ООП

Семестр 2

Лекция 1

Кафедра ИВТ и ПМ

2023

План

- Прошлые темы
- Классы в C#
  - Наследование
  - Коллекции
- Стандартная библиотека (продолжение)
  - Контейнеры
    - stack
    - map
  - regex
- Ссылки и литература

Outline

ООП

- Что такое стандарт оформления кода?

ООП

- Что такое стандарт оформления кода?
- Что такое парадигма программирования?

ООП

- Что такое стандарт оформления кода?
- Что такое парадигма программирования?
- Что такое ООП?

ООП

- Что такое стандарт оформления кода?
- Что такое парадигма программирования?
- Что такое ООП?
- В чём отличие ООП от структурного программирования?

ООП

- Что такое стандарт оформления кода?
- Что такое парадигма программирования?
- Что такое ООП?
- В чём отличие ООП от структурного программирования?
- В чём отличие ООП от модульного программирования?

ООП

- Что такое класс?

ООП

- Что такое класс?
- Что такое объект?

ООП

- Что такое класс?
- Что такое объект?

```
class MyClass{
    int x;
public:
    void foo();
};

...

MyClass c1, *cp;
// MyClass - класс (тип)
// c1 - объект (переменная)
// cp - указатель на объект (переменная)
```

ООП

- Что такое поле класса?
- Что такое метод?

ООП

- Что такое поле класса?
- Что такое метод?
- Какое минимальное количество параметров может быть у метода?

ООП

- Что такое поле класса?
- Что такое метод?
- Какое минимальное количество параметров может быть у метода?
  Можно объявить метод без параметров, однако в метод неявно передаётся указатель на текущий объект - this.

```
class MyClass{
    int x;
public:
    void foo(){
      this->x = 42;
      // с точки зрения программиста "идентификатор" th
      // однако this доступен внутри метода потому,
      // что при описании метода он неявно объявляется
      // как формальный параметр
        }};
```

ООП. Основные принципы

Основные принципы ООП?

ООП. Основные принципы

Основные принципы ООП?

- **Абстрагирование** – выделение значимой информации и исключение из рассмотрения не значимой.
- **Инкапсуляция** – механизм программирования, объединяющий вместе код и данные, которыми он манипулирует, исключая как вмешательство извне, так и неправильное использование данных.
- **Наследование** – механизм позволяющий строить новые определения классов на основе определений существующих классов
- **Полиморфизм** – свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта

Интерфейс

Что такое интерфейс класса?

```
class MyClass{
    int x;
public:
    void setX(int xx);
    int x() const;

    void foo();
    void bar();
    };
```

Интерфейс класса = способы взаимодействия с этим классом
= методы

Интерфейс

Что такое интерфейс (класс-интерфейс)?

Интерфейс

Что такое интерфейс (класс-интерфейс)?

```
class Figure{
public:
    virtual float area()=0;
    virtual float perimeter()=0;
};
```

Абстрактный класс без полей, с абстрактными (без реализации) методами.

ООП. Основные принципы

Инкапсуляция. Какой из примеров реализует инкапсуляцию?

1.
```
class Seconds{
        public: float s;};
```

2.
```
class Seconds{
    public:
        float s;
        void set_secs(float s) {...}
        float secs() {...} const;};
```

3.
```
class Seconds{
        float s;
    public:
        void set_secs(float s) {...}
        float secs() {...} const;};
```

ООП. Основные принципы

Инкапсуляция. Какой из примеров реализует инкапсуляцию?

1. ```
   class Seconds{
           public: float s;};
   ```

2. ```
   class Seconds{
       public:
           float s;
           void set_secs(float s) {...}
           float secs() {...} const;};
   ```

3. ```
   class Seconds{
           float s;
       public:
           void set_secs(float s) {...}
           float secs() {...} const;};
   ```

Пример 3 и 2 (без сокрытия данных).
   ```

ООП. Основные принципы

- Инкапсуляция. Что такое проверка предусловий?
- Инкапсуляция. Зачем она нужна?

