2.9. Файлы

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 ASCII символов и могут быть просмотрены и отредактированы с помощью любого текстового редактора. Эта последовательность символов разбивается на строки символов, при этом каждая строка заканчивается двумя кодами «перевод строки», «возврат каретки»: 13 и 10 (0xD и 0xA).

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

В языке Си не предусмотрены никакие заранее определенные структуры файлов. Все файлы рассматриваются компилятором как последовательность (поток байт) информации.

Для файлов определен маркер или указатель чтения-записи данных, который определяет текущую позицию доступа к файлу. Напомним, что с началом работы любой программы автоматически открываются стандартные потоки *stdin* и *stdout*.

В языке Си имеется большой набор функций для работы с файлами, большинство которых находятся в библиотеках ***stdio.h*** и ***io.h***. При этом потоки данных, с которыми работают функции ввода-вывода данных по умолчанию, буферизированы. Это означает, что при открытии потока с ним автоматически связывается определенный участок ОП, который и называется буфером. Все операции чтения-записи ведутся через этот буфер. Его размер фиксирован специальной константой *BUFSIZ*, которая определена в файле *stdio.h* как 512 (хотя программно ее можно изменять).

Открытие файла

Каждому файлу в программе присваивается внутреннее логическое имя, используемое в дальнейшем при обращении к нему. Логическое имя (идентификатор файла) – это указатель на файл, т.е. на область памяти, где содержится вся необходимая информация о файле.

Формат объявления указателя на файл следующий:

***FILE ****ID_указателя_на_файл;*

FILE – идентификатор структурного типа, описанный в стандартной библиотеке *stdio.h* и содержащий следующую информацию:

```
struct  FILE {
    short level;
```

– число оставшихся в буфере непрочитанных байт; обычный размер буфера – 512 байт; как только *level* = 0, в буфер из файла читается следующий блок

данных;
unsigned flags; – флаг статуса файла – чтение, запись, дополнение;
char fd; – дескриптор файла, т.е. число, определяющее его номер;
unsigned char hold; – непереданный символ, т.е. *ungetc*-символ;
short bsize; – размер внутреннего промежуточного буфера;
unsigned char buffer; – значение указателя для доступа внутри буфера; задает начало буфера, начало строки или текущее значение указателя внутри буфера в зависимости от режима буферизации;
unsigned char *curp; – текущее значение указателя для доступа внутри буфера; задает текущую позицию в буфере для обмена с программой;
unsigned istemp; – флаг временного файла;
short token; – флаг при работе с файлом;
};

Прежде чем начать работать с файлом, т.е. получить возможность чтения или записи информации в файл, его нужно открыть для доступа.

Для этого обычно используется функция

FILE* ***fopen***(char * ID_файла, char *режим);

Данная функция берет внешнее представление – физическое имя файла на носителе (дискета, винчестер) и ставит ему в соответствие логическое имя (программное имя – указатель файла).

Физическое имя, т.е. *ID* файла и путь к нему задается первым параметром – строкой, например, “*a:Mas_dat.dat*” – файл с именем *Mas_dat* и расширением *dat*, находящийся на дискете, “*d:\\work\\Sved.txt*” – файл с именем *Sved* и расширением *txt*, находящийся на винчестере в каталоге *work*.

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

При успешном открытии функция *fopen* возвращает указатель на файл (в дальнейшем – указатель файла). При ошибке возвращается *NULL*. Данная ситуация обычно возникает, когда неверно указывается путь к открываемому файлу, например, если указать путь, запрещенный для записи.

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

Возможные значения данного параметра следующие:

w – файл открывается для записи (*write*); если файла с заданным именем нет, то он будет создан; если же такой файл уже существует, то перед открытием 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+ (*w+*) – файл открывается для редактирования данных, т.е. возможны и запись, и чтение информации;

a+ – то же, что и для *a*, только запись можно выполнять в любое место файла (доступно и чтение файла);

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+*", "*rw+*", "*w+t*", "*rt+*", а также некоторые другие комбинации.

По умолчанию файл открывается в текстовом режиме.

Текстовый режим отличается от двоичного тем, что при открытии файла как текстового пара символов «перевод строки» и «возврат каретки» заменяется на один символ «перевод строки» для всех функций записи данных в файл, а для всех функций вывода – наоборот – символ «перевод строки» заменяется на два символа – «перевод строки» и «возврат каретки».

