# *Технологии программирования*

Доц. каф. «Медиаменеджмента и медиапроизводства» Евич Л.Н.

# Лекция 12. ООП в C++.

**Повторение.**

Структуры в C++ используются для логического и физического объединения данных произвольных типов, так же как массивы служат для группирования данных одного типа.

Структура в C++ задаётся следующим образом:

```
struct <имя_структуры> {
  члены (элементы) структуры
};
```

# Лекция 9. Структуры в C++.

Каждый элемент списка имеет уникальное для данного структурного типа имя. Однако следует заметить, что одни и те же имена полей могут быть использованы в различных структурных типах.

```
struct MyStruct1 {
  int id;
  char name[25];
  float mark;
};
struct StructCircle {
  int x,y,z;
  float rad,mark;
};
```

# Лекция 9. Структуры в C++.

```
struct MyStruct1 {
  int id;
  char name[25];
  float mark;
};
struct StructCircle {
  int x,y,z;
  float rad,mark;
};
```

Объявление переменной структурного типа имеет следующий вид:

```
MyStruct1 Stud1, Stud2;
StructCircle Circl1;
```

# Лекция 9. Структуры в C++.

При объявлении переменной происходит выделение памяти для размещения переменной, объем которой не меньше суммы длин всех полей структуры.
Доступ к членам структуры осуществляется посредством оператора . (точка):

```
struct MyStruct1 {
  int id;
  char name[25];
  float mark;
};
MyStruct1 Stud1, Stud2;
Stud1.id=5;
cin>>Stud1.name;
Stud1.mark = 3.4;
```

# Лекция 9. Структуры в C++.

Использовать структуры необходимо для описания пользовательских типов данных во всех случаях, когда это позволяет улучшить читаемость программы.

Вместо структуры

```
struct StructCircle {
  int x,y,z;
  float rad,mark;
};
```

Можно использовать две

```
struct SCoord {
  int x,y,z;
};
struct StructCircle {
  SCoord coord;
  float rad,mark;
};
```

# Лекция 9. Структуры в C++.

```
struct StructCircle {
  int x,y,z;
  float rad,mark;
};
```

```
StructCircle Circl1;

Circl1.x=2;
Circl1.y=5;
Circl1.z=8;
Circl1.rad=15.9;
Circl1.mark=0;
```

```
struct SCoord {
  int x,y,z;
};
struct StructCircle {
  SCoord coord;
  float rad,mark;
};
```

```
StructCircle Circl1;

Circl1.coord.x=2;
Circl1.coord.y=5;
Circl1.coord.z=8;
Circl1.rad=15.9;
Circl1.mark=0;
```

# Лекция 9. Структуры в C++.

Наиболее часто встречающейся ошибкой является включение в структуру взаимозависимых данных. То есть таких членов, значения которых могут быть вычислены на основании других членов структуры.

```
struct SCoord {
  int x,y,z;
};

struct SRect {
  SCoord Top1, Top2;
  float square;
};
```

В данном случае член square может быть вычислен на основании Top1 и Top2.

# (Лекция 11. ) Функции в C++.

Функция — это именованная последовательность описаний и операторов, выполняющая какое-либо законченное действие. Функция может принимать параметры и возвращать значение.

Функция начинает выполняться в момент *вызова*.

*Объявление* функции (*прототип*, *заголовок*, *сигнатура*) задает ее имя, тип возвращаемого значения и список передаваемых параметров.

# (Лекция 11.) Функции в C++.

*Определение* функции содержит, кроме объявления, *тело* функции, представляющее собой последовательность операторов и описаний в фигурных скобках:

```
[class:: (класс)] тип имя ([ список_параметров
]) [throw (исключения)]
{
        тело функции
}
```

**Тип** возвращаемого функцией значения может быть любым, кроме массива и функции (но может быть указателем на массив или функцию).

Если функция не должна возвращать значение, указывается тип void.

# (Лекция 11.) Функции в C++.

```
[class:: (класс)] тип имя ([список_параметров])
[throw (исключения)]
{
        тело функции
}
```

**Список параметров** определяет величины, которые требуется передать в функцию при ее вызове.

Элементы списка параметров разделяются запятыми.

Для каждого параметра, передаваемого в функцию, указывается его тип и имя (в объявлении имена можно опускать).

