Junior C++ developer

Лекция 9

- Исключения
  - std::exception
  - std::runtime_error

Что будет выведено на экран?

```
class Base
{
public:
    Base(){}
};

class Derived: public Base{
public:
    Derived(){}
};
```

```
int main()
{
    try
    {
        throw Derived();
    }
    catch (Derived &child)
    {
        std::cerr << "caught Derived";
    }
    catch (Base &parent)
    {
        std::cerr << "caught Base";
    }
}
```

Что будет выведено на экран? Ответ

```
class Base
{
public:
    Base(){}
};

class Derived: public Base{
public:
    Derived(){}
};
```

```
int main()
{
    try
    {
        throw Derived();
    }
    catch (Derived &child)
    {
        std::cerr << "caught Derived";
    }
    catch (Base &parent)
    {
        std::cerr << "caught Base";
    }
}
```

caught Derived

Что будет выведено на экран?

```
class Base
{
public:
    Base(){}
};

class Derived: public Base{
public:
    Derived(){}
};
```

```
int main()
{
    try
    {
        throw Derived();
    }
    catch (Base &parent)
    {
        std::cerr << "caught Base";
    }
    catch (Derived &child)
    {
        std::cerr << "caught Derived";
    }
}
```

Что будет выведено на экран? Ответ

```
class Base
{
public:
    Base(){}
};

class Derived: public Base{
public:
    Derived(){}
};
```

```
int main()
{
    try
    {
        throw Derived();
    }
    catch (Base &parent)
    {
        std::cerr << "caught Base";
    }
    catch (Derived &child)
    {
        std::cerr << "caught Derived";
    }
}
```

caught Base

Что будет выведено на экран?

```
class Base
{
public:
    Base(){}
};

class Derived: public Base{
public:
    Derived(){}
};
```

```
int main()
{
    try
    {
        throw Base();
    }
    catch (Derived &child)
    {
        std::cerr << "caught Derived";
    }
    catch (Base &parent)
    {
        std::cerr << "caught Base";
    }
}
```

Что будет выведено на экран? Ответ

```
class Base
{
public:
    Base(){}
};

class Derived: public Base{
public:
    Derived(){}
};
```

```
int main()
{
    try
    {
        throw Base();
    }
    catch (Derived &child)
    {
        std::cerr << "caught Derived";
    }
    catch (Base &parent)
    {
        std::cerr << "caught Base";
    }
}
```

caught Base

Правило

- Обработчики исключений *дочерних* классов должны находиться *перед* обработчиками исключений *родительского* класса.

std::exception

std::exception - класс, который является родительским для любого исключения, которое выбрасывается в стандартной библиотеке C++.

Т.е. чтобы поймать любое исключение, выброшенное стандартной библиотекой достаточно ловить и обрабатывать ***std::exception.***

```
#include <exception>
```

Исключения наследники **std::exception**

- logic_error
  - invalid_argument
  - domain_error
  - length_error
  - out_of_range
  - future_error(C++11)
- bad_optional_access(C++17)
- runtime_error
  - range_error
  - overflow_error
  - underflow_error
  - regex_error(C++11)
  - nonexistent_local_time(C++20)
  - ambiguous_local_time(C++20)
  - tx_exception(TM TS)
  - system_error(C++11)
    - ios_base::failure(C++11)
    - filesystem::filesystem_error(C++17)
- bad_typeid
- bad_cast
  - bad_any_cast(C++17)
- bad_weak_ptr(C++11)
- bad_function_call(C++11)
- bad_alloc
  - bad_array_new_length(C++11)
- bad_exception
- ios_base::failure(until C++11)
- bad_variant_access(C++17)

what()

Метод ***what()*** для std::exception возвращает C-style строку с описанием исключения

```
catch (std::exception &exception)
{
    std::cerr << "Standard exception: " << exception.what();
}
```

Пример

```
int main()
{
    try
    {
        std::string s;
        s.resize(-1); // генерируется исключение
    }

    catch (std::exception &exception)
    {
        std::cerr << "Standard exception: " << exception.what() << '\n';
    }
}
```

Пример

```
try
  {
      std::string s;
      s.resize(-1);
  }
  catch (std::logic_error &exception)
  {
      std::cerr << "You get logic error!" << '\n';
  }
  catch (std::exception &exception)
  {
      std::cerr << "Standard exception: " << exception.what() << '\n';
  }
```

std::runtime_error

Строго не рекомендуется генерировать ***std::exception*** напрямую, но можно генерировать исключения других классов стандартной библиотеки.

