**Структура** — это объединение нескольких объектов, возможно, различного типа под одним именем, которое является типом структуры. В качестве объектов могут выступать переменные, массивы, указатели и другие структуры.

Структуры позволяют трактовать группу связанных между собой объектов не как множество отдельных элементов, а как единое целое. Структура представляет собой сложный тип данных, составленный из простых типов.

**Пример** объявления структуры

 struct date

{

  int day;     // 4 байта

  char \*month; // 4 байта

  int year;    // 4 байта

};

Поля структуры располагаются в памяти в том порядке, в котором они объявлены:

 При объявлении структур, их разрешается вкладывать одну в другую.

 struct persone

{

  char lastname[20];   // фамилия

  char firstname[20]; // имя

  struct date bd;     // дата рождения

};

Полями структуры могут быть любые объявленные типы, кроме самой структуры этого же типа, но можно хранить указатель на структуру этого типа:

|  |
| --- |
| **struct** node {  **void**\* value;  **struct** node next;  }; |

//Нельзя!!!

|  |  |  |
| --- | --- | --- |
| **struct** node {  **void**\* value;  **struct** node \*next;  };  //Можно  Структура, объявленная в глобальном контексте, видна всем.  Структура также может быть объявлена внутри функции:   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | #include <stdio.h>  #include <math.h>    **struct** point\_d {  **int** x;  **int** y;  };  **void** main() {  **struct** point\_t {  **int** x;  **int** y;      };  **struct** point\_t A;  **float** distance;        A.x = 10;      A.y = 20;        distance = **sqrt**((**float**) (A.x\*A.x + A.y\*A.y));    **printf**("x = %.3f", distance);      getch(); | |

После того, как мы объявили структуру, можно создавать переменную такого типа с использованием служебного слова struct. Доступ до полей структуры осуществляется с помощью операции точка:

  A.x = 10;

  A.y = 20;

Инициализация полей структуры

struct date bd={8,"июня", 1978};

Также поля структуры можно инициализировать, приняв данные от пользователя с помощью функции scanf.

Современный стандарт си позволяет инициализировать поля структуры по имени. Для этого используется следующий синтакис:

Определение нового типа

Когда мы определяем новую структуру с помощью служебного слова struct, в пространстве имён структур создаётся новый идентификатор. Для доступа к нему необходимо использовать служебное слово struct. Можно определить новый тип с помощью служебного слова typedef. Тогда будет создан псевдоним для нашей структуры, видимый в глобальном контексте.

|  |  |
| --- | --- |
|  |  |

typedef struct thing {

    int a;

    float b;

    const char \*c;

} thing\_t;

int main() {

    thing\_t t = {

        .a = 10,

        .b = 1.0,

        .c = "ololololo"

#include <conio.h>

#include <stdio.h>

//Определяем новую структуру

struct point\_t {

    int x;

    int y;

};

//Определяем новый тип

typedef struct point\_t Point;

void main() {

    //Обращение через имя структуры

    struct point\_t p = {10, 20};

    //Обращение через новый тип

    Point px = {10, 20};

    getch();

}

Теперь при работе с типом Point нет необходимости каждый раз писать слово struct. Два объявления можно объединить в одно

typedef struct point\_t {

    int x;

    int y;

} Point;

Замечание. Если мы создаём новый тип-структуру, полем которого является указатель на этот же тип, то его необходимо объявлять явно с использованием служебного слова struct

?

typedef struct Node {

    int value;

    struct Node \*next;

} Node;

Массивы структур

 Из структур можно создавать массивы также, как массивы других типов. И все форматы определения массива структур будут аналогичны определению массивов других типов:

1

struct person people[10];

В данном случае определен массив структур person из 10 элементов.

Используем массив структур в программе:

#include <stdio.h>

struct person

{

    int age;

    char name[20];

};

int main(void)

{

    struct person people[] = { {23, "Tom"}, {32, "Bob"}, {26, "Alice"}, {41, "Sam"}};

    int n = sizeof(people)/sizeof(people[0]);

    for(int i=0; i<n; i++)

    {

        printf("Name:%s \t Age: %d \n", people[i].name, people[i].age);

    }

    return 0;

}

Указатели на структуру

указатель на структуру создаётся как обычно. Отличие заключается в том, что можно обращаться к полям структуры через указатель с помощью операции "стрелка" (минус + больше). Пример – пользователь вводит число – размер массива пользователей. Поле этого вводит для каждого из них логин и пароль. Третье поле - идентификатор – задаётся автоматически. После этого все пользователи выводятся на экран.

**typedef** **struct** User {

**char** \*login;

**char** \*password;

**int** id;

} User;

**void** jsonUser(User \*user) {

**printf**("{id: %d, login: \"%s\", password: \"%s\"}\n",

            user->id, user->login, user->password);

}

**void** freeUsersArray(User\*\* users, **unsigned** size) {

**unsigned** i;

**for** (i = 0; i < size; i++) {

**free**((\*users)[i].login);

**free**((\*users)[i].password);

    }

**void** main() {

    User \*users = NULL;

**unsigned** size;

**char** buffer[128];

**unsigned** i;

**printf**("Enter number of users: ");

**scanf**("%d", &size);

    size = size <= MAX\_SIZE? size: MAX\_SIZE;

    users = (User\*) **malloc**(size \* **sizeof**(User));

**for** (i = 0; i < size; i++) {

**printf**("user #%d\nname: ", i);

**scanf**("%127s", buffer);

        users[i].id = i;

        users[i].login = (**char**\*) **malloc**(**strlen**(buffer) + 1);

**strcpy**(users[i].login, buffer);

**printf**("password: ");

**scanf**("%127s", buffer);

        users[i].password = (**char**\*) **malloc**(**strlen**(buffer) + 1);

**strcpy**(users[i].password, buffer);

    }

**for** (i = 0; i < size; i++) {

        jsonUser(&users[i]);

    }

    freeUsersArray(&users, size);

}