

Лекция 14

**Типы данных, определяемые пользователем
typedef, enum, struct, union, битовые поля**

Типы данных, определяемые пользователем

В реальных задачах информация, которую требуется обрабатывать, может иметь достаточно сложную структуру. Для ее адекватного представления используются типы данных, построенные на основе простых типов данных, массивов и указателей.

Язык C/C++ позволяет программисту определять свои типы данных и правила работы с ними. Исторически для таких типов сложилось наименование *типы данных, определяемые пользователем*, хотя правильнее было бы назвать их *типами, определяемыми программистом*.

Переименование типов (**typedef**)

Язык C позволяет определять имена новых типов данных с помощью ключевого слова **typedef**. При этом новый тип данных **не создается**, а определяется новое имя существующему типу. Он позволяет облегчить создание машинно-независимых программ. Единственное, что потребуется при переходе на другую платформу, - это изменить оператор **typedef**. Он также может помочь документировать код, позволяя назначать содержательные имена стандартным типам данных. Стандартный вид оператора **typedef** следующий:

typedef тип имя [размерность];

где **тип** — это любой существующий тип данных, а **имя** — это новое имя для данного типа.

Новое имя определяется в дополнение к существующему имени типа, а не замещает его.

Переименование типов (typedef)

Например, можно создать новое имя для **float**, используя

```
typedef float balance;
```

Данный оператор сообщает компилятору о необходимости распознавания **balance** как другого имени для **float**. Далее можно создать вещественную переменную, используя **balance**:

```
balance past_due;
```

Здесь **past_due** – это вещественная переменная типа **balance**, другими словами – типа **float**.

Можно использовать **typedef** для создания имен для более сложных типов. Пример:

```
typedef char msg[100];
```

```
typedef struct {  
    char fio[30];  
    int date, code;  
    double salary;
```

```
} worker;
```

```
msg str[10]; // массив из 10 строк по 100 символов
```

```
worker staff[100]; // массив из 100 структур
```

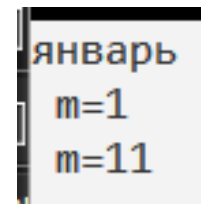
Перечисления (enum)

Перечисления – это набор именованных целочисленных констант, определяющий все допустимые значения, которые может принимать переменная. Перечисления можно встретить в повседневной жизни.

Первое имя в **enum** имеет значение 0, следующее – 1, и т. д. (если для значений констант не было явных спецификаций). Если не все значения специфицированы, то они продолжают прогрессию, начиная от последнего специфицированного значения:

```
enum months { JAN = 1, FEB, MAR, APR, MAY=10, JUN, JUL,  
AUG, SEP, OCT, NOV, DEC};
```

```
enum months m=JAN;  
    switch (m) {  
        case JAN: printf("январь"); break;  
        case FEB: printf("февраль"); break;  
        case MAR: printf("март");break;  
    }  
printf("\n m=%d",m);  
m=JUN; printf("\n m=%d", m);
```



```
январь  
m=1  
март  
m=11
```

Заблуждением считается возможность прямого ввода или вывода символов перечислений/

Структуры (struct)

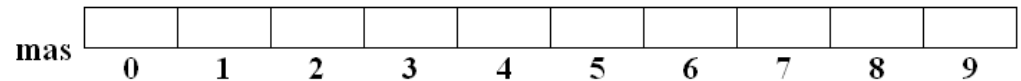
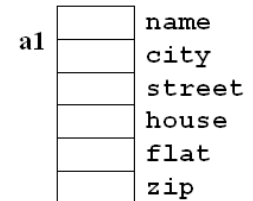
Структуры языка C, аналогично комбинированному типу языка Паскаль, объединяют в себе компоненты (поля) разных типов. Описание структуры начинается с ключевого слова **struct** и содержит список описаний полей в фигурных скобках. За словом **struct** может следовать имя структуры, которое рассматривается как имя структурного типа:

```
struct [<имя структуры>] {  
    <тип1> <поле 1>;  
    <тип2> <поле 2>;  
    . . .  
    <тип n> <поле n>;  
} [список переменных] ;
```

Поля структуры могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него. Если отсутствует имя типа, должен быть указан список переменных.

Структуры (struct)

```
struct addr {  
    char name[30];  
    char city[20];  
    char street [40];  
    int house, flat;  
    unsigned long int zip;  
}  
a1,      // a1 – структурная переменная  
mass[10]; // mass – массив структур
```



Здесь **addr** – это *имя типа*.

Это имя можно использовать в дальнейшем при объявлении других переменных, например:

```
struct addr *sptr; // язык Си, указатель на структуру  
addr *sptr; // в C++ struct можно не писать  
addr a2, a3;
```

Для доступа к полям структуры используется **операция выбора**, обозначаемая точкой:

<имя структурной переменной>.<имя поля>

Структуры (struct)

//присваивание значений структурной переменной a1

a1.house=9; a1.flat=30;

strcpy(a1.city, "Севастополь");

a1		name
	Севастополь	city
		street
	9	house
	30	flat
		zip

//присваивание значений элементу массива из структур mas

mas[1].house=13; mas[1].flat=10;

strcpy(mas[1].city, "Алушта");

mas										
		Алушта								
		13								
		10								
	0	1	2	3	4	5	6	7	8	9

Структуры (struct)

```
addr *aptr;
```

```
aptr=(struct addr *) malloc(sizeof(struct addr));
```

Если структуры размещены в динамической памяти, то **доступ к полям** выполняют **через указатели**:

```
(*aptr).house=9;
```

Скобки здесь необходимы, поскольку приоритет операции “.” выше, чем у операции раскрытия ссылки.

