Структуры в языке C.

Прежде чем говорить о структурах, вспомним массивы. Как вы, наверное, помните, массивы предназначены для хранения однотипных данных. Другими словами каждый элемент массива представляет собой значение определенного типа: целое число, символ, строка. Но зачастую, в программах, требуется хранить в одном месте данные разных типов. В качестве примера, в этом уроке будем рассматривать программу каталог книг. Про каждую книгу нам известно: название, автор, год издания, количество страниц, стоимость.

Типы переменных, используемые для хранения подобных данных очевидны:

- **char[]** – автор, название.
- **int** – год издания, количество страниц.
- **float** – стоимость.

На ум сразу приходит следующий вариант реализации. Завести для каждого отдельного качества отдельный массив. Например:

```
int book_date[100];          // дата издания
int book_pages[100];         // количество страниц
char book_author[100][50];   // автор
char book_title[100][100];   // название книги
float book_price[100];       //стоимость
```

Тогда, обращаясь по **i**-му номеру к соответствующему массиву, мы могли бы получить требуемую информацию. Например, вот так мы могли бы вывести на экран автора, название и количество страниц четвертой книги (не забываем, что нумерация элементов массива начинается с нуля).

```
printf("%s-%s %d page", book_author[3], book_title[3], book_pages[3]);
```

Оставим пока что эту реализацию, и посмотрим, как выполнить такую же задачу с использованием ***структур***. Но прежде всего, определим, что такое структура.

Что такое структура.

Задумаемся, что такое структура в обычном понимании этого слова. ***Структура*** – это строение или внутренне устройство какого-либо объекта.

Структура в языке Си – это тип данных, создаваемый программистом, предназначенный для объединения данных различных типов в единое целое.

Прежде чем использовать в своей программе структуру, необходимо её описать, т.е. описать её внутреннее устройство. Иногда это называю ***шаблоном структуры***. Шаблон структуры описывается следующим образом.

ключевое слово

имя структуры

поля структуры

```
sctruct point {
    float x;
    float y;
    int m;
};
```

На картинке слева, мы описали шаблон структуры с именем point. В любом шаблоне структуры можно выделить две основных части: ***заголовок*** (ключевое слово **`struct`** и имя структуры) и ***тело*** (поля структуры, записанные внутри составного оператора).

Точка с запятой в конце обязательна, не забывайте про неё.

Возвращаясь к нашему примеру, опишем структуру book с полями date, pages, author, title, price соответствующих типов.

```
struct book {
        int date;               // дата издания
        int pages;              // количество страниц
        char author[50];        // автор
        char title[100];        // название книги
        float price;            // стоимость
};
```

Попутно отметим, что в качестве полей структуры могут выступать любые встроенные типы данных и даже другие структуры. Подробнее об этом я расскажу в другом уроке. На имена полей накладываются те же ограничения, что и на обычные переменные. Внутри одной структуры не должно быть полей с одинаковыми именами. Имя поля не должно начинаться с цифры. Регистр учитывается.

После того, как мы описали внутреннее устройство структуры, можно считать, что мы создали новый тип данных, который устроен таким вот образом. Теперь этот тип данных можно использовать в нашей программе.

ПРИМЕЧАНИЕ: Обычно структуры описываются сразу после подключения заголовочных файлов. Иногда, для удобства, описание структур выносят в отдельный заголовочный файл.

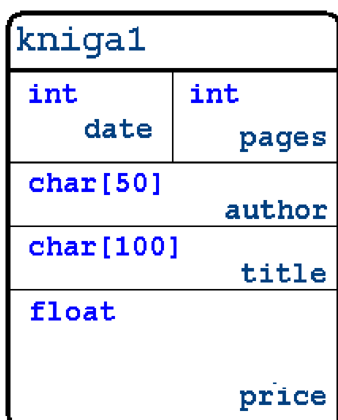
Как объявить структурную переменную (структуру).

Объявление структурной переменной происходит по обычным правилам.