Пример открытия файла:

FILE *f; – объявляется указатель на файл *f*;

f = fopen (" d:\\work\\Dat_sp.dat ", "w"); – открывается для записи файл с логическим именем *f*, имеющий физическое имя *Dat_sp.dat* и находящийся на диске *d* в каталоге *work*, или более кратко:

FILE *f = fopen ("d:\\work\\Dat_sp.dat", "w");

Закрытие файла

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

int ***fclose*** (*указатель файла*);

Например, для предыдущего примера файл закрывается так: *fclose* (*f*);

Для закрытия нескольких файлов введена функция:

void ***fcloseall*** (*void*);

Если требуется изменить режим доступа к открытому в настоящий момент файлу, то его необходимо сначала закрыть, а затем вновь открыть с другими правами доступа. Для этого используется функция

*FILE** ***freopen*** (char *ID_файла, char *режим, FILE *указатель_файла);

которая сначала закрывает файл, заданный в третьем параметре (указатель файла), как это выполняет функция *fclose*, а затем выполняет действия, аналогичные функции *fopen*, используя указанные первый и второй параметры (открывает файл с *ID_файла* и правами доступа *режим*).

В языке Си имеется возможность работы с временными файлами, которые нужны только в процессе работы программы и должны быть удалены после выполнения некоторых вычислений. В этом случае используется функция

*FILE** ***tmpfile*** (*void*);

которая создает на диске временный файл с правами доступа *w+b*. После завершения работы программы или закрытия этого (временного) файла он автоматически удаляется.

Запись-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 по блокам.

Рассмотрим основные функции для записи-чтения данных из файлов.

Для работы с текстовыми файлами в библиотеке языка Си содержится достаточно много функций, самыми распространенными из которых являются функции

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

Формат параметров этих функций практически такой же, как и формат рассмотренных ранее (см. разд. 5.3, 5.4) функций *printf*, *scanf*, *gets* и *puts*. Так же практически совпадают и действия этих функций. Отличие состоит в том, что *printf* и другие функции работают по умолчанию с экраном монитора и клавиатурой, а функции *fprintf* и другие – с файлом, указатель которого является одним из параметров этих функций.

Рассмотрим общий пример создания текстового файла:

```
#include<stdio.h>
void main(void)
{
        FILE *f1;
        int a=2, b=3;
        if( ! (f1 = fopen("d:\\work\\f_rez.txt","w+t") ) ) {          // f1 = NULL
                puts("Open File Error!");
                return;                                               // exit(1);
        }
        fprintf(f1,"\t Файл результатов \n");
        fprintf(f1," %d плюс %d = %d\n", a, b, a+b);
        fclose(f1);
}
```

Просмотрев содержимое файла любым текстовым редактором, можно убедиться, что данные в нем располагаются точно так, как на экране, если воспользоваться функцией *printf* с такими же списками параметров.

Создание текстовых результирующих файлов обычно необходимо для оформления отчетов, различных документов, а также других текстовых материалов.

Бинарные (двоичные) файлы обычно используются для организации баз данных, состоящих, как правило, из объектов структурного типа. При чтении-записи бинарных файлов удобнее всего пользоваться функциями *fread* и *fwrite*, которые выполняют ввод-вывод данных блоками.

Такой способ обмена данными требует меньше времени.

Функция

unsigned ***fread*** (*void* ****p***, *unsigned* ***size***, *unsigned* ***n***, *FILE* ****f***);

выполняет считывание из файла *f n* блоков размером *size* байт каждый в область памяти, адрес которой *p*. В случае успеха функция возвращает количество считанных блоков. При возникновении ошибки или по достижении признака окончания файла – значение ***EOF*** (*End Of File* – признак окончания файла).

Обратное действие выполняет функция:

unsigned ***fwrite*** (*void* ****p***, *unsigned* ***size***, *unsigned* ***n***, *FILE* ****f***);

при вызове которой в файл *f* будет записано *n* блоков размером *size* байт каждый из области памяти, начиная с адреса *p*.

Позиционирование в файле

Каждый открытый файл, как уже отмечалось, имеет скрытый указатель на текущую позицию в нем. При открытии файла этот указатель устанавливается на начало данных, и все операции в файле будут производиться с данными, начинающимися в этой позиции.

При каждом выполнении функции чтения или записи указатель смещается на количество прочитанных или записанных байт, т.е. устанавливается после прочитанного или записанного блока данных в файле – это ***последовательный доступ к данным***.

В языке Си/С++ можно установить указатель на некоторую заданную позицию в файле. Для этого используют стандартную функцию ***fseek***, которая позволяет выполнить чтение или запись данных в произвольном порядке.

Декларация функции позиционирования следующая:

int ***fseek***(*FILE* ****f***, *long* ***size***, *int* ***code***);

Значение параметра *size* задает количество байт, на которое необходимо сместить указатель в файле *f*, в направлении параметра *code*, который может принимать следующие значения:

– смещение от начала файла – 0 (*SEEK_SET*);
– смещение от текущей позиции – 1 (*SEEK_CUR*);
– смещение от конца файла – 2 (*SEEK_END*).