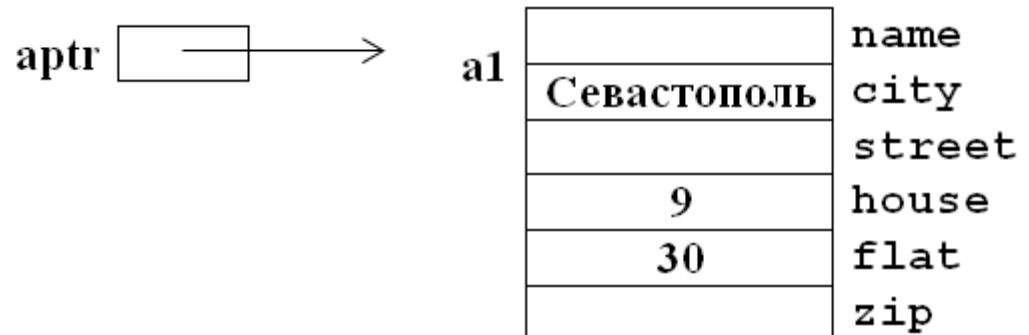
Имеется еще одна форма доступа к полям структуры через указатель:

```
aptr->house=9;
```

Структуры одного типа можно **присваивать** друг другу:

```
*aptr=a1; //присвоение динам. структуре значения статической
```

```
free (aptr) ; // освобождение памяти
```



Структуры (struct)

Структуру можно **передавать в функцию** и возвращать в качестве значения функции:

```
struct addr f1(struct addr a1) {  
    // ...  
    return a1;  
}
```

Статические структуры можно **инициализировать** перечислением значений их полей в порядке описания:

```
struct addr a2 = {"Петя", "Ялта", "Университетская", 33,  
2, 99000};
```

Программа сравнения двух дробей

Напишем программу, сравнивающую две дроби

```
#include <stdio.h>

struct drob {
    int ch, zn;
};

struct drob vvod() {
    drob d;
    printf("введите числитель и знаменатель\n");
    scanf("%d%d", &d.ch, &d.zn);
    return d;
}

void prin_drob (struct drob d) {
    printf("числитель и знаменатель\n");
    printf("%d %d\n", d.ch, d.zn);
}
```

Программа сравнения двух дробей

```
int compare(struct drob d1, struct drob d2) {
    if (d1.ch*d2.zn==d1.zn*d2.ch)
        return 0;
    else if (d1.ch*d2.zn>d1.zn*d2.ch)
        return 1;
    else return -1;
}

struct drob concat(struct drob d1, struct drob d2)
{
    struct drob d;
    d.zn=d1.zn*d2.zn;
    d.ch=d1.ch*d2.zn+d1.zn*d2.ch;
    return d;
}
```

Программа сравнения двух дробей

```
int main() {
    SetConsoleOutputCP(65001);
    drob d1,d2;
    d1=vvod();
    d2=vvod();
    prin_drob(d1);
    prin_drob(d2);
    int x=compare(d1,d2);
    if (x==0) printf("равны");
    else if (x) printf("больше");
        else printf("меньше");

    printf("\n результат суммы: \n");
    d1=cancat(d1, d2);
    prin_drob(d1);
    return 0;
}
```

```
введите числитель и знаменатель
2
3
введите числитель и знаменатель
4
5
числитель и знаменатель
2 3
числитель и знаменатель
4 5
больше
    результат суммы:
числитель и знаменатель
22 15
```

```
введите числитель и знаменатель
3
5
введите числитель и знаменатель
6
10
числитель и знаменатель
3 5
числитель и знаменатель
6 10
равны
    результат суммы:
числитель и знаменатель
60 50
```

Объединения (union)

Объединение (**union**) представляет собой частный случай структуры, все поля которой располагаются по одному и тому же адресу. Формат описания такой же, как у структуры, только вместо ключевого слова **struct** используется слово **union**.

Длина объединения равна наибольшей из длин его полей.

В каждый момент времени в переменной типа объединение хранится только одно значение, и ответственность за его правильное использование лежит на программисте.

