OTUS

Онлайн образование

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Правила вебинара

Активно
участвуем

Off-topic обсуждаем
в учебной группе в
Telegram

Задаем
вопрос в чат

Вопросы вижу в чате,
могу ответить не сразу

Тема вебинара

Разработчик C++ - базовый курс

Исключения в C++

Использование и нюансы
Гарантии безопасности

Пальчуковский Евгений

C++ разработчик

Email: eugene@palchukovsky.com
Telegram: @palchukovsky
Slack: @Eugene Palchukovsky

Цели вебинара

1. Обрабатывать ошибки в программах на C++
2. Обрабатывать ошибки разными способами
3. Использовать исключения
4. Обеспечивать уровни гарантий безопасности исключений
5. Понимать и применять идиому “copy and swap”

Смысл

Зачем вам это уметь

1. Обработка ошибок - важная часть программирования
2. Писать идиоматический код

Ошибки в программах

Типы ошибок в программировании

1. Вспомните, какие типы ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок в программировании

1. Вспомните, какие **типы** ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок

- ошибки компиляции

Типы ошибок в программировании

1. Вспомните, какие **типы** ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок

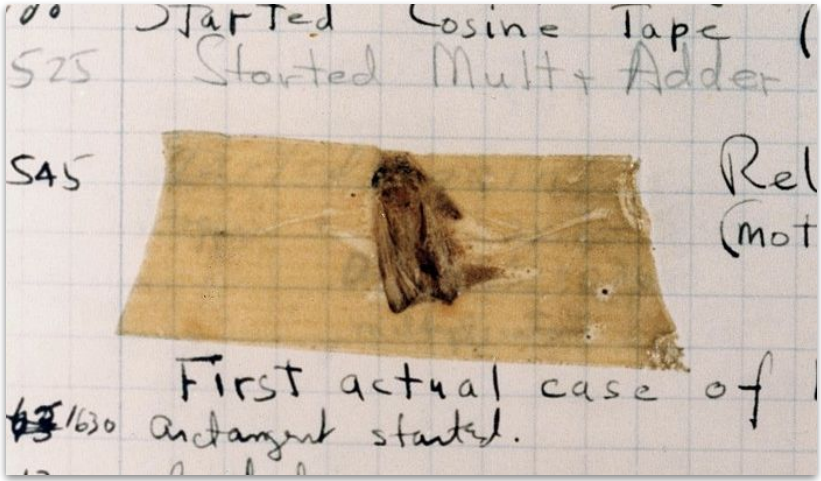
- ошибки компиляции
- ошибки компоновки

Типы ошибок в программировании

1. Вспомните, какие **типы** ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок

- ошибки компиляции
- ошибки компоновки
- логические ошибки



Типы ошибок в программировании

1. Вспомните, какие **типы** ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок

- ошибки компиляции
- ошибки компоновки
- логические ошибки

- некорректные входные данные
- ошибки выделения ресурсов

Типы ошибок в программировании

1. Вспомните, какие **типы** ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок

- ошибки компиляции
- ошибки компоновки
- логические ошибки

- некорректные входные данные
- ошибки выделения ресурсов

- ошибки в сторонних библиотеках

Типы ошибок в программировании

1. Вспомните, какие **типы** ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок

- ошибки компиляции
- ошибки компоновки
- логические ошибки

- некорректные входные данные
- ошибки выделения ресурсов

- ошибки в сторонних библиотеках
- ошибки в компиляторах

Типы ошибок в программировании

1. Вспомните, какие **типы** ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок

- ошибки компиляции
- ошибки компоновки
- логические ошибки

- некорректные входные данные
- ошибки выделения ресурсов

- ошибки в сторонних библиотеках
- ошибки в компиляторах
- ошибки в процессорах

Типы ошибок в программировании

1. Вспомните, какие **типы** ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок

- ошибки компиляции
- ошибки компоновки
- логические ошибки

- некорректные входные данные
- ошибки выделения ресурсов

- ошибки в сторонних библиотеках
- ошибки в компиляторах
- ошибки в процессорах

Что делать?

Типы ошибок в программировании

1. Вспомните, какие **типы** ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок

- ошибки компиляции
- ошибки компоновки
- логические ошибки

- некорректные входные данные
- ошибки выделения ресурсов

- ошибки в сторонних библиотеках
- ошибки в компиляторах
- ошибки в процессорах

Что делать?

Хорошо делай - хорошо будет :)
В том числе тестирование!

