Структуры данных

- Как мы будем хранить данные о студенте?

```
int age;                    // возраст
char name[128];             // имя
char surname[128];          // фамилия
char fname[128];            // отчество
int course;                 // курс

age = 22;
course = 2;
```

- Если у нас студентов несколько – становится сложнее.
- Приходится дублировать большое количество связанных по смыслу значений.

```
int age1;                   // возраст
char name1[128];            // имя
char surname1[128];         // фамилия
char fname1[128];           // отчество
int course1;                // курс
```

- Структура – составной тип данных, содержащий набор элементов разных типов. Составляющие структуру элементы называются ее полями.
- Любую информацию можно выразить через числа, но это не очень удобно.
- Для описания структуры используют следующий синтаксис:

```
struct имя_структуры {
    тип_1 имя_поля_1;
    тип_2 имя_поля_2;
};
```

- Любую информацию можно выразить через числа, но это не очень удобно.
- Для хранения связанных по смыслу данных используют структуры.

```
struct Student
{
    int age;                    // возраст
    string name;                // имя
    int course;                 // курс
};

Student s1 {18, "Mike", 1};
Student s2 {19, "Alex", 2};
```

- Для того, чтобы ссылаться к конкретному полю структуры, используется оператор выбора поля (.)

```
s1.age = 19;
s1.course = 2;

void print(Student stud) {
    cout << stud.age << " " << stud.name << " " << stud.course << endl;
}
```

- Со структурой можно работать также, как и с любой другой переменной – хранить в коллекциях, передавать как аргумент функции и т.д.

```
void print(Student stud) {
    cout << stud.age << " " << stud.name << " " << stud.course << endl;
}
```

- У скалярных типов нет поведения, а у структур его можно реализовать.

```
struct Student
{
    int age;                    // возраст
    string name;                // имя
    int course;                 // курс

    void print() {
        cout << age << " " << name << " " << course << endl;
    }
};

Student s1 {18, "Mike", 1};
s1.print();
```

- Со структурой можно работать также, как и с любой другой переменной – хранить в коллекциях, передавать как аргумент функции и т.д.

```
void print(Student stud) {
    cout << stud.age << " " << stud.name << " " << stud.course << endl;
}
```

- У скалярных типов нет поведения, а у структур его можно реализовать.

```
struct Student
{
    int age;                    // возраст
    string name;                // имя
    int course;                 // курс

    void print() {
        cout << age << " " << name << " " << course << endl;
    }
};

Student s1 {18, "Mike", 1};
s1.print();
```

- Над структурами доступны операции присваивания. Если не определен нетривиальный оператор копирования, то по-умолчанию будет выполнено глубокое копирование.

```
Student s1 {18, "Mike", 1};
Student s2 = s1;
s1.print();
S2.print();

Out:
18 Mike 1
18 Mike 1
```

- Определяя структуру, можно сразу создать один или более ее инстансов.

```
struct Student
{
    int age;                    // возраст
    string name;                // имя
    int course;                 // курс

    void print() {
        cout << age << " " << name << " " << course << endl;
    }
} s1, s2;

s1.age = 19; s1.name = "Mike";
s2.age = 20; s2.name = "Alex";

s1.print();
s2.print();
```

- Внутри структуры поля по умолчанию открыты.
- Поля, доступ извне к которым нужно запретить, могут быть помечены private.

```
struct Student
{
    int age;                    // возраст
    string name;                // имя
    int course;                 // курс

    void print() {
        cout << age << " " << name << " " << course << " " << universe << endl; // ok
    }
private:
    string universe = "MIPT";
};

s1.age = 19; s1.name = "Mike";
s2.age = 20; s2.name = "Alex";
s1.universe = "MGU";            // error, universe is private within this context
s1.print();
s2.print();
```