

Структуры

Структура как и массив, относится к составным типам данных.

Это объект, состоящий из нескольких элементов (в Паскале — запись, состоящая из полей). Элементами структуры могут являться объекты любого типа, в том числе массивы, структуры.

Объявление структурного типа

Информация о книге (указаны типы и названия полей):

author	name	year	price
Автор	Название	Год	Цена
строка	строка	число	число

```
struct book
```

```
{  
char author[20], name[60];  
int year, price;  
};
```

Тип может быть локальным (внутри функции) или глобальным (вне функции). Описание типа не вызывает выделения памяти.

Определение структурных переменных

Определим структурную переменную kn1:

```
book kn1;  
// book - тип переменной; kn1 — имя переменной;  
book *uk_book;  
// *uk_book - указатель на структуру book
```

Определение структурной переменной без предварительного объявления типа:

```
struct book  
{  
int a; float b;  
} z1, z2;
```

```
// z1, z2 - две переменных типа struct;
```

Принадлежность к внешнему типу определяется местом объявления структурной переменной, а не типа.

```
book kn1={"Иванов Н.И.", "физика", 2009, 175}; //инициализация
```

Для доступа к элементам структуры используются составные имена: *<имя переменной>.<имя элемента>*

```
book kn1;
```

```
kn1.author //имя символьного массива[20] – адрес
```

```
kn1.name //имя символьного массива[60]
```

```
kn1.year, kn1.price //имена переменных типа int
```

```
&kn1.price //адрес
```

Эти имена могут быть использованы так же, как и обычные имена переменных этого же типа, например:

```
kn1.author[6] = 'a';
```

```
scanf("%d",&kn1.price);
```

```
kn1.year =2013 ;
```

```
* (kn1. author+1) = 'p' ;
```

```
gets(kn1.name);
```

Массивы структур

`kniga kns[3]; // массив из трех структур`

`kns[0].author //указатель на массив;`

`kns[1].year //третий элемент второй структуры;`

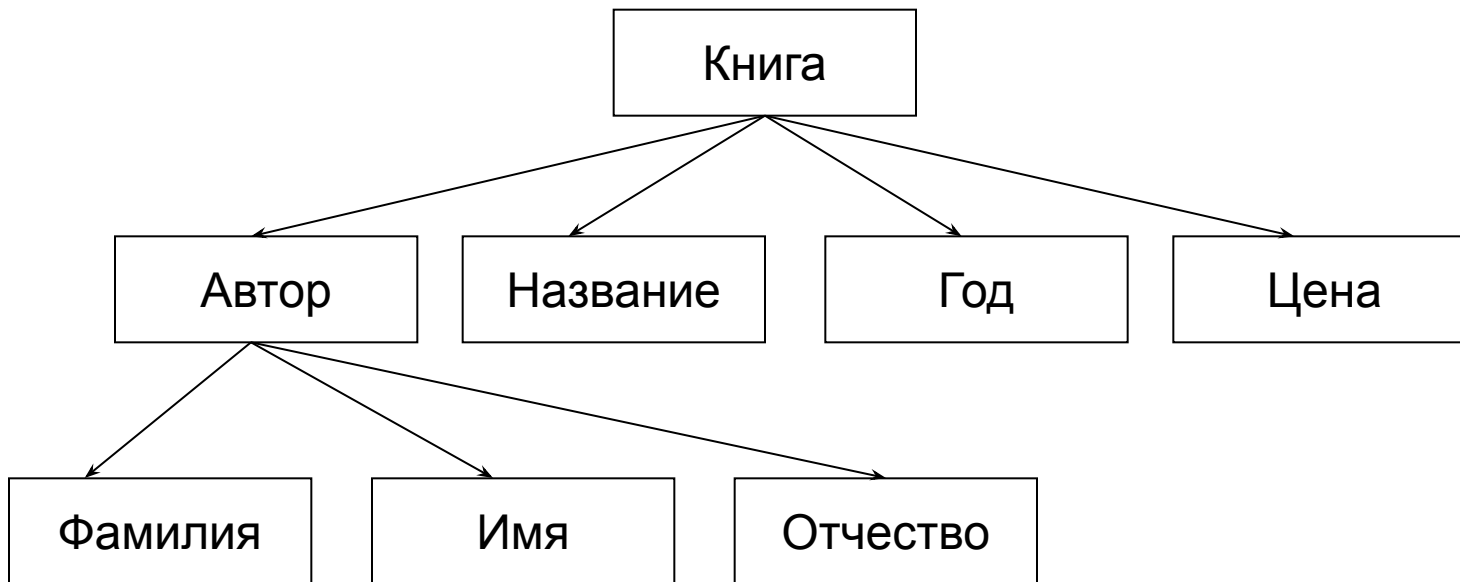
`kns[2].name[0] //первый символ названия.`

Номер элемента массива всегда указывается после имени.

