

**Структуры**

**Структура** – это составной объект языка Си, содержащий данные, объединенные в группу под одним именем. Данные, входящие в эту группу, называют **полями** (членами структуры). В отличие от массивов поля могут иметь различные типы.

Для создания объектов-структур надо:

- объявить **структурный тип** данных, т.е. описать пользовательский тип (выделения памяти не происходит);
- объявить **структурные переменные** описанного типа, при этом происходит выделение памяти.

**Объявление структурного типа** выполняется в виде **шаблона**, общий формат которого:

```
struct Имя_Типа {  
    Описание полей  
};
```

Структурный тип обычно декларируется в глобальной области, т.е. до первой выполняемой функции. Тогда его можно использовать во всех функциях, входящих в проект.

Структурный тип обычно применяется для групповой обработки объектов. Параметрами таких операций являются адрес и размер структуры.

Так как одним из параметров обработки структур является размер, нельзя объявлять поля структуры указателем на объект переменной размерности, т.е. использовать указатели на ***char*** и объекты классов *String* и *AnsiString*.

Пример шаблона для обработки информации о результатах сессии студентов

```
struct Spisok {  
    char fio [20];    - Фамилия студента  
    double s_bal;    - Средний балл  
};
```

**Создание структурных переменных** можно выполнить двумя способами.

**Способ 1.** В любом месте программы объявить переменные, используя описанный ранее структурный тип.

Например для описанного ранее типа ***Spisok*** :

**Spisok zap, \*pzap, mas\_zap[30];**

Объявлены

**zap** – структурная переменная (запись),

**pzap** – указатель на структуру,

**mas\_zap[30]** – массив структур.

**Способ 2.** Объявляют переменные в шаблоне структуры между закрывающейся фигурной скобкой и символом «;».

Для приведенного ранее примера:

```
struct Spisok {  
    char fio[20];  
    double s_bal;  
} zap, *pzap, mas_zap[30] ;
```

В таком случае, если объявлены все необходимые переменные, *Имя\_Типа* (***Spisok***) может отсутствовать.

**Обращение к полям структур** выполняется с помощью составных имен, которые образуются двумя способами:

- 1) при помощи операции **принадлежности** ( **.** ) от значения (имени структурной переменной) к полю:

**Имя\_Структуры . Имя\_Поля**

или

**( \*Указатель\_Структуры ) . Имя\_Поля**

- 2) при помощи операции **косвенной адресации** ( **→** ) от адреса к полю

**Указатель\_Структуры → Имя\_Поля**

или

**( &Имя\_Структуры ) → Имя\_Поля**

Для объявленных ранее переменных

***Spisok zap, \*pzap, mas\_zap[30];***

содержащих поля ***char fio[20];*** и ***double s\_bal;***

1) обращение к полям ***fio*** и ***s\_bal*** переменной ***zap***:

а) с помощью операции принадлежности

***zap . fio*** и ***zap . s\_bal***

б) с помощью операции косвенной адресации

***( &zap ) -> fio*** и ***( &zap ) -> s\_bal***

2) обращение к первому символу строки ***fio*** переменной ***zap***:

***zap . fio [0]***



3) обращение к полям ***fio*** и ***s\_bal*** от указателя ***pzap***:

а) с помощью операции косвенной адресации

***pzap -> fio***      и      ***pzap -> s\_bal***

б) с помощью операции принадлежности

***( \*pzap ) . fio***      и      ***( \*pzap ) . s\_bal***

4) обращение к полям ***fio*** и ***s\_bal*** *i*-го элемента массива ***mas\_zap***:

а) с помощью операции принадлежности

***mas\_zap[i] . fio***      и      ***mas\_zap[i] . s\_bal***

б) с помощью операции косвенной адресации

***( mas\_zap+i ) -> fio***      и      ***( mas\_zap+i ) -> s\_bal***

Рассмотрим пример программы создания динамического массива структур, содержащих поля ***fio*** и ***s\_bal*** (как раньше), его заполнение, вывод всей информации на экран и поиск сведений о студентах, у которых средний балл выше 7,99 баллов.