Объединения применяют для экономии памяти в тех случаях, когда известно, что больше одного поля одновременно не требуется:

```
union num{  
    int    n;  
    double f;  
};
```

где **num** — имя типа объединения, **n**, **f** — члены объединения.

```
union num d;           // объявление переменной d (язык C)  
num d;                 // объявление переменной d (язык C++)
```

Объединения (union)

При объявлении переменной типа объединение её можно инициализировать значением, которое должно иметь тип первого члена объединения. Например,

```
union num d = { 1 };      // правильно, d = 1
```

```
union num d = { 1.0 };   // ошибка
```

Как и в случае со структурами, для доступа к элементу объединения используется оператор точка '.' или '->'. Например:

```
union num x, *g;
```

```
x.n = 1;          g->n=2;
```

```
x.f = 1.1;        g->f=2.2;
```

Объединения одного типа можно присваивать друг другу. В этом случае оператор присваивания выполняет поэлементное копирование объединений. Например,

```
union num y, z;
```

```
y = z;
```

Также как и структуры объединения **нельзя** сравнивать и объединение не может содержать битовые поля.

Битовые поля

Язык С имеет возможность, называемую битовыми полями, позволяющую работать с отдельными битами. Битовые поля полезны по нескольким причинам. Ниже приведены три из них:

- если ограничено место для хранения информации, можно сохранить несколько логических (истина/ложь) переменных в одном байте;
- некоторые интерфейсы устройств передают информацию, закодировав биты в один байт;
- некоторым процедурам кодирования необходимо получить доступ к отдельным битам в байте.

Битовое поле – это особый тип структуры, определяющей, какую длину имеет каждый член. Стандартный вид объявления битовых полей следующий:

```
struct имя структуры {  
    тип имя1: длина;  
    тип имя2: длина;  
    ...  
    тип имяN: длина;  
}
```


Битовые поля

Битовым полем называется элемент структуры или объединения, который определяет последовательность бит.

Битовое поле может иметь один из следующих типов: **int**, **unsigned** или **signed**. При объявлении битового поля после его имени указывается длина поля в битах. Например,

```
struct device {  
    unsigned active : 1;  
    unsigned ready : 1;  
    unsigned xmt_error : 1;  
} dev_code;
```

Данная структура определяет три переменные по одному биту каждая.

Длина битового поля должна быть неотрицательным целым числом и не должна превышать длины базового типа данных битового поля.



Битовые поля

Инициализируются битовые поля так же, как и обычные элементы структуры. Например,

```
struct device {  
    unsigned active : 1;  
    unsigned ready : 1;  
    unsigned xmt_error : 1;  
} dev_code{0,1,0};  
        // active = xmt_error = 0,  
        // ready = 1
```

Доступ к элементам битового поля осуществляется так же, как и доступ к обычным членам структуры. Например,

```
int main() {  
    printf("\n %d",dev_code.active);  
    dev_code.ready=0;  
    printf("\n %d",dev_code.ready);  
    return 0;  
}
```

Массивы структур

Рассмотрим на примере:

```
#include <stdio.h>
#include <string.h>
#define N 10
struct data
{
    float height;
    char gender,  name[30];
};
struct data x[N];
```

Требуется описать функцию, определяющую средний рост мужчин.

Массивы структур

```
float sr_rost (struct data x[N])  
{  
    float s=0;  
    int k=0;  
  
    for (int i=0; i<N; i++)  
        if (x[i].gender=='m')  
            {s+=x[i].height;  
             k++;  
            }  
    if(s==0) return 0;  
    return s/k;  
}
```

Описать функцию, определяющую имя самого низкого мужчины из группы

Массивы структур

Описать функцию, определяющую имя самого низкого мужчины из группы

```
int poisk1(struct data x[N])
{
    int imin=-1;
    for (int i=0; i<N; i++)
        if (x[i].gender=='m') {imin=i; break;}
    if(imin==-1) return -1;
    for ( i=imin+1; i<N; i++)
        if ((x[i].gender=='m')
            && (x[i].height < x[imin].height))
            imin=i;
    return imin;
}
```

Описать функцию, определяющую есть в группе двое мужчин одного роста.

Массивы структур

Описать функцию, определяющую есть в группе двое мужчин одного роста

```
int poisk2 (struct data x[N])
{
    for (int i=0; i<N-1; i++)
        for (int j=i+1; j<N; j++)
            if ((x[i].gender=='m')
                && (x[i].gender == 'm')
                && (x[i].height == x[j].height))
                return 1;
    return 0;
}
```

Описать функцию, определяющую есть в группе две девушки с одинаковыми именами

Массивы структур

Описать функцию, определяющую есть в группе две девушки с одинаковыми именами

```
int poisk3(struct data x[N])
{
    for (int i=0; i<N-1; i++)
        for (int j=i+1; j<N; j++)
            if ((x[i].gender == 'g')
                && (x[i].gender == 'g')
                && (strcmp(x[i].name, x[j].name)==0))
                return 1;
    return 0;
}
```

Массивы структур

ВЫЗОВ:

```
int main()  
{   x=vvod(x) ;  
  
float sr=sr_rost(x) ;  
    printf("Средний рост мужчин = %f", sr) ;  
  
    int k=poisk1(x) ;  
    printf("Имя самого низкого мужчины->%s",  
x[k].name) ;  
  
if(poisk2(x)==1) printf("yes") ;  
    else printf("no") ;  
  
if(poisk3(x)==1) printf("yes") ;  
    else printf("no") ;  
}
```