	Автор	Название	Год	Цена
0				
1				
2				

Вложенные структуры

Элементом структуры может быть другая структура, например



```
struct author
{
char fam[15];
char im[10] ;
char ot[15];
};
```

```
struct book1
{
author avt; /*Вложенная структура */
char name[60];
int year, price;
};
```

Указатели на структуры

Использование указателей на структуры удобно по трем причинам:

- так же, как и указатели на массивы, они проще в использовании, чем сами массивы;
- во многих способах представления данных используются структуры, содержащие указатели на другие структуры;
- указатель на структуру удобно передавать в функцию.

Определение переменной и инициализация

```
/* инициализация */
```

```
book1 kn09 = { {"Иван", "Иванович", "Иванов"}, "физика", 2009, 175 };
```

```
Доступ: kn09.avt.fam[1]='р'; //Иванов
```

```
author avt1;
```

```
author *ukavt;
```

```
book1 *ukknig; /* объявление двух указателей на структуры */
```

Все операции с указателями, используемые для обычных переменных, можно применять и к структурным переменным, например


```
ukavt=&avt1;
```

```
ukknig=&kn09;
```

Имя структуры - это не адрес!

```
book1 kni[100]; //массив структур
```

```
ukknig=kni;  ukknig=&kni[0];
```

```
ukknig=kni+1;  ukknig=&kni[1];
```

Прибавление "1" к указателю увеличивает его значение (адрес) на число байтов, которое занимает соответствующий тип.

```
ukknig++;  ukknig=&kni[2]; *ukknig=kni[2];
```

Операции над структурами

Присваивание: `struct book kn1, kn2; kn2=kn1;`

Сравнение структур - сравниваются поля.

Доступ к элементу структуры выполняется при помощи указателя:

```
struct avtor *ukavt, avt1;
```

```
//ukavt - указатель на структуру,
```

```
//avt1 - переменная структурного типа
```

```
ukavt=&avt1; //указатель ссылается на avt1
```

```
avt1.im [0] ='B';
```


С помощью указателя `ukavt` можно получить доступ к элементу одним из двух способов:

1. Операция присоединения

```
ukavt=&avt1; ukavt->im[0]= 'a';
```

Структурный указатель, за которым следует операция `->`, работает так же, как имя структуры с последующей операцией «точка». Нельзя записать `ukavt.im [0]` т.к. `ukavt` - не является именем структуры.

`ukavt` - указатель,

`ukavt->im [0]` - элемент структуры, на которую настроен указатель (элемент имеет тип `char`).

2. Составное имя

```
(*ukavt ).im[0]= 'E';
```

Круглые скобки необходимы, т.к. операция `"."` имеет более высокий приоритет, чем `"*"`.

Содержимое по адресу `ukavt`:

```
ukavt=&avt1; *ukavt=*&avt1;
```

```
//т.е. *ukavt=avt1;
```

Передача информации о структурах в функцию

Структуру можно использовать в качестве формального параметра функции.

1. Можно передавать в качестве фактического параметра элемент структуры или его адрес.

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

```
struct pr  
{  
int a;  
float b;  
};
```

```
int ab (int a, float *b)  
{*b=2.5*(float)a;  
return (2*a);  
}
```

```
int main()  
{ int c; pr pr1;  
scanf ("%d", &pr1.a);  
/* адрес элемента */  
c=ab(pr1.a, &(pr1.b));  
printf("result= %d b=%10.5f\n",c,pr1.b);  
return 0;  
}
```

2. Можно использовать адрес структуры в качестве фактического параметра.

Объявим шаблон и переменные одновременно в задаче определения остатка от деления целых чисел.

```
struct sab
```

```
{ int a,b; } pr1 = {23,3};
```

```
int ab1(sab *prim) //т.е. prim - указатель на структуру pr1
```

```
{  
    return (prim->a%prim->b); //остаток от деления  
}
```

```
int main()
```

```
{  
    printf("Result=%d", ab1(&pr1) );  
    return 0;  
}
```

3. Можно использовать имя массива структур в качестве фактического параметра (то есть адрес первого элемента массива).

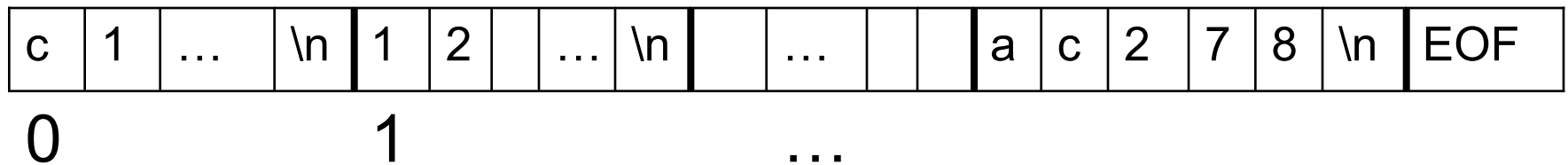
```
#include <stdio.h>
/*найдем сумму всех
элементов всех структур*/
struct pr
{
int a; float b;
};
//глобальная
int main()
{ //в массиве 10 структур
int i;
float ab2(pr *prim);    /*прототип
функции*/
pr pr2[10];
```

```
puts("Input twenty numbers");
for(i=0; i<10; i++)
scanf("%d %f",&pr2[i].a, &pr2[i].b);
printf( "sum=%f", ab2 (pr2) ) ;
/* pr2 — имя массива структур,
адрес первой структуры. */
return 0;
}

float ab2(struct pr *prim)
{
int i;
float sum=0;
for(i=0; i<10; i++, prim++)
sum+=prim->a+prim->b;
return (sum);
}
```

Текстовые файлы

Текстовый файл - последовательность символов (кодов), разделенная на строки.



