Семинар #7: Структуры. Классные задачи.

Часть 1: Основы структур

Структуры служат для объединения нескольких типов в один. В примере ниже был создан новый тип под названием `struct point`. Переменные этого типа будут содержать внутри себя 2 значения типа `float`.

```
#include <stdio.h>

struct point {
    float x, y;
}; // <--------------------------- Не забудьте тут точку с запятой!

int main() {
    struct point a = {2.1, 4.3};
    a.x = 7.8;
    printf("(%f, %f)", a.x, a.y);
}
```

Операции со структурами

1. При создании структуры её элементы можно инициализировать с помощью фигурных скобочек.

   ```
   struct point a = {2.1, 4.3};
   ```

   Однако нельзя таким образом присваивать

   ```
   a = {5.6, 7.8}; // Ошибка, фигурными скобочками можно только инициализировать!
   ```

2. Доступ к элементу структуры осуществляется с помощью оператора точка

   ```
   a.x = 5.6;
   a.y = 7.8;
   ```

3. Структуры можно присваивать друг другу. При этом происходит побайтовое копирование содержимого одной структуры в другую.

   ```
   struct point b;
   b = a;
   ```

Массив структур

Структуры, как и обычные переменные, можно хранить в массивах. В примере ниже создан массив под названием `array`, содержащий в себе 2 точки.

```
#include <stdio.h>
struct point {
    float x, y;
};
int main() {
    struct point array[3] = {{2.1, 4.3}, {7.0, 3.1}, {1.5, 0.2}};
    array[1].x = 1.8;
    printf("(%f, %f)", array[0].x, array[0].y);
}
```

Передача структуры в функцию

Структуры можно передавать в функции и возвращать из функций также как и обычные переменных. При передаче в функцию происходит полное копирование структуры и функция работает уже с копией структуры. При возвращении из функции также происходит копирование.

```
#include <stdio.h>
struct point {
    float x, y;
};
void print_point(struct point a) {
    printf("(%f, %f)", a.x, a.y);
}
struct point add_points(struct point a, struct point b) {
    struct point result;
    result.x = a.x + b.x;
    result.y = a.y + b.y;
    return result;
}
int main() {
    struct point a = {2.1, 4.3}, b = {6.7, 8.9};
    struct point c = add_points(a, b);
    print_point(c);
}
```

Структуры содержащие более сложные типы данных

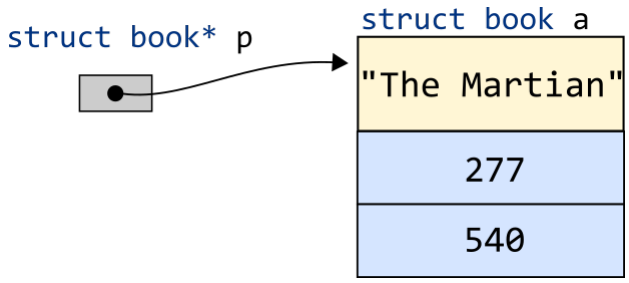
Структуры могут содержать в себе не только базовые типы данных, но и более сложные типы, такие как массивы (в том числе строки), указатели, а также другие структуры.
Пример программы, в которой описывается структура для удобной работы с объектами Книга (`struct book`).

```
#include <stdio.h>
#include <string.h>
struct book {
    char title[50];
    int pages;
    float price;
};
void print_book(struct book b) {
    printf("Book info:\n");
    printf("Title: %s\nPages: %d\nPrice: %g\n\n", b.title, b.pages, b.price);
}
int main() {
    // Создаём книгу:
    struct book a = {"The Martian", 10, 550.0};

    // Меняем количество страниц книги и её название и печатаем её
    a.pages = 369;
    strcpy(a.title, "The Catcher in the Rye");
    print_book(a);

    // Пример работы с массивом структур
    struct book scifi_books[10] = {{"Dune", 300, 500.0}, {"Fahrenheit 451", 400, 700.0},
                                                         {"Day of the Triffids", 304, 450.0}};
    scifi_books[2].price = 2000.0;
    print_book(scifi_books[2]);
}
```

Часть 3: Указатели на структуры:

Указатель на структуру хранит адрес первого байта структуры. Для доступа к полям структуры по указателю нужно сначала этот указатель разыменовать, а потом использовать: `(*p).price`. Для удобства был введён оператор стрелочка `->`, который делает то же самое: `p->price`.

