

# Структуры, модули, файлы

Юрий Литвинов  
y.litvinov@spbu.ru

28.09.2021

# Структуры

- ▶ Способ группировки родственных по смыслу значений
- ▶ Структура — это тип
  - ▶ В памяти представляется как поля, лежащие друг за другом, возможно, с “дырками” (padding)
  - ▶ Объявляется вне функции
- ▶ Объявление структуры:

```
struct Point {  
    int x;  
    int y;  
};
```

- ▶ Использование:

```
void main() {  
    struct Point p;  
    p.x = 10;  
}
```

## Структуры (2)

- ▶ Или, чтобы **struct** каждый раз не писать:

```
typedef struct {  
    int x;  
    int y;  
} Point;
```

- ▶ **typedef** — объявление синонима типа

- ▶ Использование:

```
void main() {  
    Point p = {10, 20};  
    printf("(%d, %d)", p.x, p.y);  
}
```

- ▶ Продвинутая инициализация:

```
Point p = {.x = 10, .y = 20};
```

# Указатели и структуры

- ▶ Структуры и указатели настолько часто используются вместе, что есть оператор `->` (разыменовывать указатель на структуру и обратиться к её полю)

```
▶ int main() {  
    Point* p = malloc(sizeof(Point));  
    if (p == NULL) {  
        return -1;  
    }  
    p->x = 10;  
    p->y = 20;  
    printf("(%d, %d)", p->x, p->y);  
    free(p);  
  
    return 0;  
}
```

- ▶ То же самое, что `(*p).x` и `(*p).y`

# Операция взятия адреса

```
► int main() {  
    Point p1 = { 10, 20 };  
    Point* p = &p1;  
    int *test = &(p1.x);  
    printf("(%d, %d)\n", p->x, p->y);  
    *test = 30;  
    printf("(%d, %d)\n", p->x, p->y);  
    printf("(%d, %d)\n", p1.x, p1.y);  
}
```

# Структуры и строки

```
typedef struct {
    char *name;
    char phone[30];
} PhoneBookEntry;

int main() {
    PhoneBookEntry entry;
    const char* name = "Ivan Ivanov";
    entry.name = malloc(sizeof(char) * (strlen(name) + 1));
    if (entry.name == NULL) {
        return -1;
    }
    strcpy(entry.name, name);
    strcpy(entry.phone, "+7 (911) 123-45-67");
    printf("%s - %s", entry.name, entry.phone);
    free(entry.name);
    return 0;
}
```

# Полезные операции со строками

Модуль `string.h/cstring`:

- ▶ `strcpy` — скопировать строку в буфер
- ▶ `strcmp` — сравнить две строки
- ▶ `strcat` — склеить две строки (в буфере должно быть достаточно места!)
- ▶ `strlen` — узнать длину строки
- ▶ `strstr` — найти подстроку в строке
- ▶ Строки нельзя сравнивать `==`
- ▶ Строки нельзя присваивать `=`

## Чтение строки прямо в структуру

```
int main() {  
    PhoneBookEntry entry;  
    entry.name = malloc(sizeof(char) * 30);  
    if (entry.name == NULL) {  
        return -1;  
    }  
    scanf("%s", entry.name);  
    scanf("%[^\\n]", entry.phone);  
  
    printf("%s - %s", entry.name, entry.phone);  
  
    free(entry.name);  
    return 0;  
}
```



# Структуры могут указывать сами на себя

```
typedef struct ListElement {  
    int value;  
    struct ListElement *next;  
} ListElement;
```

```
int main() {  
    ListElement* element1 = malloc(sizeof(ListElement));  
    element1->value = 1;  
    ListElement* element2 = malloc(sizeof(ListElement));  
    element2->value = 2;  
    element2->next = NULL;  
    element1->next = element2;  
  
    printf("%i - %i", element1->value, element1->next->value);  
  
    free(element1);  
    free(element2);  
  
    return 0;  
}
```

# Файлы

- ▶ Последовательность байтов на диске
  - ▶ Бывают “сырые” и “текстовые”
    - ▶ Самому файлу всё равно, это лишь способы интерпретации его содержимого
  - ▶ Режимы доступа: r, w, a, r+, w+, a+
  - ▶ Курсор
  - ▶ EOF
- ▶ Функции для работы с файлами:
  - ▶ fopen, fclose, fprintf, fscanf, fseek, ftell, fgetc
- ▶ Файлы надо не забывать закрывать

## Пример, как писать в файл

```
int main() {  
    FILE* out = fopen("ololo.txt", "w");  
    if (out == NULL) {  
        return -1;  
    }  
    fwrite("Ololo\n", sizeof(char), 6, out);  
    fprintf(out, "%s", "Ololo");  
    fclose(out);  
    return 0;  
}
```

- ▶ Файлы нелишне добавлять в проект как «Файлы ресурсов»
- ▶ stdin/stdout — это тоже файлы

# Пример, как читать из файла

```
#include <stdio.h>
```

```
int main() {  
    FILE *file = fopen("test.txt", "r");  
    if (file == NULL) {  
        printf("file not found!");  
        return 1;  
    }  
    char *data[100] = {0};  
    int linesRead = 0;  
    while (!feof(file)) {  
        char *buffer = malloc(sizeof(char) * 100);  
        const int readBytes = fscanf(file, "%s", buffer);  
        if (readBytes < 0) {  
            break;  
        }  
        data[linesRead] = buffer;  
        ++linesRead;  
    }  
    fclose(file);  
    ...  
}
```

# Тонкости

- ▶ Чтение строки целиком: `fscanf(file, "%[^\n]", buffer);`
- ▶ Или: `fgets(buffer, sizeof(buffer), file);`
- ▶ Working directory
  - ▶ Свойства проекта -> Отладка -> Рабочая папка
  - ▶ По умолчанию `$(ProjectDir)`, папка с `.vcxproj`

# Модули

- ▶ Способ группировки кода в логически обособленные группы
- ▶ В C это реализуется с помощью заголовочных файлов и файлов с реализацией
  - ▶ .h и .c
- ▶ В отдельный модуль выносятся объявления типов данных и функции, которые делают одно дело
  - ▶ Например, разные функции сортировки
  - ▶ Или всё для работы с матрицами
- ▶ В интерфейсную часть модуля выносятся только то, что может использовать другой код
  - ▶ Меньше знаешь — крепче спишь
- ▶ Функции, используемые только для реализации, пишутся только в .c-файле
  - ▶ Например, функция разделения массива для быстрой сортировки или `swap`

# Модули

Заголовочный файл:

```
#pragma once
```

```
// Комментарий к функции 1
```

```
int function1(int x, int y);
```

```
// Комментарий к функции 2
```

```
void function2();
```

.c-файл:

```
#include <имя заголовочного файла.h>
```

```
#include <все остальные библиотеки>
```

```
int function1(int x, int y)
```

```
{  
    ...  
}
```

```
void function2()
```

```
{  
    ...  
}
```

# Тонкости

- ▶ Реализации функций в .h-файле писать нельзя
  - ▶ Иначе будет беда, если один .h-ник подключат в два .с-шника
- ▶ Комментарии обязательны
- ▶ #pragma once обязательна
- ▶ Подключать «свой» заголовочный файл в .с обязательно
- ▶ Файлы .h/.c всегда ходят парами, кроме файла с main