ООП. Основные принципы

- Инкапсуляция. Что такое проверка предусловий?
- Инкапсуляция. Зачем она нужна?

```
class Seconds{
        float s;
    public:
        void set_secs(float s1) {
            if ( s1 >= 0 && s1 < 60 )  // проверка предусловия
                this-> s = s1;
                }
```

ООП. Основные принципы

```
class Fighter{
    float mass;
    float max_speed;
    Armament arm;
public:
    ...};

class Airliner{ // passenger aircraft
    float mass;
    float max_speed;
    unsigned capasity;
public:
    ...};
```

Проблема?

ООП. Основные принципы

```
class Fighter{
    float mass;
    float max_speed;
    Armament arm;
public:
    ...};

class Airliner{ // passenger aircraft
    float mass;
    float max_speed;
    unsigned capasity;
public:
    ...};
```

Проблема?

У классов одинаковые поля и соответственно методы доступа к полям также должны быть реализованы дважды.

ООП. Наследование

```
class Aircraft{
    float mass;
    float max_speed;
public:
    ...};

class Fighter: public Aircraft{
    Armament arm;
public:
    ...};

class Airliner: public Aircraft{ // passenger aircraft
    unsigned capasity;
public:
    ...
};
```

ООП. Наследование

Наследование в фреимворках для создание приложений с GUI?

ООП. Наследование

Наследование в фреймворках для создание приложений с GUI?
При создании окон:

```
class MainWindow : public QMainWindow {

    Q_OBJECT  // макрос для создания метаобъекта
    public:
        explicit MainWindow(QWidget *parent = 0);
        ~MainWindow();
    private:

// Класс Ui::MainWindow генерируется автоматически из файла интерфейса
// в нём описаны все элементы интерфейса, их расположение и свойства
пользователя mainwindow.ui
    Ui::MainWindow *ui;
    // другие методы и поля ...
}
```

Программист строит свой класс MainWindow на основе класса QMainWindow из фреймворка Qt.

ООП. Прошлые темы

```
class Circle{
    float r;
public:
    float area() const {return M_PI*r*r;}};
class Square{
    float a;
public:
    float area() const {return a*a;}};
...
// Найти общую площадь всех фигур
list<Circle> circles;
list<Square> squares;
...
float S=0;
for (Circle f: circles) S+=f.area();
for (Square f: squares) S+=f.area();
```

Проблема?

ООП. Прошлые темы

```
class Circle{
    float r;
public:
    float area() const {return M_PI*r*r;}};
class Square{
    float a;
public:
    float area() const {return a*a;}};
...
// Найти общую площадь всех фигур
list<Circle> circles;
list<Square> squares;
...
float S=0;
for (Circle f: circles) S+=f.area();
for (Square f: squares) S+=f.area();
```

Проблема?

Выполнение одинаковых действий с объектами приходится разделять из-за различий в типах.

ООП. Полиморфизм

```
class Figure{
public: virtual float area() const = 0;};

class Circle: public Figure{
    float r;
public: float area() const {return M_PI*r*r;}
};

class Square: public Figure{
    float a;
public: float area() const {return a*a;}};
};

list<Figure*> figs;
figs.push_back(new Circle());
figs.push_back(new Square());
...
float S = 0;
for (Figure *f: figs) S += f->area();
```

ООП. Отношения между классами

- Перечислите отношения между классами
- Когда каждое стоит использовать?
- Как отношения показываются на диаграмме классов?
- Как отношения представляются в коде?

ООП.

- Как работает assert?
- Что такое автоматическое модульное тестирование? Зачем оно нужно?
- Что такое TDD?
- Что такое SOLID?
- Опишите принципы SOLID

Outline

Популярность языков программирования

Programming Language	2023	2018	2013	2008	2003	1998	1993
Python	1	4	8	7	13	27	17
C	2	2	1	2	2	1	1
Java	3	1	2	1	1	19	-
C++	4	3	4	4	3	2	2
C#	5	5	5	8	10	-	-
Visual Basic	6	15	-	-	-	-	-
JavaScript	7	7	10	9	8	22	-
Assembly language	8	12	-	-	-	-	-
SQL	9	251	-	-	7	-	-
PHP	10	8	6	5	6	-	-
Objective-C	17	18	3	46	48	-	-
Ada	28	28	16	18	15	8	8
Lisp	31	31	13	15	14	7	4
Pascal	227	136	15	20	99	12	3
(Visual) Basic	-	-	7	3	4	3	6