...

struct **Spisok** {           - Шаблон структуры

    char **fio** [21];

    double **s\_bal**;

};

Spisok **In** (void);       - Функция ввода

void **Out** (Spisok);   - Функция вывода

void main ()

{

**Spisok \*Stud**;                   - Указатель для массива

    int i, n;

    cout << " Input **n** : ";       - Количество студентов

    cin >> n;

```
Stud = new Spisok [n]; - Захват памяти
for(i=0;i<n;i++)          - Заполнение массива
    Stud [ i ] = In ( );
cout << "\n\t Spisok " << endl;
for(i=0;i<n;i++)
    Out ( Stud [ i ] );    - Вывод информации
cout << "\n\t Ball > 7.99" << endl;
for(i=0;i<n;i++)
    if ( Stud [ i ] . s_bal > 7.99)    - Поиск
        Out ( Stud [ i ] );          и вывод
delete [ ] Stud;          - Освобождение памяти
}
```

//----- Функция **Ввода** одного элемента структуры -----

```
Spisok In ( ) {
```

```
Spisok z;
```

```
cout << "\n FIO  - ";
```

```
fflush ( stdin );      - Очистка стандартного буфера ввода
```

***stdin***, необходимая в данном случае перед использованием функции ***gets***

```
gets ( z.fio );
```

```
cout << "\n Ball  - ";
```

```
cin >> z.s_bal;
```

```
return z;
```

```
}
```

//----- Функция **Вывода** одного элемента структуры -----

```
void Out ( Spisok z ) {
```

```
cout << setw(20) << z.fio << " \t " << z.s_bal << endl;
```

```
}
```

**Ф а й л ы**

**Файл** – это набор данных, размещенный на внешнем носителе и рассматриваемый в процессе обработки как единое целое. В файлах размещаются данные, предназначенные для длительного хранения.

Различают два вида файлов: *текстовые* и *бинарные*.

**Текстовые файлы** представляют собой последовательность символов и могут быть просмотрены и отредактированы с помощью любого текстового редактора.

**Бинарные (двоичные) файлы** представляют собой последовательность данных, структура которых определяется программно.

Файлы рассматриваются компилятором как последовательность (поток байт) информации. В начале работы любой программы автоматически открываются стандартные потоки ввода (***stdin***) и вывода (***stdout***).

Для файлов определен ***указатель*** (маркер) чтения-записи данных, который определяет текущую позицию доступа к файлу.

В языке Си имеется большой набор функций для работы с файлами, большинство в ***stdio.h*** и ***io.h***.

Потоки данных, с которыми работают функции ввода-вывода по умолчанию, буферизированы. При открытии потока с ним связывается определенный участок памяти, который называется ***буфером***. Все операции чтения-записи ведутся через этот буфер.



Для обработки любого файла необходимо выполнить следующие действия:

- 1) открыть файл;
- 2) обработать данные файла (запись, чтение, поиск и т.п.);
- 3) закрыть файл.

## ***Открытие файла***

Каждому файлу в программе присваивается внутреннее логическое имя, используемое в дальнейшем при обращении к нему.

Логическое имя (имя файла) – это указатель на файл, т.е. на область памяти, где содержится вся необходимая информация о нем.

Формат объявления :

***FILE \*Имя\_Указателя;***

***FILE*** – структурный тип, описанный в библиотеке ***stdio.h*** (содержит 9-ть полей).

Прежде чем начать работать с файлом, его нужно открыть для доступа с помощью функции

***foren ( Имя\_Файла, Режим )***

Данная функция фактически **Имени Файла** на носителе (дискета, винчестер) ставит в соответствие логическое имя (***Указатель файла***).

Имя файла и путь к нему задается первым параметром – строкой, например:

“***d:\\work\\Sved.txt***” – файл с именем ***Sved***, расширением ***txt***, находящийся на винчестере в папке ***work***.

Обратный слеш «\», как специальный символ в строке записывается дважды.

Если путь к файлу не указан, его размещением будет текущая папка.

При успешном открытии функция ***fopen*** возвращает указатель на файл (указатель файла).

При ошибке возвращается ***NULL***.

Ошибки обычно возникают, когда неверно указывается путь к открываемому файлу, например, если указать путь, запрещенный для записи.

Второй параметр – строка, в которой задается режим доступа к файлу:

**w** – файл открывается для записи (**write**);  
если файла нет, то он создается; если файл уже есть, то прежняя информация уничтожается;

**r** – файл открывается для чтения (**read**); если такого файла нет, то возникает ошибка;

**a** – файл открывается для добавления (**append**)  
новой информации в конец;

**r+** – файл открывается для редактирования  
данных;

**t** – файл открывается в текстовом режиме;

**b** – файл открывается в двоичном режиме.

Последние два режима используются совместно с рассмотренными выше. Возможны следующие комбинации режимов доступа: ***w+b***, ***wb+***, ***rt+***, а также некоторые другие комбинации.

По умолчанию файл открывается в текстовом режиме.

