

Структуры, объединения, перечисления

Лекция + практика

Перечисление enum

Когда выражение должно принимать значение только из заранее определенного конечного множества значений используют **Перечисление** или **enum**.

Ключ. слово

При вычислении выражений вместо именованных констант компилятор подставляет целочисленные значения.

Конечное множество значений именованных констант целого знакового типа по умолчанию

```
enum ACTION {ADD=0, DEL, SAVE, CANCEL};
```

имя

В списке инициализации значения могут повторяться:

```
enum{a=1, b, c, A=1, B, C};  
// a=1,b=2,c=3, A=1,B=2,C=3
```

Структуры (struct). Объявление.

- Иногда удобно оперировать совокупностью переменных как одним программным объектом.
- Структуры C предоставляют возможность формирования составных (агрегатных) типов данных.
- Структура может включать в свой состав произвольное количество типов – полей структуры. Поле структуры может быть любой определенный ранее тип данных – базовый, указатель или такой же составной.
- **Объявление структуры – описание внутреннего устройства нового агрегатного типа, исходя из которого компилятор будет создавать экземпляры пользовательского типа и манипулировать ими.**

При объявлении – память не выделяется. Это только описание, исходя из которого компилятор будет резервировать память при создании экземпляра переменной агрегатного типа.

```
struct имя_типа{  
    список_полей_структуры;  
};
```

```
struct human{  
    int age;  
    char name[30];  
    int weight;  
};
```

Структуры. Создание экземпляров.

- Создание объекта пользовательского типа происходит так же как создание переменной базового типа.
- Компилятор резервирует для такой переменной – **sizeof(структурный_тип)** байтов.

```
// "human.h"  
struct human{  
    int age;  
    char name[30];  
    int weight;  
};
```

Обращение к полям структуры :

Имя_переменной_имя_поля;

```
typedef struct {  
    int age;  
    char name[30];  
    int weight;  
} human;  
  
struct human man1; → human man1;
```

```
#include "human.h"  
int main() {  
    // C  
    struct human man1; // резервирование памяти  
    struct human people[10]; // массив структур  
  
    // C++  
    human woman1;      // резервирование памяти  
  
    ...  
  
    // обращение к полям переменной структурного типа  
    woman1.age = 30;  
    strcpy(woman1.name, "Sofia");  
    return 0;  
}
```

Анонимные структуры

- Иногда для локального использования можно совместить объявление структуры и создание переменной.

```
// "human.h"

struct human{
    int age;
    char name[30];
    int weight;
};
```



Создание
переменных

```
int main(){

    struct human{
        int age;
        char name[30];
        int weight;
    } man1, woman1, peple[10];

    woman1.age = 30;
    strcpy(woman1.name, "Sofia");
    return 0;
}
```

Без использования
имени типа, структура
становится -
анонимной

Инициализация структурных переменных

Инициализация структур похожа на инициализацию массивов с использованием списка инициализаторов.

```
struct human man1 = {30, "Вася", 90};
```

```
struct human people[10] = {  
    {30, "Вася", 90},  
    {40, "Петя", 100},  
    {35, "Юля", 50},  
    ...  
};
```

```
struct human man2 = {30};
```

Неполная
инициализация
так же работает

Инициализация
массивов структур
похожа на
инициализацию
многомерных массивов

```
// "human.h"  
struct human{  
    int age;  
    char name[30];  
    int weight;  
};
```

Вложенные структуры

```
"line.h"  
struct Line{  
    struct Point{  
        int x, y;  
    } point1, point2;  
    int color;  
    int type;  
};
```

Объявление структуры может включать объявление других вспомогательных пользовательских типов

```
#include "line.h"  
int main() {  
    // C  
    struct Point p1; // резервирование памяти  
  
    // C++  
    Line::Point p2 = {1,2}; // резервирование памяти  
    ...  
    return 0;  
}
```

Требуется оператор разрешения области видимости (::) к вложенной структуре

Указатели и структуры

Работать со
структурной
переменной можно
при помощи указателя

```
#include "human.h"
int main() {
// C
    struct human *pman1 = (struct human*)
        malloc(sizeof(struct human)); //резервирование памяти
// C++
    human *pwoman1 = new human;// резервирование
    ...
    // обращение к полям переменной структурного типа

    pwoman1->age = 30;
    strcpy(woman1->name, "Sofia");
    free(pman1);    // C
    delete(pwoman1); //C++
    return 0;
}
```