См. также jetbrains.com/ru-ru/lp/devecosystem-2021/

Типы данных C#

- Появился в 2000 г.
- Разработан в Microsoft
- Входит в пятёрку самых популярных языков программирования (по оценке TIOBE)
- ОО язык общего назначения
- Компилируется в байт-код
- Есть сборщик мусора

Типы данных C#

Data Type	Range
byte	0 .. 255
sbyte	-128 .. 127
short	-32,768 .. 32,767
ushort	0 .. 65,535
int	-2,147,483,648 .. 2,147,483,647
uint	0 .. 4,294,967,295
long	-9,223,372,036,854,775,808.. 9,223,372,036,854,775,807
ulong	0 .. 18,446,744,073,709,551,615
float	-3.402823e38 .. 3.402823e38
double	-1.7976931348623e308.. 1.7976931348623e308
decimal	-79228162514264337593543950335 to 79228162514264337593543950335
char	A Unicode character.
string	A string of Unicode characters.
bool	True or False.
object	An object.

Структура программы C#

```
// подключение модулей
using System;

namespace YourNamespace {
    // классы, объявленные программистом
    class YourClass
    {
    }

    // обязательный главный класс программы
    class YourMainClass
    {
        // главная функция программы -- статический метод
        static void Main(string[] args)
        {
            //Your program starts here...
        }
    }
}
```

Классы в C#

- Класс – ссылочный тип
  Все переменные – на самом деле ссылки
- Все классы наследуются от `System.Object`
  внимательнее с присваиванием объектов
- `null` - пустое значение

  ```
  MyClass object = null;
  ```
- Объекты создаются динамически

  ```
  MyClass object = new MyClass();
  ```
- Ссылка внутри класса на самого себя - `this`
- В C# есть сборщик мусора

Классы в C#

Объявление классов и создание объектов

```
// перед классом модификатор доступа
public class MyClass  {
// поля и методы...
} // здесь нет точки с запятой


 class Program   {
    static void Main(string[] args) {
        // создание экземпляра класса
        // и сохранение его адреса в ссылке
        MyClass object1 = new MyClass();
        // объявление ссылки на класс
        MyClass object2;
        MyClass object3 = new MyClass();
        MyClass object4 = object3;
        // object3 и object4 идентичны,
        // т.е. указывают на один и тот же объект
        }      }
```

В примере класс приведён в том же файле где и функция Main, однако классы стоит объявлять в отдельных модулях.

Классы в C#

Поля

```
class SampleClass
{
    // модификатор доступа указывается перед каждым полем
    public string sampleField;

    // закрытое поле
    private string sampleField2;

    // открытое поле. константа
    public const int months = 12;
}
```

Классы в C#

Методы

```
class Example{
    public void method1(){
        Console.WriteLine("method1");
    }
    public void method3(int x){
        Console.WriteLine("method2" + x.ToString());
    }
}
//...
class YourMainClass
{
    static void Main(string[] args)
    {
        Example ex = new Example();
        ex.method1();
        ex.method3(42);
    }
}
```

Классы в C#

Конструкторы

- Имя конструктора совпадает с именем класса
- Не указывается возвращаемый тип данных
- Виды конструкторов:
  - Конструктор по умолчанию
  - Конструктор с параметрами
- Если нет конструктора по умолчанию, то компилятор создаёт его автоматически

Классы в C#

Свойства

Свойство — это член, предоставляющий гибкий механизм для чтения, записи или вычисления значения частного поля. Свойства можно использовать, как если бы они были членами общих данных, но фактически они представляют собой специальные методы, называемые методами доступа.

```
// задание и получение значения поля через вызов метода
my_object.set_x( 42 );
int a = my_object.get_x();

// задание значения поля через свойство (неявный возов сеттера)
my_object.X = 42;
// задание значения поля через свойство (неявный возов геттера)
int a = my_object.X;
```