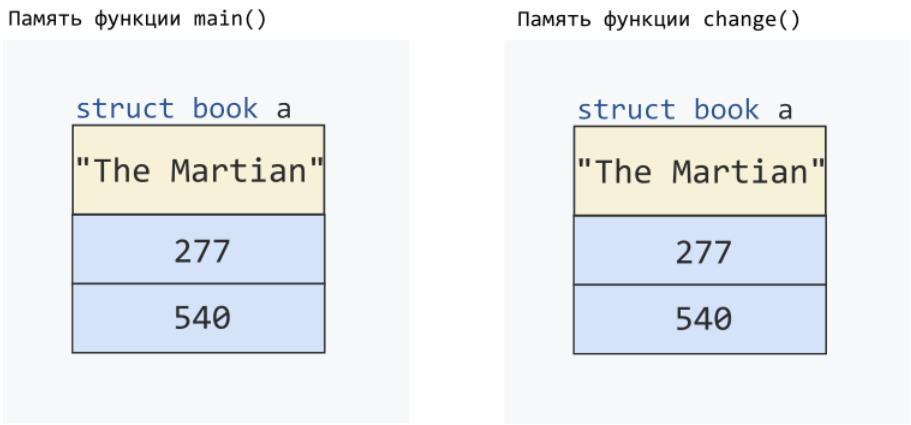
```c
#include <stdio.h>

struct book {
    char title[50];
    int pages;
    float price;
};
int main() {
    struct book a = {"The Martian",277,540};
    struct book* p = &a;

    // Три способа доступа к полю:
    a.price += 10;
    (*p).price += 10;
    p->price += 10;
}
```



Передача по значению

При обычной передаче в функцию всё содержимое копируется. Функция работает с копией.

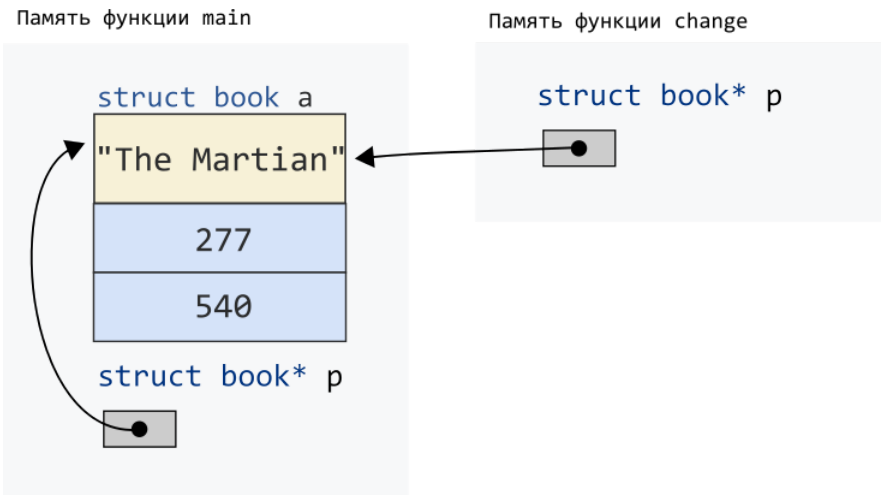
```c
#include <stdio.h>
struct book {
    char title[50];
    int pages;
    float price;
};
void change(struct book a) {
    a.price += 10;
}
int main() {
    struct book a = {"The Martian",277,540};
    change(a); // внутри функции структура a НЕ изменится
}
```



Передача по указателю

При передаче в функцию по указателю копируется только указатель.

```
#include <stdio.h>
struct book {
    char title[50];
    int pages;
    float price;
};
void change(struct book* p) {
    p->price += 10;
}
int main() {
    struct book a = {"The Martian",277,540};
    struct book* p = &a;
    change(p); // внутри функции структура a изменится
}
```



Такой способ передачи имеет 2 преимущества:

1. Можно менять структуру внутри функции, и изменения будут действительны вне функции
2. Не приходится копировать структуры, поэтому программа работает быстрее.