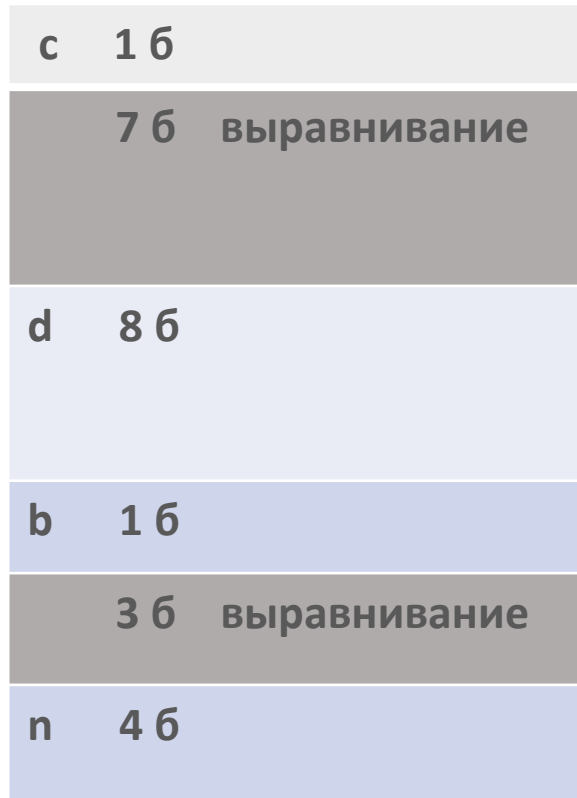
Для обращения к полям
структуры по средством
указателя используется
селектор (->)

Упаковка полей структуры.

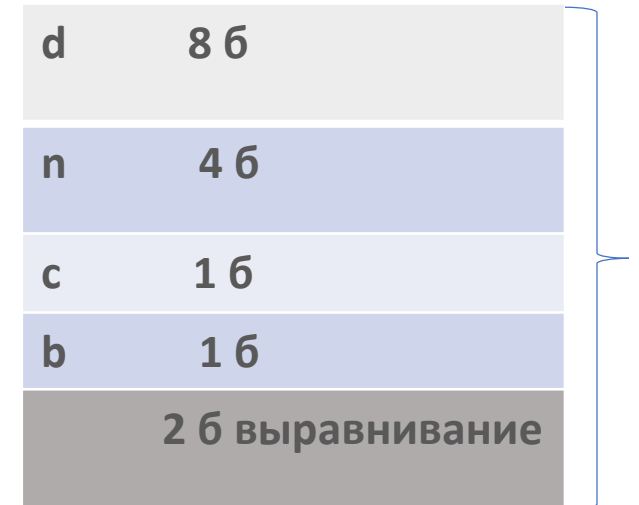
При выделении памяти под структурную переменную:

- Выделится больше или равное сумме всех полей структуры;
- Память выделяется в том порядке в котором поля объявлены в структуре;

```
struct A{  
    char c;  
    double d;  
    bool b;  
    int n;  
};
```



```
struct A{  
    double d;  
    int n;  
    char c;  
    bool b;  
};
```



Структуры и функции

Передача структуры в функцию по значению – не эффективна.
Правильнее передавать её по **указателю** или по **ссылке**.

```
struct Human * find(const struct Human * ar,  
const char *pname);
```

```
void print(const struct Human *p_smbd);
```

```
void print(const Human & ref_smbd);
```

```
void print(struct Human smbd);
```

объявления

```
struct Human{  
    int age;  
    char name[30];  
    int weight;  
};
```

Массив структур в функцию передается аналогично массивам базового типа – адрес начала массива.

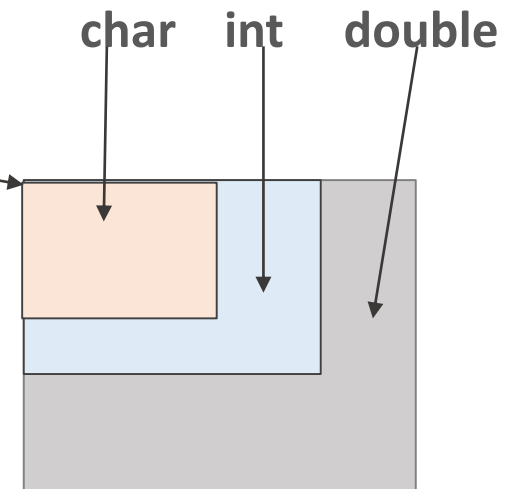
Объединение (union)

Union – агрегатный тип данных похожий на структуру, но имеются отличия: в экземпляре объединения компилятор располагает все поля, начиная с одного и того же адреса. Одним и тем же содержимым `union` можно пользоваться как данными разных типов.

```
union B{  
    double x;  
    int y;  
    char c;  
    char z[4];  
};
```

```
union B big;
```

Размер **union** определяется размером его наибольшего поля.




union используются для унификации обмена данными разного типа между функциями или приложениями. (совместно со структурами).