Классы в C#

Свойства

```
class TimePeriod
{  // поле
   private double _seconds;

   // свойство
   public double Hours
   {
        // получение значения
       get { return _seconds / 3600; }

       // задание значения
       set {
       // value - входной параметр
          if (value < 0 || value > 24) // проверка предусловий
             throw new ArgumentOutOfRangeException(
                    \$"{nameof(value)} must be between 0 and 24.");
          _seconds = value * 3600;
       }
   }
}
```

Классы в C#

Свойства. Использование

```
static void Main(string[] args)
        {
            TimePeriod p = new TimePeriod();
            p.Hours = 20;   // вызывается метод set
            double h = p.Hours; // вызывается метод get
        }
```

Классы в C#

Параметры методов

- Передача по значению

  `foo(int x)`

  внутри метода создаётся локальная копия параметра
- Передача по ссылке

  `foo(ref int x)`

  копии не создаётся, внутри метода можно изменить фактический параметр
- Выходной параметр

  `foo(out int x, out int y)`

  аналогично передаче по ссылке, только внутри метода переменной обязательно должно быть присвоено значение. Используется если нужно вернуть из метода несколько значений.

При передаче объекта в параметр по значению, внутри метода создаётся копия ссылки на объект,а не новый объект!

Классы в C#

Перегрузка операторов

```
public class Vector2D
{
    public float x { get; set; }
    public float y { get; set; }

    public static Vector2D operator +(Vector2D v1, Vector2D v2)
    {   Vector2D v = new Vector2D();
        v.x = v1.x + v2.x;
        v.y = v1.y + v2.y;
        return v;
    }
}
```

```
Vector2D v1 = new Vector2D {x=-2, y=4};
Vector2D v2 = new Vector2D {x=1, y=2};
Vector2D v3 = v1 + v2;
```

Классы в C#

Неосвещённые темы:

- Передача и возвращение параметров метода
- Автоматические свойства
- Сокращённая запись свойств
- Константы
- Поля для чтения
- Индексаторы
- Статические члены и классы
- Полиморфизм
- ...

Outline

Наследование

```
public class Vector2D
{
    public double x { get; set; }
    public double y { get; set; }

    public double length(){
                return Math.Sqrt(x*x + y*y); }
}


public class Vector3D : Vector2D {
        public double z { get; set; }

        public double length() {
                return Math.Sqrt(x*x + y*y + z*z); }
}
```

Outline

List

```
List<Vector2D> vectors = new List<Vector2D>();

Vector2D v1 = new Vector2D();
vectors.Add(v1);  // добавление элемента в конец

Random r = new Random();
// добавление элементов в конец
for (int i = 0; i<10; i++){
        Vector2D v = new Vector2D();
        v.x = r.Next(10);
        v.y = r.Next(10);
        vectors.Add(v); }

double sum = 0;
// перебор элементов
foreach  (Vector2D v in vectors) {
        // v - это ссылка на объект
        sum += v.length();}
System.Console.WriteLine(sum);
Vector2D v2 = vectors[2];  // доступ по индексам
int len = vectors.Count;   // число элементов в списке
vectors.RemoveAt(2);       // удаление элемента по номеру
```

https://docs.microsoft.com/ru-ru/dotnet/api/system.collections.generic.list-1?view=netframework-4.7.2

Некоторые другие коллекции

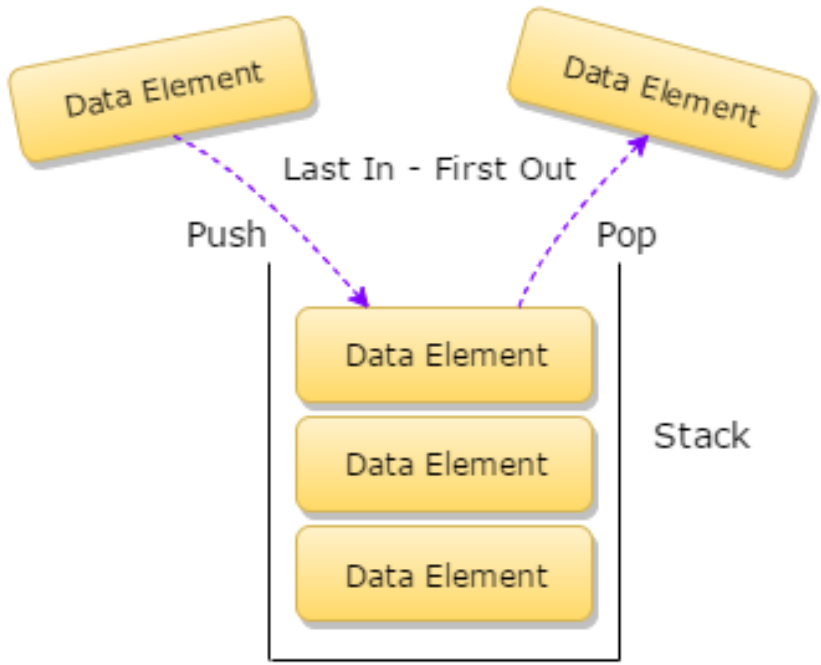
- `Dictionary<TKey,TValue>` - словарь
- `Queue<T>` - очередь
- `Stack<T>` - стэк