Пример открытия файла:

**FILE \*f;**      – Объявляется указатель ***f***

**f = fopen ("Dat\_sp.dat ", "w");**

– открывается для записи файл с указателем ***f*** в текущей папке, имеющий имя ***Dat\_sp.dat***,

или более кратко:

**FILE \*f = fopen ("Dat\_sp.dat", "w");**

## ***Закрытие файла***

После работы с файлом доступ к нему необходимо закрыть с помощью функции

***fclose ( Указатель\_Файла );***

Для предыдущего примера: ***fclose ( f );***

Если надо изменить режим доступа к уже открытому файлу, то его необходимо закрыть, а затем открыть с другим режимом:

***freopen ( Имя\_Файла, Режим, Указатель );***

- закрывается файл с заданным в третьем параметре *Указателе* (аналогично функции ***fclose***), а затем открывается файл, используя первый и второй параметры (аналогично функции ***fopen***).

## ***Запись-чтение информации***

Основными действиями при работе с файлами являются запись и чтение информации.

Все действия по чтению-записи данных в файл можно разделить на три группы:

- операции посимвольного ввода-вывода;
- операции построчного ввода-вывода;
- операции ввода-вывода блоками.

Рассмотрим основные функции записи-чтения данных.



Создание текстовых результирующих файлов обычно необходимо для оформления различных отчетов.

Для работы с текстовыми файлами чаще всего используются функции

***fprintf, fscanf, fgets, fputs***

Параметры и действия этих функций аналогичны рассмотренным ранее функциям ***printf, scanf, gets*** и ***puts***.

Отличие состоит в том, что ***printf*** и др. работают по умолчанию с экраном монитора и клавиатурой, а функции ***fprintf*** и др. – с файлом, указатель которого является одним из параметров.

Например:

. . .

```
FILE *f1;
```

```
int a = 2, b = 3;
```

```
printf ( " %d + %d = %d \n ", a, b, a+b);
```

```
fprintf ( f1," %d + %d = %d\n ", a, b, a+b);
```

```
fclose ( f1 );
```

. . .

Просмотрев файл **f1** любым текстовым редактором, можно убедиться, что данные в нем располагаются так же, как и на экране.

**Бинарные файлы** обычно используются для обработки данных, состоящих из структур, чтение и запись которых удобно выполнять **блоками**.

Функция

***fread*** ( *p*, *size*, *n*, *f* );

выполняет чтение из файла «***f***» «***n***» блоков размером «***size***» байт каждый в область памяти, адрес которой «***p***». В случае успеха функция возвращает количество считанных блоков.

Функция

***fwrite*** ( *p*, *size*, *n*, *f* );

выполняет запись в файл «***f***» «***n***» блоков размером «***size***» байт каждый из области памяти, с адресом «***p***».

## ***Позиционирование в файле***

Каждый открытый файл имеет скрытый указатель на текущую позицию в нем.

При открытии файла этот указатель устанавливается на позицию, определенную режимом, и все операции в файле будут выполняться с данными, начинающимися в этой позиции.

При каждом чтении (записи) указатель смещается на количество прочитанных (записанных) байт – это ***последовательный доступ к данным***.

С помощью функции ***fseek*** можно выполнить чтение или запись данных в произвольном порядке.

## ***fseek ( f, size, code )***

выполняет смещение указателя файла ***f*** на ***size*** байт в направлении ***code*** :

**0** – смещение от начала;

**1** – смещение от текущей позиции;

**2** – смещение от конца файла.

Смещение может быть как положительным, так и отрицательным, но нельзя выходить за пределы файла.

В случае успеха функция возвращает 0, 1 –при ошибке, например, выход за пределы файла.

Доступ к файлу с использованием этой функции называют ***произвольным доступом***.

Рассмотрим некоторые полезные функции:

- 1) ***ftell*** ( ***f*** ) – определяет значение указателя на текущую позицию в файле, **-1** в случае ошибки;
- 2) ***fileno*** ( ***f*** ) – определяет значение дескриптора (***fd***) файла ***f***, т.е. число, определяющее номер файла;
- 3) ***filelength*** ( ***fd*** ) – определяет длину файла в байтах, имеющего дескриптор ***fd***;
- 4) ***chsize*** ( ***fd***, ***pos*** ) – выполняет изменение размера файла, имеющего номер ***fd***, признак конца файла устанавливается после байта с номером ***pos***;
- 5) ***feof*** ( ***f*** ) – возвращает ненулевое значение при правильной записи признака конца файла.