Иерархия исключений C++

Вопросы

Хорошая ли идея бросать `int`?

Хорошая ли идея ловить "что угодно"?

Исключения-классы

В C++ есть договоренность - бросать исключения классовых типов (тип дает информацию о природе исключений)

```
class DivisionByZero {
  std::string info_;

 public:
  explicit DivisionByZero(const char* info) noexcept : info_(info) {
  }
  const char* What() const noexcept {
    return info_.c_str();
  }
};

template <class T>
T Divide(T x, T y) {
  if (y == 0) { throw DivisionByZero("some info..."); }
  return x / y;
}
```

Исключения-классы

```
class DivisionByZero {
  std::string info_;

 public:
  explicit DivisionByZero(const char* info) noexcept : info_(info) {
  }
  const char* What() const noexcept {
        return info_.c_str();
  }
};

int main() {
  try {
    Divide(1, 0);
  } catch (const DivisionByZero& error) {
    std::cerr << error.What() << '\n';
  } catch (...) {
    // а что произошло здесь?
  }
}
```

Исключения-классы

Давайте сделаем следующее: заведем общий базовый класс `Exception`, а все остальные исключения будем наследовать от него.

```
class Exception {
 public:
  virtual const char* What() const noexcept { return "Exception"; }
  virtual ~Exception() = default;
};
class DivisionByZero : public Exception {
  // ...
  const char* What() const noexcept override { return "DivisionByZero"; }
};

int main() {
  try {
    Divide(1, 0);
  } catch (const Exception& ex) {
    std::cerr << ex.What() << '\n';  // DivisionByZero
  }
}
```

Стандартная библиотека исключений C++

В C++ есть готовый базовый класс исключений - `std::exception`.

Он содержит единственный виртуальный метод - `what()`.

Все стандартные классы исключений унаследованы от него:

`std::logic_error`, `std::runtime_error`, `std::bad_cast` (бросает `dynamic_cast`), `std::bad_alloc` (бросает `new` при неудаче), `std::bad_weak_ptr` и много других

От них, в свою очередь, могут быть унаследованы (уточнены) другие исключения. Например, `std::out_of_range` унаследован от `std::logic_error`, а `std::bad_any_cast` от `std:bad_cast`

Стандартная библиотека исключений C++

Таким образом, можем группировать исключения по степени общности и по смыслу:

```
int main() {
  try {
    f();  // потенциально бросает исключения
  }
  catch (std::out_of_range& oor) { /* обрабатываем выход за границы */ }
  catch (std::logic_error& le) { /* обрабатываем logic_error'ы */ }
  catch (std::runtime_error& re) { /* обрабатываем runtime_error'ы */ }
  catch (std::exception& ex) { /* обрабатываем все остальное */ }
}
```

Для тех, кто не спит: для чего принимаем по ссылке?

Стандартная библиотека исключений C++

Свои классы исключений также стоит наследовать от одного из стандартных классов ошибок

```
class DivisionByZero : public std::runtime_error {
public:
    using std::runtime_error::runtime_error;
};
```

Бонус 1: function try блок

Неприятность

Что, если хочется навесить блок `try` на все тело функции?

```
void f() {
  try {
    // ...
  } catch (std::exception& exception) {
    // ...
  }
}
```

Выглядит не очень

Проблема

```
class B {
  int* ptr_;
  A a_;
 public:
  B() : ptr_(new int(11)), a_(0) {  // <--!
  }
  // ...
};
```

Что, если при конструировании `a_` вылетит исключение?

Решение: function try block

Блок `try-catch` можно навесить на функцию целиком (вместе со списком инициализации)

```
void f() try {
  // ...
} catch (std::exception& exception) { /* ... */ }
```

```
class B {
  int* ptr_;
  A a_;
 public:
  B() try : ptr_(new int(11)), a_(0) {  // <--!
    // ...
  } catch (...) {
    delete ptr_; throw;
  }
  // ...
};
```


Упражнение: найти баг в предложенном решении

Бонус 2: Метаинформация о функции

Макросы `__func__`, `__LINE__`, `__FILE__`:
https://en.cppreference.com/w/c/language/function_definition

`std::source_location` (C++20):
https://en.cppreference.com/w/cpp/utility/source_location