Outline

Outline

Стек

Стек (stack — стопка) — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (last in — first out, «последним пришёл — первым вышел»).



Стек. Применение

- Переворачивание последовательности
- Стек вызовов
- Разбор арифметических выражений
- Разбор записи программы на языке программирования
- ...

Стэк

```
#include <stack>

stack<int> s1;
s1.push(20);
s1.push(14);
s1.push(42);

int a;
a = s1.size(); // 3
a = s1.top(); // 42
s1.pop();
a = s1.top(); // 14
s1.pop();
a = s1.top(); // 20
s1.pop();

a = s1.empty(); // true (1)
```

http://www.cplusplus.com/reference/stack/stack/stack/

Ассоциативный массив

Ассоциативный массив (словарь) — абстрактный тип данных (интерфейс к хранилищу данных), позволяющий хранить пары вида «(ключ, значение)» и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу.

Работа со словарём похожа на работу с массивом, где в качестве ключа используется индекс элемента. В словаре же, в качестве ключа может использоваться произвольные значения.

map

Пример

```
#include <map>

map<int, float> m1;
m1[42] = 34.4;
m1[-5] = 444;
m1[0] = -55.01;
m1[452266] = -1.5;

cout << m1[0] << endl; //-55.01

// добавится элемент с ключом 5, со значением по умолчанию
cout << m1[5] << endl;
```

Аналогичный класс в Python называется dict.

map

Пример. Строковый ключ

```
map<string, float> m2;

m2["Иван"] = 185.1;
m2["Олег"] = 170;
m2["Настя"] = 180;
```

map

```
string month = ... ;
int days = 0;
swicth ( month ){
    case "январь":  days=31; break;
    case "февраль": days=28; break;
    case "март":    days=31; break;
    case "апрель":  days=30; break;
    case "май":     days=31; break;
    case "июнь":    days=30; break;
    case "июль":    days=31; break;
    case "август":  days=31; break;
    case "сентябрь":days=30; break;
    case "октябрь": days=31; break;
    case "ноябрь":  days=30; break;
    case "декабрь": days=31; break;
}
```

Как записать код короче?

map

```
const map<string, int> MONTH_DAYS = {
        {"декабрь",  31}, {"январь",   31}, {"февраль",  28},
        {"март",     31}, {"апрель",   30}, {"май",      31},
        {"июнь",     30}, {"июль",     31}, {"август",   31},
        {"сентябрь", 30}, {"октябрь",  31}, {"ноябрь",   30},  };

   string month = ...;

   int days << MONTH_DAYS.at( month );
```

map

Пример. Использование ссылок на функции

```
using Foo = void(*)();

void foo(){
    cout << "foo" << endl;}

void bar(){
    cout << "foo" << endl;}

void baz(){
    cout << "foo" << endl;}

int main(){
    map<int,Foo> dict;

    dict[0] = foo;
    dict[1] = bar;
    dict[2] = baz;

    dict[1]();}
```

Outline

regex

Регулярные выражения (regular expressions, RegEx) — формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов.