Типы ошибок в программировании

1. Вспомните, какие **типы** ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок

- ошибки компиляции
- ошибки компоновки
- логические ошибки

- некорректные входные данные
- ошибки выделения ресурсов

- ошибки в сторонних библиотеках
- ошибки в компиляторах
- ошибки в процессорах

Что делать?

Хорошо делай - хорошо будет :)
В том числе тестирование!

Пиши код для обработки ошибок
Наша тема!

Типы ошибок в программировании

1. Вспомните, какие **типы** ошибок вы знаете
2. Напишите ваши варианты в чате

Типы ошибок	Что делать?
● ошибки компиляции ● ошибки компоновки ● логические ошибки	Хорошо делай - хорошо будет :) В том числе тестирование!
● некорректные входные данные ● ошибки выделения ресурсов	Пиши код для обработки ошибок Наша тема!
● ошибки в сторонних библиотеках ● ошибки в компиляторах ● ошибки в процессорах	Старайся использовать свежие Но по делу!

Пример. Наибольший общий делитель

Нам выставили ТЗ:

- написать программу для вычисления НОД
- для вычисления использовать Алгоритм Евклида
- входные данные через аргументы командной строки
- на выходе - значение НОД

Мы написали программу - **gcd.cpp**

Какие ошибки могут возникнуть при ее работе?

Обработка ошибок

Специальные коды возврата

- добавим в наш код обработку ошибок
- будем писать сообщения об ошибках
- функции будут возвращать специальное значение при ошибке
- программа будет завершаться с кодом неуспеха при ошибке

Пример: gcd_err_handling.cpp

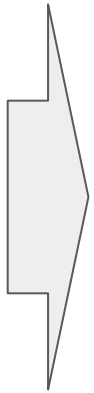
Было/стало

```
unsigned gcd(unsigned a, unsigned b);

int main(int, char *argv[]) {
   unsigned a = std::atoi(argv[1]);
   unsigned b = std::atoi(argv[2]);

   std::cout << gcd(a, b) << "\n";
   return EXIT_SUCCESS;
}

unsigned int gcd(unsigned int a, unsigned int b) {
   do {
       unsigned r = a % b;
       a = b;
       b = r;
   } while (b);
   return a;
}
```



```
unsigned gcd(unsigned a, unsigned b);
unsigned parse_unsigned(char *str);

int main(int argc, char *argv[]) {
   if (argc < 3) {
       std::cerr << "Usage: gcd a b\n";
       return EXIT_FAILURE;
   }
   unsigned a = parse_unsigned(argv[1]);
   if (!a) {
       return EXIT_FAILURE;
   }
   unsigned b = parse_unsigned(argv[2]);
   if (!b) {
       return EXIT_FAILURE;
   }
   unsigned result = gcd(a, b);
   if (!result) {
       return EXIT_FAILURE;
   }

   std::cout << result << "\n";
   return EXIT_SUCCESS;
}
```

```
unsigned gcd(unsigned a, unsigned b) {
   if (!b) {
       std::cerr << "GCD error: divison by zero\n";
       return 0;
   }
   do {
       unsigned r = a % b;
       a = b;
       b = r;
   } while (b);
   return a;
}

unsigned parse_unsigned(char *str) {
   std::string_view sw{str};
   if (!std::all_of(sw.begin(), sw.end(), ::isdigit)) {
       std::cerr << "Number parse error: all chars must be digits\n";
       return 0;
   }

   long val = std::atol(str);
   if (val < 0) {
       std::cerr << "Number parse error: input must be positive integer\n";
       return 0;
   }
   if (val > std::numeric_limits<unsigned>::max()) {
       std::cerr << "Number parse error: input number is too large\n";
       return 0;
   }
   return static_cast<unsigned>(val);
}
```

Отдельные коды ошибок

Проблемы со **специальными кодами** возврата:

- не всегда получается зарезервировать
- как возвращать разные типы ошибок

Отдельные коды ошибок

Проблемы со **специальными кодами** возврата:

- не всегда получается зарезервировать
- как возвращать разные типы ошибок

Способы решения:

1. Использовать out параметры (пример: gcd_err_out.cpp)

```
unsigned int gcd(unsigned int a, unsigned int b, int &err)
```

Отдельные коды ошибок

Проблемы со **специальными кодами** возврата:

- не всегда получается зарезервировать
- как возвращать разные типы ошибок

Способы решения:

1. Использовать out параметры (пример: gcd_err_out.cpp)

```
unsigned int gcd(unsigned int a, unsigned int b, int &err)
```

2. Несколько возвращаемых значений (как делает std::map::insert)

```
std::pair<unsigned int, int> gcd(unsigned int a, unsigned int b);
```