EOF – специальный символ для проверки и обозначения конца файла. Определен в заголовочном файле `stdio.h`; ввод с клавиатуры этого символа соответствует нажатию клавиш CTRL+Z.

Возможны следующие операции:

```
int ch;
```

```
ch = EOF; //присваивание
```

```
if (ch == EOF) //проверка конца файла
```

Обычно объявляется указатель на файлы, который используется в функциях ввода/вывода, например:

```
FILE *f1;
```

Стандартные указатели на файлы определены в <stdio.h>

```
FILE *stdin,*stdout ;
```

stdin - стандартный входной файл;

stdout - стандартный выходной файл.

Открытие файла. При открытии указывается способ использования файла (считывание, запись, добавление).

Функция открытия файла fopen () :

```
FILE *fopen(char *filename, char *type);
```

Функция fopen () возвращает указатель на файл (если нельзя открыть, то NULL).

Функция fopen () имеет два аргумента:

filename – имя открываемого файла (может содержать путь);

type - способ использования (строка):

"r" – чтение;

"w" - запись ;

"a" – дополнение.

```
FILE *fl ;
```

```
fl = fopen("prog1.res", "w");
```

Здесь prog1.res - имя файла.

Заккрытие файла осуществляется функцией `fclose()`, аргумент которой - указатель на файл:

```
int fclose(FILE *fl);
```

```
fclose (fl) ;
```

При нормальном закрытии файла функция возвращает 0, в противном случае возвращает EOF .

Функция `fcloseall ()` осуществляет закрытие всех потоков, освобождает буферы ввода/вывода, куда данные записываются перед их пересылкой в файл на диск. Буферы используются для ускорения обмена ОЗУ с дисками.

```
void rewind (FILE *fl) – устанавливает указатель в начало файла.
```

Функции ввода и вывода

1. `int fgetc(FILE *fl) ;` - чтение одного символа из файла `fl`. Возвращает код введенного символа. Если достигнут конец файла или произошла ошибка, то вернет EOF.
2. `int fputc(int c, FILE *fl);` - вывод символа `c` с кодом `c` в файл `fl`.

Форматный ввод/вывод

3. `int fscanf(FILE *fl, "форматы", <список аргументов>);`
4. `int fprintf (FILE *fl, "форматы", <список аргументов>);`

Обе функции возвращают количество успешно обработанных аргументов, при ошибке возвращается EOF.

Ввод строки

```
char *fgets(char *str, int n, FILE *fl);
```

Функция `fgets ()` считывает из файла `fl` в строку `str` символы до тех пор, пока не будет выполнено одно из условий:

1. начнется новая строка, т.е. встретится символ `'\n'`;
2. будет достигнут конец файла (EOF);
3. условия 1 и 2 не выполнены, но прочитано `n-1` символов.

После считывания строка дополняется нуль символом `'\0'`. Если при чтении встретился символ конца строки, то он переносится в строку `str` и нуль-символ записывается за ним. Если считывание прошло успешно, то возвращается адрес строки `str`, в противном случае - `NULL`.

Напомним, функция `gets ()` заменяет символ `'\n'` на `'\0'`. Считывание символов осуществляется из стандартного входного потока `stdin`. Если входной поток прерывается символом перехода на новую строку `'\n'`, то он отбрасывается и не попадает в строку `str`.

Пример: Символы переписываются функцией puts () в стандартный выходной поток stdout, строка str дополняется символом конца строки '\n'.

```
char str[10];
```

```
FILE *fl;
```

```
fl=fopen("f1.txt","r"); //fl:
```

a	b	c	d	\n
---	---	---	---	----

```
fgets(str,3,fl); //str:
```

a	b	\0
---	---	----

```
puts(str);
```

ab

```
fgets(str,10,fl); //str:
```

a	b	c	d	\n	\0
---	---	---	---	----	----

```
puts(str);
```

abcd

Вывод строки

```
int fputs (char *str, FILE *fl);
```

Строка `str`, ограниченная символом `'\0'`, переписывается в файл `fl`, причем символ `'\0'` **отбрасывается**.

```
fputs ("abcd",fl) ;
```

```
fputs("ef\n",fl);
```

```
abcdef\n
```

Результат в файле `fl`:

В отличие от `puts ()` функция `fputs` не добавляет символ `'\n'` в файл.

Определение конца файла

```
int feof(FILE *fl);
```

Выдает истинное значение, если при чтении достигнут конец файла, в противном случае - нулевое.