Для поиска используется строка-образец (pattern, «шаблон», «маска»), состоящая из символов и метасимволов и задающая правило поиска.

regex

Простой поиск

```
/ февра

TEST STRING

8-е февраля, февральская революция, январь, февраль, март,
Февраль, феВраль
```

Запрос с ИЛИ

```
/ [Фф]евраль

TEST STRING

8-е февраля, февральская революция, январь, февраль, март,
Февраль, феВраль
```

Любые 4 буквы и или цифры подряд

```
r" \w\w\w\w

TEST STRING

8-е февраля, февральская революция, январь, февраль, март,
Февраль, феВраль
```

Любые 4 цифры подряд

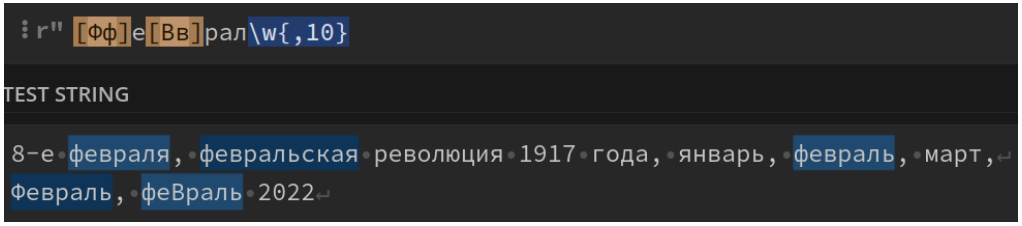
```
r" \d\d\d\d

TEST STRING

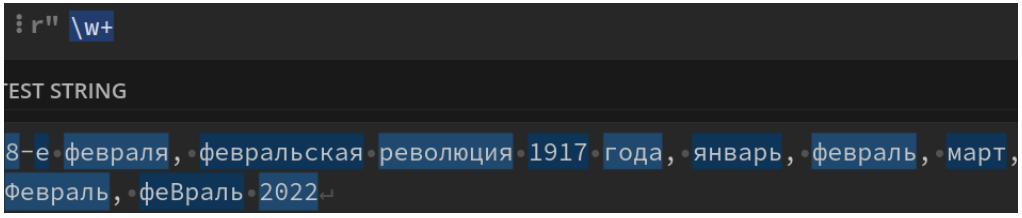
8-е февраля, февральская революция 1917 года, январь, февраль, март,
Февраль, феВраль 2022
```

regex

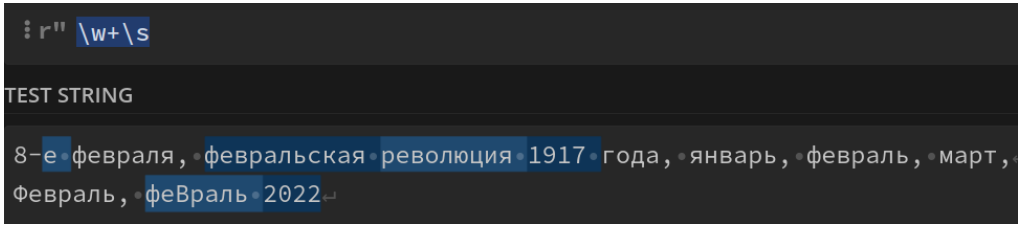
Слово, которое может заканчивается любой последовательностью от 0 до 10 букв и или цифр



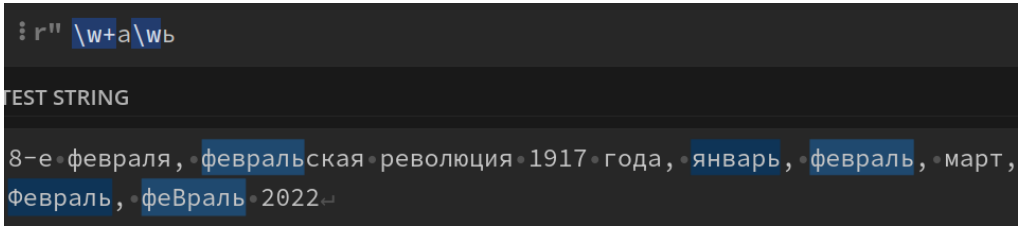
Любые 'слова' – один или больше символ класса *w* подряд



'Слова', после которых есть пробел, табуляция, перенос строки, ...