Например:

```
struct book kniga1;
```

Такое объявление создает в памяти переменную типа book, с соответствующими полями.



Отличие структурной переменной от обычной переменной удобно проиллюстрировать на примере с коробками. Считаем, что обычная переменная это просто коробка, в которую можно положить объект определенного типа, например, целое число.

Структурная переменная, это тоже коробка, внутри которой есть отдельные секции для хранения различных данных. Количество этих секций и типы данных, которые мы можем там хранить, задаются шаблоном структуры. На рисунке я постарался схематично изобразить устройство структурной переменной.

Отмечу, что я сознательно не касаюсь вопроса о том, как хранится структура в памяти, так как считаю, что для новичков эти тонкости будут излишни.

Кроме того, еще одну удобную интерпретацию структуры дает нам книга K&R. Можно думать о ней, как о строчке в таблице, где столбцами выступают поля структуры.

int date	int pages	char[50] author	char[100] title	float price

Если кто-то имел дело с реляционными базами данных (MySQL, Oracle), то вам эта такая интерпретация будет очень знакома.

Как инициализировать структуру.

Хорошо, переменную мы объявили. Самое время научиться сохранять в неё данные, иначе, зачем она нам вообще нужна. Мы можем присвоить значения сразу всем полям структуры при объявлении. Это очень похоже на то, как мы присваивали значения массиву при его объявлении.

```
struct book kniga1 = {1998, 230, "D. Ritchi",
                      "The C programming language.", 540.2};
```

int date	int pages	char[50] author	char[100] title	float price
1998	230	D. Ritchi	The C programming language.	540.2

Важный момент. Порядок и тип аргументов, должен совпадать с порядком и типом полей, описанных в шаблоне структуры. В нашем примере мы сначала записали дату, потом количество страниц, имя автора, название книги и стоимость. Пропустить какой-то аргумент нельзя, но можно не объявлять несколько последних. Т.е. вполне допустима следующая конструкция:

```
struct book kniga1 = {1998, 230, "D. Ritchi"};
```

Как обращаться к полям структуры.

Для обращения к отдельным полям структуры используется оператор доступа `"."`. Да-да, обычная точка.

Примеры:

```
kniga1.pages = 250;     // записываем в поле pages переменной kniga1
                        // значение 230.
printf("%s-%s %d page", kniga1.author, kniga1.title, kniga1.pages);
```

Задание: Проверить, что хранится в полях структуры до того, как им присвоено значение.

Сейчас, в одной переменной типа **book** храниться вся информация об одной книге. Но так как в каталоге вряд ли будет всего одна книга, нам потребуется много переменных одного и того же типа **book**. А значит что? Правильно, нужно создать массив таких переменных, то есть массив структур. Делается это аналогично, как если бы мы создавали массив переменных любого стандартного типа.

```
struct book kniga[100];
```

Каждый элемент этого массива это переменная типа **book**. Т.е. у каждого элемента есть свои поля **date, pages, author, title, price**. Тут-то и удобно вспомнить о второй интерпретации структуры. В ней массив структур будет выглядеть как таблица.

	int date	int pages	char[50] author	char[100] title	float price
0					
1					
2					
3					
4					
5					
6					

Используя массив структур получить информацию об отдельной книге можно следующим образом.

```
printf("%s-%s %d page", kniga[3].author, kniga[3].title, kniga[3].pages);
```

Обращаясь к kniga[3].author мы обращаемся к четвертой строке нашей таблицы и столбику с именем author. Удобная интерпретация, не правда ли?

На данном этапе мы научились основам работы со структурами. Точнее мы переписали с использованием структур тот же код, который использовал несколько массивов. Кажется, что особой разницы нет. Да, на первый взгляд это действительно так. Но дьявол, как обычно, кроется в мелочах.

Например, очевидно, что в нашей программе нам часто придется выводить данные о книге на экран. Разумным решением будет написать для этого отдельную функцию.