Чтение/Запись структуры

```
#include <stdio.h>
struct person
{
    char name[20];
    int age;
};
int main() {
    // запись файла
    struct person tom = {"Tom", 22}; // структура
    для записи
    int size = sizeof(struct person);
    FILE *fp = fopen("person.bin", "w");
    // записываем одну структуру
    size_t count = fwrite(&tom, size, 1, fp);
    printf("wrote %zu elements out of %d\n",
count, 1);
    fclose(fp);
```

```
// считывание структуры
struct person unknown; // структура для чтения
fp = fopen("person.bin", "r");
// считываем одну структуру
count = fread(&unknown, size, 1, fp);
if(count == 1)
{
    printf("Name: %s \n", unknown.name);
    printf("Age: %d \n", unknown.age);
}
fclose(fp);
}
```



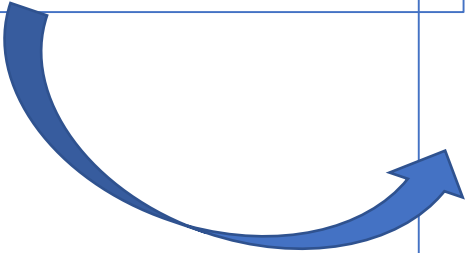
Чтение/Запись массива структур

```
#include <stdio.h>
struct person
{
    char name[20];
    int age;
};
int main() {
    char * filename = "people.bin";
    // массив для записи
    struct person people[] = { {"Tom", 23}, {"Alice", 27},
{"Bob", 31}, {"Kate", 29 } };
    int size = sizeof(people[0]);          // размер всего
массива
    int count = sizeof(people) / size;     //
количество структур
    // запись файла
    FILE *fp = fopen(filename, "w");
    // записываем массив структур
    size_t written = fwrite(people, size, count, fp);
    printf("wrote %zu elements out of %d\n", written,
count);
    fclose(fp);
```

```
// считывание файла
    struct person users[count]; // массив для чтения
структур
    fp = fopen(filename, "r");
    size_t read = fread(users, size, count, fp);
    printf("read %zu elements\n", read);
    if(read > 0)
    {
        for(int i = 0; i < count; i++)
        {
            printf("Name: %s \t Age: %d\n", users[i].name,
users[i].age);
        }
    }
    fclose(fp);
}
```

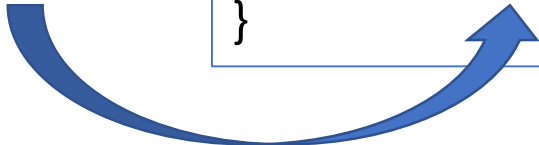
Практика 1. Перечисления (enum)

```
#include <stdio.h>
enum operation{ //
арифметическая операция
    ADD = 1,    // сложение
    SUBTRACT = 2, // вычитание
    MULTIPLY = 4 // умножение
};
```



```
int calculate(int x, int y, enum operation
op){
    switch(op){
        case ADD:
            return x + y;
        case SUBTRACT:
            return x - y;
        case MULTIPLY:
            return x * y;
        default:
            return 0;
    }
}
```

```
int main(void){
    enum operation op = MULTIPLY;
    int result = calculate(4, 3, op);
    printf("Result: %d\n", result); // Result: 12
    result = calculate(4, 3, ADD);
    printf("Result: %d\n", result); // Result: 7
    result = calculate(4, 3, SUBTRACT);
    printf("Result: %d\n", result); // Result: 1
    return 0;
}
```



Практика 2. Структуры

Одну структуру можно присваивать другой структуре того же типа. При копировании элементы структуры получают копии значений

Копирование структур

```
#include <stdio.h>

struct person
{
    int age;
    char * name;
};

int main(void)
{
    struct person tom = {38, "Tom"};
    // копируем значения из структуры tom в структуру bob
    struct person bob = tom;
    bob.name = "Bob";
    printf("Name: %s \t Age: %d \n", bob.name, bob.age);
    printf("Name: %s \t Age: %d \n", tom.name, tom.age);
    return 0;
}
```

Инициализация структур

```
#include <stdio.h>

struct person
{
    int age;
    char * name;
} tom = {38, "Tom"};

int main(void)
{
    printf("Name:%s \t Age: %d", tom.name, tom.age);
    return 0;
}
```

Практика 3. Ввод с консоли данных для структуры.

```
#include <stdio.h>

struct person {
    int age;
    char name[20];
};

int main(void)
{
    struct person tom = {23, "Tom"};
    printf("Enter name: ");
    scanf("%s", tom.name);
    printf("Enter age: ");
    scanf("%d", &tom.age);
    printf("Name:%s \t Age: %d", tom.name, tom.age);
    return 0;
}
```


Практика 4. Объединения (union)

```
#include <stdio.h>

typedef enum{
    NODE_STRING,
    NODE_INT
} node_type;
typedef union{
    int int_value;
    char* str_value;
} node_data;
typedef struct{
    node_type type;
    node_data data;
} node;

void print_node(node n){
    if(n.type == NODE_STRING){
        printf("String: %s\n", n.data.str_value);
    }
    else if(n.type == NODE_INT){
        printf("Int: %d\n", n.data.int_value);
    }
}
```

```
...
int main(void) {
    node n1;
    n1.type = NODE_INT;
    n1.data.int_value=22;
    node n2;
    n2.type = NODE_STRING;
    n2.data.str_value= "Hello World";
    print_node(n1);
    print_node(n2);
    return 0;
}
```

Курсовая работа #1.

Сделать программу «Телефонный справочник». Программа должна хранить связку «имя - телефон» с помощью структуры. Программа должна работать в режимах:

1. Добавление новых записей с консоли. Хранение записей в файле.
2. Выводить все записи справочника
- 3.* Режим поиска существующей записи