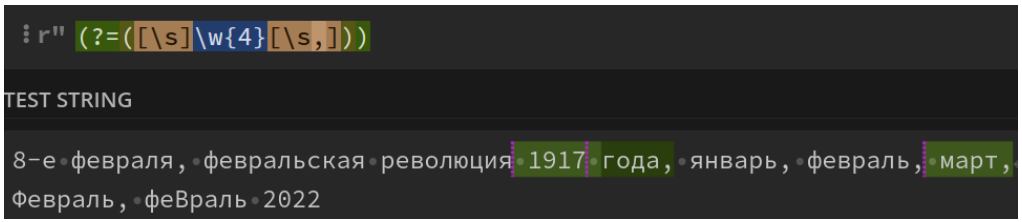


Все слова из 4 w

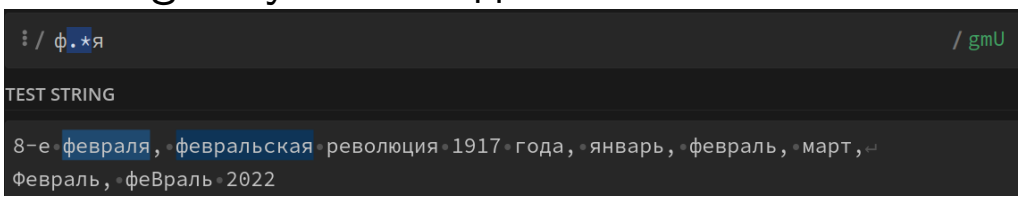


regex

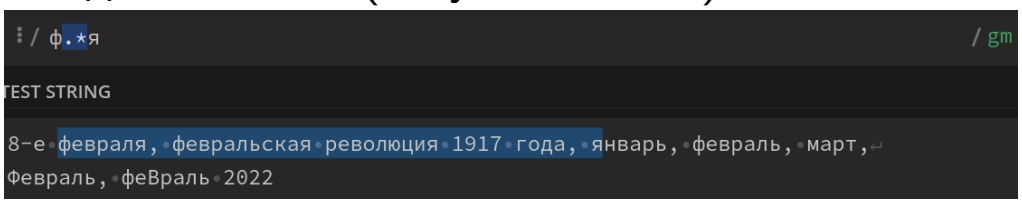
‘Слова‘ длиной 4, перед которыми стоит пробел, после которых пробел или запятая; (?=) – поиск с возможностью пересечения найденных последовательностей



U - Ungreedy - Не жадный поиск



Жадный поиск (по умолчанию)



regex

```cpp
#include <regex>
using namespace std;
int main(){

    string str = "Hello world";
    string number = "42";
    regex rx("\\d\\d");

    bool result = regex_match(str, rx); // false
    result = regex_match(number, rx); // true
}
```

regex

```
string str2 = "Don't Panic!";
regex rx2("Panic");

// ищет совпадение в строке
result = regex_search(str2, rx2);
cout << result << endl;

// проверяет на соответствие строку целиком
result = regex_match(str2, rx2);
cout << result << endl;
```

regex

```
string str2 = "Don't Panic! Panic!";
regex rx2("Panic");
smatch m; // для сохранения информации о совпадениях

// поиск всех совпадений
while( regex_search(str2, m, rx2) ){
    // строка соответствующая выражению
    cout << m[0].str() << endl;
    cout << m.position(0) << endl;
    cout << endl;
    // берём остаток строки для продолжения поиска
    str2 = m.suffix();
}
```

Вывод:

```
Panic
6

Panic
2
```

См. также

- ru.cppreference.com/w/cpp/memory/unique_ptr (Шпаргалка по использованию умных указателей в C++)
- Поддержка типов
- Исключения
- ...

Outline

Ссылки и литература

1. Информация о C# https://metanit.com/sharp/
2. dotnetfiddle.net – онлайн интерпретатор C#

Дополнительно:

3. Java. Базовый курс stepik.org/course/187/syllabus
4. InteliJ IDEA
5. Герберт Шилдт. Java 8: руководство для начинающих, 6-е изд., 2015. - 720 с

Материалы курса

Слайды, вопросы к экзамену, задания, примеры

github.com/VetrovSV/OOP