Если бы мы пользуемся несколькими массивами, то эта функция выглядела бы примерно так:

```
void print_book (int date, int pages, char *author, char *title,
                 float price){

      printf("%s-%s %d page.\nDate: %d \nPRICE: %f rub.\n", author,
                                                             title, pages,
                                                             date, price);
}
```

А её вызов выглядел как-то вот так:

```
print_book (book_date[3], book_pages[3], book_author[3],
                          book_title[3], book_price[3]);
```

Не очень-то и компактно получилось, как вы можете заметить. Легче было бы писать каждый раз отдельный **printf();**.

Совсем другое дело, если мы используем структуры. Так как структура представляет собой один целый объект (большую коробку с отсеками), то и передавать в функцию нужно только его. И тогда нашу функцию можно было бы записать следующим образом:

```
void sprint_book (book temp){

    printf("%s-%s %d page.\nDate: %d \nPRICE: %f rub.", temp.author,
                            temp.title, temp.pages, temp.date, temp.price);
}
```

И вызов, выглядел бы приятнее:

```
sprint_book (kniga[3]);
```

Вот это я понимаю, быстро и удобно, и не нужно каждый раз писать пять параметров. А представьте теперь, что полей у структуры бы их было не 5, а допустим 10? Вот-вот, и я о том же.

Стоит отметить, что ***передача структурных переменных в функцию***, как и в случае обычных переменных осуществляется по значению. Т.е. внутри функции мы работаем не с самой структурной переменной, а с её копией. Чтобы этого избежать, как и в случае переменных стандартных типов используют указатель на структуру. Там есть небольшая особенность, но об этом я расскажу в другой раз.

Кроме того, мы можем ***присваивать структурные переменные***, если они относятся к одному и тому же шаблону. Зачастую это очень упрощает программирование.

Например, вполне реальная задача для каталога книг, упорядочить книги по количеству страниц.

Если бы мы использовали отдельные массивы, то сортировка выглядела бы примерно так.

```
for (int i = 99; i > 0; i--)
    for (int j = 0; j < i; j++)
      if (book_pages[j] > book_page[j+1]){
            //меняем местами значения во всех массивах
            int temp_date;
            int temp_pages;
            char temp_author[50];
            char temp_title[100];
            float temp_price;

            temp_date = book_date[i];
            book_date[i] = book_date[j];
            book_date[j] = temp_date;

            temp_pages = book_pages[i];
            book_pages[i] = book_pages[j];
            book_pages[j] = temp_pages;

            //и так далее для остальных трех массивов
      }
```

Совсем другой дело, если мы используем структуры.

```
for (int i = 99; i > 0; i--)
    for (int j = 0; j < i; j++)
      if (knigi[j].pages > knigi[j+1].pages){
           struct book temp;
           temp = knigi[j];      //присваивание структур
           knigi[j] = knigi[j+1];
           knigi[j+1] = temp;
      }
```

Неоспоримое удобство, не правда ли?

Надеюсь, у меня получилось достаточно убедительно показать преимущества использования структур.

На этой радостной ноте, я и завершаю сегодняшний урок.

Практическое задание:

1. Добавить в структуру поле количество прочитанных страниц.
2. Напишите несколько дополнительных функций для описанной программы.
   - Чтение данных в структуру из файла. В файле запись о каждой книге хранится в следующем формате:

```
Автор||Название||Год издания||Прочитано||Количество страниц||Стоимость
```

Примеры:

```
Khnut||Art of programming. T.1||1972||129||764||234.2
Ritchie||The C Programming Language. 2 ed.||1986||80||512||140.5
Cormen||Kniga pro algoritmy||1996||273||346||239
```

Количество записей в файле не превышает 50 штук.

- Вывод в файл в виде отформатированной таблицы содержимое всего каталога.
- Функцию добавления книги в каталог.
- Функцию поиска по названию книги, по автору и по году издания. Например, вводим год издания, на экране формируется таблица с книгами этого года издания.
- Сортировка книг по стоимости.
- Функцию подсчитывающее количество прочитанных страниц.