Таким образом, смещение может быть как положительным, так и отрицательным, но нельзя выходить за пределы файла.

В случае успеха функция возвращает нулевое значение, а в случае ошибки (например, попытка выхода за пределы файла) – единицу.

Доступ к файлу с использованием функции позиционирования (*fseek*) называют ***произвольным доступом***.

Иногда нужно определить текущее положение в файле. Для этого используют функцию со следующей декларацией:

long ***ftell***(*FILE* ***f*);

которая возвращает значение указателя на текущую позицию в файле или –1 в случае ошибки.

Дополнительные файловые функции

В заключение рассмотрим наиболее распространенные функции, с помощью которых можно организовать работу с файлами:

int ***fileno*** (*FILE* ***f***) – определяет и возвращает значение дескриптора (*fd*) файла *f*, т.е. число, определяющее номер файла;

long ***filelength*** (*int* ***fd***) – возвращает длину файла, имеющего дескриптор *fd* , в байтах;

int ***chsize*** (*int* ***fd***, *long* ***pos***) – выполняет изменение размера файла, имеющего номер *fd*, признак конца файла устанавливается после байта с номером *pos*;

int ***feof*** (*FILE* ***f***) – возвращает ненулевое значение при правильной записи признака конца файла;

int ***fgetpos*** (*FILE* ***f***, *long* ***pos***) – определяет значение текущей позиции *pos* файла *f*.

Пример программы работы с файлом структур

Создать программу, в которой реализованы создание, добавление и просмотр файла, содержащего информацию о фамилии и среднем балле студентов. Процесс добавления информации заканчивается при нажатии точки.

```
#include <stdio.h>
#include <stdlib.h>
struct Sved {
        char Fam[30];
        double S_Bal;
        } zap,zapt;
char Spis[]="c:\\work\\Sp.dat";
FILE *F_zap;
FILE* Open_file(char*, char*);

void main (void)
{
    int i, j, kodR, size = sizeof(Sved), kod_read;
    while(1) {
        puts("Создать  – 1\n Добавить– 3\nПросмотреть– 2\nВыход – 0");
        scanf("%d",&kodR);
        switch(kodR) {
            case 1: case 3:
```

```
                if(kodR==1) F_zap = Open_file (Spis,"w+");
                        else F_zap = Open_file (Spis,"a+");
                while(2) {
                        puts("\n Fam (. – end) ");
                        scanf("%s",zap.Fam);
                        if((zap.Fam[0])=='.')  break;
                        puts("\n Ball: ");
                        scanf("%lf",&zap.S_Bal);
                        fwrite(&zap,size,1,F_zap);
                }
                fclose(F_zap);
                break;
                case 2:   F_zap = Open_file (Spis,"r+"); int nom=1;
                while(2) {
                        if(!fread(&zap,size, 1, F_zap)) break;
                        printf(" %2d: %20s %5.2lf\n",
                                nom++, zap.Fam, zap.S_Bal);
                }
                fclose(F_zap);
                break;
                case 0:   return;               // exit(0);
        }                                       // Закрывает switch()
    }                                           // Закрывает while()
}
// Функция обработки ошибочной ситуации при открытии файла
FILE* Open_file(char *file, char *kod)
{
    FILE *f;
    if(!(f = fopen(file, kod))) {
        puts("Open File Error!");
        exit(1);
    }
    return f;
}
```

Советы по программированию

При выполнении вариантов заданий придерживайтесь следующих ключевых моментов.

1. Объекты типов структуры и объединения применяются для логически связанных между собой данных различных типов.

2. После описания шаблона структурного типа данных ставится точка с запятой.

3. Элементы данных, входящие в структуры и объединения, называются полями. Поля могут быть любого базового (стандартного) типа данных, массивом, указателем, объединением или структурой.

4. Для обращения к полю используется операция принадлежности (привязки, выбора) «.» (точка) при обращении через *ID* структуры, или «–>» (стрелка) при обращении через указатель.

5. Структуры одного типа можно присваивать друг другу с использованием стандартной функции *memcpy*.

6. Ввод-вывод структур выполняется поэлементно.

7. Структуры, память под которые выделяет компилятор, можно инициализировать значениями их полей.

8. Файл – это именованный объект, хранящий данные на каком-либо носителе, хотя может располагаться и на электронном диске в ОП.

9. Файл не имеет фиксированной длины, т.е. может увеличиваться или уменьшаться в процессе обработки.

10. Перед работой файл необходимо открыть (функция *fopen*), а после работы закрыть (функция *fclose*).