Передача по указателю на константу

Иногда мы не хотим менять структуру внутри функции, но хотим чтобы ничего не копировалось. Тогда желательно использовать передачу по указателю на константу.

```
#include <stdio.h>
struct book {
    char title[50];
    int pages;
    float price;
};
void print_book_info(const struct book* p) {
    printf("Title: %s\nPages: %d\nPrice: %g\n\n", p->title, p->pages, p->price);
}
int main() {
    struct book a = {"The Martian",277,540};
    print_book_info(&a);
}
```

Часть 4: Выравнивание

Пусть есть структура `struct test` и нам нужно узнать её размер если размеры типов `char`, `int` и `double` равны `1`, `4` и `8` байт соответственно.

```
struct test {
    int a;
    char b;
    double c;
};
```

Кажется, что размер этой структур равен сумме размеров состовляющих её элементов, то есть `13`. Но это не так. На самом деле размер этой структуры будет отличаться в зависимости от вычислительной системы, на которой запускается код (как, впрочем, и размеры других типов). Но на большинстве вычислительных систем размер структуры `struct test` будет больше суммы состовляющих её элементов. Это можно проверить с помощью следующего кода:

```
int main() {
    printf("Size of char   = %llu\n", sizeof(char));
    printf("Size of int    = %llu\n", sizeof(int));
    printf("Size of double = %llu\n", sizeof(double));
    printf("Size of test   = %llu\n", sizeof(struct test));
}
```

Причина по которой это происходит заключается в том, что система значительно быстрей работает с данными, если они лежат в памяти по адресам, кратным 4-м или 8-ми. Поэтому компилятор автоматически выравнивает элементы структуры в памяти так, чтобы их адреса были кратны некоторой степени двойки.

Проверьте чему будет равен размер структуры `struct test` в зависимости от последовательности её полей.

Часть 5: Работа с текстовыми файлами

- `fopen`: Открывает файл для чтения/записи
  Режимы открытия файла:

`r`	открыть существующий файл для чтения (read)
`w`	создать новый файл и открыть его для записи (write)
	если файл уже существует, то он удалится перед записью
`a`	открыть для записи в конец файла (append)
`r+`	открыть для чтения/записи, с начала файла
`w+`	создать новый файл и открыть его для чтения/записи
`a+`	открыть для чтения/записи в конец файла

  Для бинарных файлов в Windows нужно добавить символ `b`.

- `fclose`: Закрывает файл

- `fprintf/fscanf`: Функции работают аналогично `printf/scanf`, но только работают с файлом. Файл(указатель на специальную структуру `FILE`) нужно передать первым аргументом.

Пример программы, которая создаёт файл и записывает в него `Hello world!`.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE* fp = fopen("myfile.txt", "w");
    if (fp == NULL)
    {
        printf("Error!\n");
        exit(1);
    }
    fprintf(fp, "Hello world!");
    fclose(fp);
}
```

В этой программе мы делаем следующее:

– Создаём и открываем файл `"myfile.txt"` на запись (так как режим открытия `w`).

– Проверяем получилось ли открыть файл. Если не получилось, то пишем сообщение об ошибке и выходим. В дальнейших примерах эта проверка будет опускаться для экономии места.

– Если получилось открыть, то записываем в файл строку с помощью `fprintf`.

– Закрываем файл.

Задачи

- Скомпилируйте программу `3fprintf.c` и запустите. В результате выполнения программы должен появиться файл `myfile.txt` с содержимым `Hello world!`.

- Напишите программу, которая будет создавать файл `numbers.txt` и записывать туда все числа от 0 до 1000, делящиеся на 7.

- В файле `input.txt` лежат числа (сначала идёт количество чисел, а потом сами числа). Вам нужно считать эти числа и вывести их сумму на экран.

- Измените программу из предыдущей задачи так, чтобы она записывала результат не на экран, а в файл `output.txt`.

Часть 6: Посимвольное чтение из файла

`fgetc` - посимвольное чтение из файла - возвращает ASCII код следующего символа из файла. Если символов не осталось, то она возвращает константу `EOF` равную `-1`.
Пример программы, которая находит количество цифр в файле:

```
#include <stdio.h>
int main()
{
    FILE* f = fopen("input.txt", "r");
    int c;
    int num_of_digits = 0;

    while ((c = fgetc(f)) != EOF)
    {
        if (c >= '0' && c <= '9')
            num_of_digits += 1;
    }
    printf("Number of digits = %d\n", num_of_digits);
    fclose(f);
}
```

Эта программа содержится в файле `5number_of_digits.c`

Задачи

- Написать программу `symbolcount`, которая считает количество символов в файле. название файла должно передаваться через аргумент командной строки:

```
gcc -o symbolcount main.c
./symbolcount war_and_peace.txt
3332371
```

- Написать программу `linecount`, которая находит количество строк в файле.
- Написать программу `wordcount`, которая находит количество слов в файле. Слово это любая последовательность символов, разделённая *одним или несколькими* пробельными, символами. Пробельные символы это пробел, перенос на новую строку(`\n`) либо табуляция(`\t`).