# (Лекция 11.) Функции в C++.

Пример 1

Пример функции, возвращающей сумму двух целых величин:

```
#include <iostream>;
#include <conio.h>;
using namespace std;
int sum(int a, int b);          // объявление функции

void main() {
    int a = 2, b = 3, c, d;
    c = sum(a, b);              // вызов функции
    cin >> d;
    cout << sum(c, d);          // вызов функции
    _getch();
}
int sum(int a, int b) {         // определение функции
    return (a + b);
}
```

# Лекция 12. Классы в C++.

Класс предоставляет механизм для создания объектов.

В классе отражены важнейшие концепции объектно-ориентированного программирования:

- инкапсуляция,
- наследование,
- полиморфизм.

С точки зрения синтаксиса, класс в C++ – это структурированный тип, образованный на основе уже существующих типов.

Класс можно считать расширением понятия структуры.

# Лекция 12. Классы в C++.

В простейшем случае класс можно определить с помощью конструкции:

***тип_класса имя_класса{список_членов_класса};***

где
*тип_класса* – одно из служебных слов: **class, struct, union**;

*имя_класса* – идентификатор;

*список_членов_класса* – определения и описания типизированных данных и принадлежащих классу функций.

**Функции** – это методы класса, определяющие операции над объектом.
**Данные** – это поля объекта, образующие его структуру. Значения полей определяет состояние объекта.

# Лекция 12. Классы в C++.

**Примеры.**

```
struct date                          // дата
{int month,day,year;                 // поля: месяц, день, год
  void set(int,int,int);             // метод – установить дату
  void get(int*,int*,int*);          // метод – получить дату
  void next();                       // метод – установить следующую дату
  void print();                      // метод – вывести дату
};

struct class complex                 // комплексное число
{double re,im;
  double real()  {return(re);}
  double imag()   {return(im);}
  void set(double x,double y)   {re = x; im = y;}
  void print()    {cout<<“re = “<<re; cout<<“im = “<<im;}
};
```

# Лекция 12. Классы в C++.

Для описания объекта класса (экземпляра класса) используется конструкция

***имя_класса имя_объекта;***

```
date   today,my_birthday;
date   *point = &today;              // указатель на объект типа
date
date   clim[30];                     // массив объектов
date   &name = my_birthday;          // ссылка на объект
```

# Лекция 12. Классы в C++.

В определяемые объекты входят данные, соответствующие членам − данным класса.
Функции − члены класса позволяют обрабатывать данные конкретных объектов класса.
Обращаться к данным объекта и вызывать функции для объекта можно двумя способами

Первый с помощью “квалифицированных” имен:
***имя_объекта. имя_данного***
***имя_объекта. имя_функции***

Второй способ доступа использует указатель на объект
***указатель_на_объект–>имя_компонента***

Например:

```
complex x1,x2;
x1.re = 1.24;
x1.im = 2.3;
x2.set(5.1,1.7);
x1.print();
```

```
complex *point = &x1;
//  или  point = new complex;
point –>re = 1.24;
point –>im = 2.3;
point –>print();
```

# Лекция 12. Классы в C++.

**Доступность компонентов класса.**

В рассмотренных ранее примерах классов компоненты классов являются общедоступными. В любом месте программы, где “видно” определение класса, можно получить доступ к компонентам объекта класса. Тем самым не выполняется основной принцип абстракции данных – инкапсуляция (сокрытие) данных внутри объекта.

Для изменения видимости компонент в определении класса можно использовать спецификаторы доступа:

**public, private, protected**.

# Лекция 12. Классы в C++.

**Доступность компонентов класса.**

**public, private, protected**.

Общедоступные (public) компоненты доступны в любой части программы. Они могут использоваться любой функцией как внутри данного класса, так и вне его.

Доступ извне осуществляется через имя объекта:

**имя_объекта.имя_члена_класса**
**ссылка_на_объект.имя_члена_класса**
**указатель_на_объект->имя_члена_класса**

# Лекция 12. Классы в C++.

## Доступность компонентов класса.

**public, private, protected**.

Собственные (**private**) компоненты локализованы в классе и не доступны извне.

Они могут использоваться функциями – членами данного класса и функциями – “друзьями” того класса, в котором они описаны.

Защищенные (**protected**) компоненты доступны внутри класса и в производных классах.

# Лекция 12. Классы в C++.

**Доступность компонентов класса.**

Изменить статус доступа к компонентам класса можно и с помощью использования в определении класса ключевого слова **class**.
В этом случае все компоненты класса по умолчанию являются собственными.

Пример.

```
class complex
{
  double re, im;                    // private по умолчанию
  public:
  double real(){return re;}
  double imag(){return im;}
  void set(double x,double y){re = x; im = y;}
};
```