ЗАДАНИЕ 6. Создание и обработка структур

Первый уровень сложности

Написать программу по обработке массива структур, содержащего следующую информацию о студентах:

– фамилия и инициалы;

– год рождения;

– номер группы;

– оценки за семестр: физика, математика, информатика, химия;

– средний балл.

Организовать ввод исходных данных, средний балл рассчитать по введенным оценкам.

1. Распечатать анкетные данные студентов, сдавших сессию на 8, 9 и 10.

2. Распечатать анкетные данные студентов-отличников, фамилии которых начинаются с интересующей вас буквы.

3. Распечатать анкетные данные студентов-отличников из интересующей вас группы.

4. Распечатать анкетные данные студентов, фамилии которых начинаются с буквы *А* и сдавших математику на 9 и 10.

5. Распечатать анкетные данные студентов интересующей вас группы, имеющих оценку 9 по физике и оценку 10 по высшей математике.

6. Распечатать анкетные данные студентов интересующей вас группы. Фамилии студентов начинаются с букв *В*, *Г* и *Д*.

7. Распечатать анкетные данные студентов, не имеющих оценок 4 и 5 по информатике и математике.

8. Вычислить общий средний балл всех студентов и распечатать список студентов со средним баллом выше общего среднего балла.

9. Вычислить общий средний балл всех студентов и распечатать список студентов интересующей вас группы, имеющих средний балл выше общего среднего балла.

10. Распечатать анкетные данные студентов интересующей вас группы, имеющих оценки 3 и 4.

11. Распечатать анкетные данные студентов интересующей вас группы, имеющих оценку 9 по информатике.

12. Распечатать анкетные данные студентов, имеющих оценку 8 по физике и оценку 9 по высшей математике.

13. Вычислить общий средний балл студентов интересующей вас группы и распечатать список студентов этой группы, имеющих средний балл выше общего среднего.

14. Распечатать анкетные данные студентов-отличников интересующей вас группы.

15. Распечатать анкетные данные студентов интересующей вас группы, имеющих средний балл выше введенного с клавиатуры.

Второй уровень сложности

Написать программу предыдущего варианта, создав из предложенных анкетных данных динамический массив введенной с клавиатуры размерности. Полученные данные упорядочить: для символьных данных – по алфавиту (выбрав нужное поле), для числовых данных – по возрастанию (убыванию).

ЗАДАНИЕ 7. Создание и обработка файлов

Первый уровень сложности

Написать программу по обработке файла, состоящего из структур, содержащих информацию задания 6. Средний балл рассчитать программно по введенным оценкам. Массив структур не использовать.

В программе реализовать следующие действия по обработке файла:

– создание;

– просмотр;

– добавление нового элемента;

– удаление (редактирование);

– решение индивидуального задания (первый уровень сложности задания 6).

Результаты выполнения индивидуального задания записать в текстовый файл.

Второй уровень сложности

Задачи шифровки. Составить программу, которая вводит строку с клавиатуры; признак окончания ввода – нажатие клавиши *Enter*, шифрует введенный текст в файл на диске по определенному алгоритму. Программа должна считывать эту строку из файла и далее дешифровать текст, выводя его на экран и записывая в выходной файл.

В программе реализовать следующие действия:

– ввод с клавиатуры исходной строки текста и запись в файл *a.txt*;

– считывание строки из файла и вывод на экран;

– шифровка текста;

– расшифровка.

Алгоритмы шифровки:

1. Каждая буква от «а» до «ю» заменяется на следующую по алфавиту, а «я» заменяется на «а».

2. Первая буква «а» заменяется на 11-ю, вторая «б» – на 12-ю, третья – на 13-ю, ... , последняя «я» – на 10-ю.

3. После каждой согласной буквы вставляется буква «а».

4. После каждой согласной буквы вставляется слог «ла».

5. Каждая пара букв «ле» заменяется на «ю», «са» – на «щ», «ик» – на «ж».

6. Каждая из пары букв «си», «ли» и «ти» заменяются соответственно на «иис», «иил» и «иит».

7. После каждой гласной буквы вставляется буква «с».

8. После каждой гласной буквы вставляется слог «ла».

9. Каждая из букв «а», «о», «и» заменяется соответственно на «ц», «ш», «щ».

10. Каждая буква заменяется на следующую в алфавите по часовой стрелке.

11. Каждая буква заменяется на следующую в алфавите против часовой стрелки.

12. Каждая буква «а» заменяется на слог «си», а «и» – на «са».

13. Четные и нечетные символы меняются местами.

14. Символы, кратные двум по порядку следования, заменяются на единицы.

15. Символы, кратные двум по порядку следования, заменяются на свой порядковый номер.