Тема 6

typedef, enum, struct, union

typedef

Можно типам задавать новые имена

```
typedef тип новое_имя [ размерность ];
```

размерность - не обязательно

```
typedef unsigned int UINT;
typedef char Msg[100];
typedef struct {
    char fio[30];
    int date, group;
    double averageMark;} Student;

UINT i, j;              // две переменные типа unsigned int
Msg str[10];            // массив из 10 строк по 100 символов
Student staff[100];     // массив из 100 структур
```

typedef

Где использовать:

- задание коротких псевдонимов для типов с длинным описанием
- для облегчения переносимости программ

enum

Позволяет определить список констант, имеющих различное значение.
В случае определения переменной такого типа, контролируется, чтобы она (переменная) принимала значения только из этого списка.

```
enum [ имя_типа ] { список констант };
```

```
enum Err {ERR_READ, ERR_WRITE, ETT_CONVERT};
Err error;
...
switch (error)
{
    case ERR_READ: ... break;
    case ERR_WRITE: ... break;
    case ERR_CONVERT: ... break;
}

// ERR_READ = 0
// ERR_WRITE = 1
// ERR_CONVERT = 2
```

enum

```
enum
{
    two = 2,
    three,
    four,
    ten = 10,
    eleven,
    fifty = ten + 40
};
```

```
// three = 3
// four = 4
// eleven = 11
```

struct

В отличие от массива, может содержать элементы разных типов

```
struct [ имя_типа ]
{
    тип_1 элемент_1;
    тип_2 элемент_2;
    ...
    тип_n элемент_n;
} [ список_описателей ] ;
```

Элементы структуры - поля структуры, могут иметь любой тип, кроме типа той же структуры, но могут быть указателем на него.

Если имя типа отсутствует, то должен быть указан список описателей переменных, указателей или массивов.
В этом случае определение структуры заменяет тип переменных.

struct

Определение массива структур и указателя на структуру:

```
struct
{
    char fio[100];
    int date, group;
    double averageMark;
} students[100], *ps;
```

Если список отсутствует, описание структуры определяет новый тип:

```
struct Student
{
    char fio[30];
    int date, group;
    double averageMark;
};
Student all[100], *ps;
```

//определение массива типа Student, и указатель на него

struct - инициализация

Для инициализации структуры значения ее элементов перечисляют в фигурных скорбках в порядке их описания:

```
struct
{
    char fio[100];
    int date, group;
    double average;
} student = {"Ivan Pupkin", 31, 101, 4.55};
```

Инициализация массива структур, каждый элемент массива заключается в фигурные скобки

```
struct complex { float real, im;};
complex compl[2][3] = {
    { {1,1}, {1,1}, {1,1} },    // строка 1, он же compl[0]
    { {1,1}, {1,1}, {1,1} }     // строка 2, compl[1]
};
```

struct

=

.

->

```
struct A {int a; double x;};
struct B {A a; double x;} x[2];

x[0].a.a = 1;
x[1].x = 1.0;
```

struct - битовые поля

Битовые поля - особый вид полей структуры. Используются для плотной упаковки данных, например флажков типа "да/нет".

Минимальная адресуемая ячейка памяти - 1 байт, а для хранения флажка достаточно одного бита. При описании битового поля после имени через двоеточие указывается длина поля в битах (целая положительная константа):

```
struct Options{
    bool centerX : 1;
    bool centerY : 1;
    unsigned int shadow : 2;
    unsigned int palette : 4;
};
```

Доступ к полю осуществляется по имени. Адрес поля получить нельзя, однако в остальном используются также как и обычные поля структуры. Размещение битовых полей в памяти зависит от компилятора и аппаратуры. Имя поля может отсутствовать, такие поля служат для выравнивания на аппаратную границу слова.

union

- = struct + все поля по одному адресу
- в каждый момент времени хранится только одно значение
- длина объединения = наибольшей из длин его полей

```
enum PayType {CARD, CHECK};
struct
{
  PayType ptype;
  union payment
  {
    char cardN [20];
    long checkN;
  };
} info;
/*присваивание значений info*/
switch (info.ptype)
{
  case CARD: cout << "You pay with card#" << info.cardN; break;
  case CHECK: cout << "You pay with check " << info.checkN;
break;
```

union

Применяется

- для экономии памяти, когда известно, что более одного поля не потребуется (одновременно)

Ограничения по сравнению со структурами:

- объединение может инициализироваться только значением его первого элемента;
- объединение не может содержать битовые поля;
- объединение не может содержать виртуальные методы, конструкторы, деструкторы и операцию присваивания;
- объединение не может входить в иерархию классов