Наиболее популярным является ***std::runtime_error***

```
#include <stdexcept>
```

Пример использования std::runtime_error

```
#include <iostream>
#include <stdexcept>
int main()
{
    try
    {
        throw std::runtime_error("Bad things happened");
    }
    catch (std::exception &exception)
    {
        std::cerr << "Standard exception: " << exception.what();
    }
    return 0;
}
```

Классы-исключения

- Должны наследоваться от ***std::exception*** (тип наследования ***public***)
- Должны переопределять функцию ***what()***

```
const char* what() const { return m_error.c_str(); } // до C++11

const char* what() const noexcept { return m_error.c_str(); } // C++11 и выше
```

noexcept

Спецификатор ***noexcept*** означает, что функция обещает не выбрасывать исключения самостоятельно.

Пример класса-исключения

```
#include <iostream>
#include <string>
#include <exception>
class ArrayException: public std::exception
{
private:
    std::string m_error;
public:
    ArrayException(std::string error)
        : m_error(error)
    {
    }
    const char* what() const noexcept { return m_error.c_str(); } // для
версий C++11 и старше
};
```

Практика

Реализовать функцию, которая принимает строку с датой формата ***dd/mm/yyyy*** парсит эту строку и возвращает заполненную структуру ***Date***. Структура Date должна содержать поля ***date***, ***month***, ***year*** типа ***int***.

В случае, если формат входной строки неверный должно выбрасываться исключения типа ***std::runtime_error***.

Реализация

Как можно
улучшить код?

```
Date ParseDate(string const & str)
{
    stringstream stream(str);
    Date date;
    stream >> date.day;

    if (stream.peek() != '/')
    {
        throw runtime_error("unexpected char");
    }
    stream.ignore(1);
    stream >> date.month;
    if (stream.peek() != '/')
    {
        throw runtime_error("unexpected char");
    }
    stream.ignore(1);

    stream >> date.year;

    return date;
}
```

Реализация. Убираем дублирование

```
void checkNextSymbol(stringstream& stream)
{
    if (stream.peek() != '/')
    {
        throw runtime_error("unexpected char");
    }
    stream.ignore(1);
}
```

```
Date ParseDate(string const & str)
{
    stringstream stream(str);
    Date date;

    stream >> date.day;
    checkNextSymbol(stream);

    stream >> date.month;
    checkNextSymbol(stream);

    stream >> date.year;

    return date;
}
```

Реализация. Использование

```
int main()
{
    string test_date = "11/05/2018";

    try

    {
        Date date = ParseDate(test_date);
        cout << date.day << " " << date.month << " " << date.year << endl;
    }
    catch (exception &e)
    {
        cout << e.what() << endl;
    }

}
```

Что будет выведено на экран?

```
#include <iostream>
#include <exception>
#include <stdexcept>

using namespace std;
int Aggressive() {
    throw runtime_error("fail");
    return 2;
}

int main() {
    try {
        cout << Aggressive() << endl;
    } catch (const exception& ex) {
        cout << "exception: " << ex.what() << endl;
    }
    return 0;
}
```

Что будет выведено на экран? Ответ

```
#include <iostream>
#include <exception>
#include <stdexcept>

using namespace std;
int Aggressive() {
    throw runtime_error("fail");
    return 2;
}

int main() {
    try {
        cout << Aggressive() << endl;
    } catch (const exception& ex) {
        cout << "exception: " << ex.what() << endl;
    }
    return 0;
}
```

exception: fail