Отдельные коды ошибок

Проблемы со **специальными кодами** возврата:

- не всегда получается зарезервировать
- как возвращать разные типы ошибок

Способы решения:

1. Использовать out параметры (пример: gcd_err_out.cpp)

```
unsigned int gcd(unsigned int a, unsigned int b, int &err)
```

2. Несколько возвращаемых значений (как делает std::map::insert)

```
std::pair<unsigned int, int> gcd(unsigned int a, unsigned int b);
```

3. Специальные обертки для возвращаемых значений

```
std::optional<unsigned int> gcd(unsigned int a, unsigned int b);
```

#include <system_error>

Расширяемые инструменты для работы с кодами ошибок:

- **std::error_code** - для зависящих от платформы кодов ошибок
- **std::error_condition** - для не зависящих от платформы кодов ошибок
- **std::error_category** - базовый класс для работы с кодами ошибок
  - виртуальный метод **message()** для конвертации кодов ошибок в строки
- **error_code** и **error_condition** содержат ссылку на наследника error_category

#include <system_error>

Расширяемые инструменты для работы с кодами ошибок:

- **std::error_code** - для зависящих от платформы кодов ошибок
- **std::error_condition** - для не зависящих от платформы кодов ошибок
- **std::error_category** - базовый класс для работы с кодами ошибок
  - виртуальный метод **message()** для конвертации кодов ошибок в строки
- **error_code** и **error_condition** содержат ссылку на наследника error_category

Пример: gcd_err_code.cpp

- перечислим коды ошибок с помощью **enum class**
- создадим наследника для **error_category**
- напишем перегрузку для функции **make_error_condition()**

Задание

Функция gcd обрабатывает ошибки по-старому

1. Добавьте код ошибки для деления на нуль
2. Поменяйте тип ошибки с int на error_condition в gcd()
3. Добавьте сообщение об этой ошибке в message()
4. Поменяйте код main для работы с error_condition в gcd()
5. Соберите и запустите программу
6. Убедитесь, что программа корректно обрабатывает деление на нуль

Сроки выполнения: 5 минут

Промежуточные тезисы

- обработка ошибок - важная обязательная практика разработки ПО

Промежуточные тезисы

- обработка ошибок - важная обязательная практика разработки ПО
- код обработки ошибок может **значительно усложнять чтение логики**

Промежуточные тезисы

- обработка ошибок - важная обязательная практика разработки ПО
- код обработки ошибок может **значительно усложнять чтение логики**
- обработка ошибок через коды возврата (return codes) **не всегда удобна**

Промежуточные тезисы

- обработка ошибок - важная обязательная практика разработки ПО
- код обработки ошибок может **значительно усложнять чтение логики**
- обработка ошибок через коды возврата (return codes) **не всегда удобна**
- использование отдельных кодов ошибок **усложняет сигнатуру функций**

Промежуточные тезисы

- обработка ошибок - важная обязательная практика разработки ПО
- код обработки ошибок может **значительно усложнять чтение логики**
- обработка ошибок через коды возврата (return codes) **не всегда удобна**
- использование отдельных кодов ошибок **усложняет сигнатуру функций**
- использование кодов ошибок заставляет **проверять код возврата всегда**

Исключения

Исключения

Механизм передачи управления внутри программы, который используется при возникновении ошибки (исключительной ситуации).

```
try {
    //...
    throw std::make_error_condition();
} catch(const std::error_condition &err) {
    std::cerr << err.message() << std::endl;
    return EXIT_FAILURE;
}
```

Исключения

- можно передавать объект любого типа
- обработчик автоматически выбирается по типу
- … в том числе с помощью полиморфизма
- … в том числе можно поймать “любой тип”: “catch (...)”
- необязательно “ловить”
- … но проигнорировать невозможно: приложение будет прервано
- можно поймать, обработать, и передать дальше: “throw;”

Раскрутка стека

Алгоритм раскрутки стека (stack unwinding) при возникновении исключения:

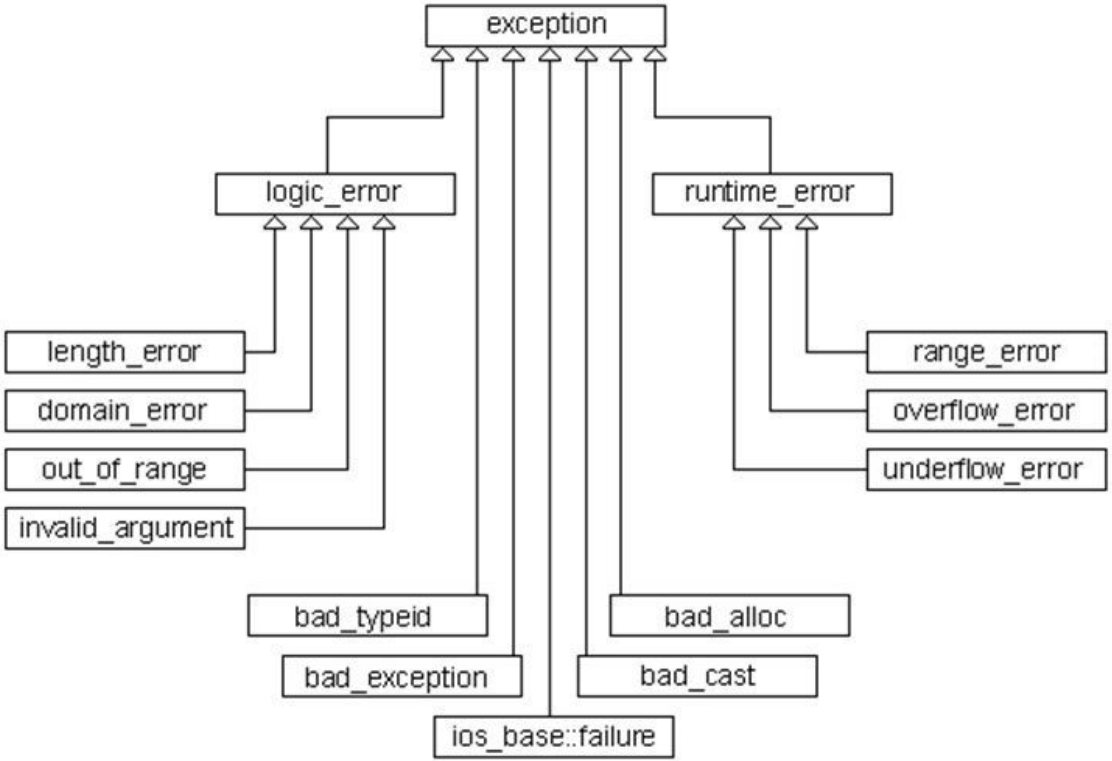
- ищется ближайший подходящий catch block вверх по стеку вызовов
- если найден, то для локальных объектов деструкторы
  - RAII позволяет корректно управлять ресурсами даже в исключительных ситуациях!
- иначе вызывается **std::terminate()**

Если при раскрутке стека возникнет еще одно исключение, то будет вызван std::terminate()

По этой причине выброс исключения за пределы деструктора может привести **к аварийному завершению программы**

По этой причине все деструкторы в C++ по умолчанию **noexcept**

Иерархия исключений std



Гарантии безопасности исключений

Уровни гарантий

- **нет гарантий** - всё может быть очень-очень плохо
- **базовые гарантии** - ничего критичного, но возможно странное
- **строгие гарантии** - будто бы и никакого исключения не было
- **гарантия отсутствия исключений** - никогда не будет исключения

Задание

1. Каким гарантиям удовлетворяет ex_safery_no_garanty.cpp?

2. Добавьте перехват исключения std::bad_alloc

3. Обеспечьте базовые гарантии исключений

Сроки выполнения: 5 минут

Идиома copy and swap

Copy and swap

Популярная идиома языка C++ для обеспечения строгих гарантий безопасности исключений

```
Person &operator=(const Person &other) {
    // Copy and swap idiom
    Person tmp{other};
    swap(tmp);
    return *this;
}
```

Задание

1. Откройте файл copy_and_swap.cpp?
2. Допишите реализацию метода Person::swap()
3. Раскомментируйте вызов swap
4. Скомпилируйте файл без ошибок

Сроки выполнения: 5 минут

Заключение

Ключевые тезисы

1. Обработка ошибок - это сложно
2. Обработка ошибок - это важно
3. Коды ошибок - это удобно
4. Исключения - это способ передачи управления
5. В своем коде нужно следить за безопасностью исключений

Цели вебинара

1. Обрабатывать ошибки в программах на C++
2. Обрабатывать ошибки разными способами
3. Использовать исключения
4. Обеспечивать уровни гарантий безопасности исключений
5. Понимать и применять идиому “copy and swap”

Следующий вебинар

30 июня 2023

Семантика перемещения, noexcept, и как они нам помогают

- Ссылка на вебинар будет в ЛК за 15 минут
- Материалы к занятию в ЛК — можно изучать

Вопросы?

Ставим “+”,
если вопросы есть

Ставим “–”,
если вопросов нет

Заполните, пожалуйста, опрос о занятии по ссылке в чате