)))
2    // ...
3    return 0;
4  }
5  constexpr setNoexcept = true;
6  int doSmth(string filename, int i) noexcept(setNoexcept)
7    // ...
8    return 0;
9  }
10 int doSElse(string f, int i) noexcept(noexcept(doSmth(f, i)))
11   // ...
12   return 0;
13 }
```

Завершение работы программы

Принципы

- Не бросать исключение, пока не обработано предыдущее
- Не бросать исключение, которое не будет поймано

Будет вызван terminate()

Деструкторы объектов не будут вызваны.

- Не найден обработчик для исключения
- noexcept-функция выбрасывает исключение
- деструктор объекта при раскрутке стека по исключению выходит по исключению
- код по распространению исключения (конструктор копирования, например) бросает исключение
- выполнен throw; (перегенерация исключения), хотя исключение не обрабатывается
- Исключение выброшено конструктором/деструктором статического объекта
- обработчик atexit() бросает исключение

Гарантии исключений

Базовая

Отсутствие утечки ресурса при завершении функции с исключением.

Строгая

Операция, реализующая данную гарантию, либо выполняется полностью, либо не выполняется вообще.

nothrow-гарантия

Базовая гарантия + функция никогда не будет бросать исключение.

Finally

Use dtors!