2. Теоретические сведения

Структуры данных

Понятие структуры

Очень часто при обработке информации приходится работать с блоками данных, в которых присутствуют разные типы данных. Например, информация о книге в каталоге библиотеки включает в себя автора, название книги, год издания, количество страниц и т.п.

Для хранения этой информации в памяти не подходит обычный массив, так как в массиве все элементы должны быть одного типа. Конечно, можно использовать несколько массивов разных типов, но это не совсем удобно.

В современных языках программирования существует особый тип данных, который может включать в себя несколько элементов более простых (причем разных!) типов.

Структура - это тип данных, который может включать в себя несколько **полей** – элементов разных типов (в том числе и другие структуры).

В общем случае при работе со структурами следует выделить четыре момента:

- объявление и определение типа структуры,
- объявление структурной переменной,
- инициализация структурной переменной,
- использование структурной переменной.

Данные типа структура, как и массивы, относятся к сложным структурам данных. Структура состоит из фиксированного числа элементов, называемых полями. Однако, в отличие от массива, поля могут быть различного типа. Например, структурой можно считать строку экзаменационной ведомости:

Андреева С.В. 4 5 5

Данная структура состоит из четырех полей: одно поле - строка (ФИО студента) и три числовых поля (оценки студента по предметам).

Поскольку структура - это новый тип данных, его надо предварительно объявить. Определение типа структуры делается так:

```
struct Имя
    {
      <тип> <имя 1-го поля>;
      <тип> <имя 2-го поля>;
      ……
      <тип> <имя последнего поля>;
    };
```

Например, задание типа записи строки экзаменационной ведомости выглядит следующим образом:

```
struct student
{
     char  fam[20];
     int mathematics, informatics, history;
 };
```

Тогда при описании переменных можно использовать этот тип:

```
struct student   varStud;
```

Здесь varStud - переменная типа структура; struct student - тип; fam, mathematics, informatics, history - поля структуры.

Отметим, что определение типа структуры может быть задано в программе на внешнем уровне, при этом имя пользовательского типа имеет глобальную видимость (при этом память не выделяется). Определение типа структуры также может быть сделано внутри функции, тогда имя типа структуры имеет локальную видимость.

В более поздних версиях языка С ключевое слово typedef позволяет в программе создать синоним типа, который удобно использовать для объявления переменных структурного типа. Например:

```
typedef struct student
{
   char  fam[20];
   int mathematics, informatics, history;
} STUD;
```

Идентификатор STUD представляет собой синоним типа struct student. С помощью синонима STUD можно объявить переменную:

```
STUD varStud;
```

Для обращения к отдельным полям переменной типа структура используется составное имя:

<имя переменной>.<имя поля>

Например, для переменной varStud обращения к полям записываются следующим образом:

```
varStud.fam, varStud. mathematics, varStud. informatics,
varStud. history.
```

При размещении в памяти структурной переменной можно выполнить ее инициализацию. Неявная инициализация производится для глобальных переменных, переменных класса static. Структурную переменную можно инициализировать явно при объявлении, формируя список инициализации в виде константных выражений.

Формат: struct Имя переменная ={значение1, … значениеN};

Внутри фигурных скобок указываются значения полей структуры, например:

```
struct student varStud={"Андреева С.В.", 4, 5, 5};
```

при этом первое значение записывается в первое поле, второе значение – во второе поле и т. д., а сами значения должны иметь тип, совместимый с типом поля.

Обработка структур

Над структурами возможны следующие операции:

- присваивание значений структурной переменной;
- получение адреса переменной с помощью операции &;
- ввод и вывод значений переменных структурного типа;
- сравнение полей переменных структурного типа.

Операция присваивания применима, как к отдельным полям переменной структурного типа, так и к переменным в целом.

При присваивании полям структуры значений, необходимо учитывать типы полей. Например:

```
#include <stdio.h>
#include <string.h>
struct student  // описание структуры
{
char  fam[20];
int mathematics, informatics, history;
} ;
   main()
  { student varStud;           //описание переменной структурного типа
strcpy(varStud.fam, "Андреева С.В. "); /*копирование фамилии в поле fam
переменной X */
    varStud. mathematics=4;
    varStud. informatics=5;
    varStud. history=5;
    printf("\n %s  %d   %d   %d", varStud.fam, varStud.mathematics,
varStud.informatics,varStud.history);/*вывод информации из полей
переменной varStud*/
  // .  .  .
}
```

Для структурного типа возможно присваивание значений одной структурной переменной другой структурной переменной, при этом обе переменные должны иметь один и тот же тип.

Работа со структурной переменной обычно сводится к работе с отдельными полями структуры. Такие операции, как ввод с клавиатуры, сравнение полей и вывод на экран применимы только к отдельным полям. Например, в выше приведенном примере вывод информации о студенте осуществляется выводом значений отдельных полей с помощью функции printf().

С помощью структурного типа можно формировать массивы записей. Так, например информацию о 20 студентах можно хранить в массиве из 20 элементов структурного типа:

```
struct student
{
    char  fam[20];
   int mathematics, informatics, history;
};
main()
{ student Spis[20];
  .  .  .
}
```

Пример задачи с использованием структурированных данных

Рассмотрим пример программы, в которой вводится информация об абонентах сети: ФИО, телефон и возраст. В программе выбираются абоненты моложе 25 лет и их список выводится в алфавитном порядке.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <windows.h>
struct abonent  //описание структуры
{ char f[10],i[10],o[10];
  long tel;
  int voz;
};
int main() {
 SetConsoleCP(1251);
 SetConsoleOutputCP(1251);

const int n=5;
abonent z[n],y[n];    //описание массивов структур

for (int i=0; i<n; i++)//ввод в цикле исходной информации о пяти
абонентах
{printf("Введите ФИО абонента:");
 scanf("%s%s%s",z[i].f, z[i].i, z[i].o);
 printf("введите его телефон и возраст:");
 scanf("%ld%d",&z[i].tel,&z[i].voz);
}
printf("-----------------------------------------------------\n");
printf("| Фамилия |  Имя   | Отчество| Телефон |  Возраст |\n");
printf("-----------------------------------------------------\n");

for (int i=0;i<n;i++) //вывод в цикле информации о пяти абонентах
 printf("|%9s|%8s|%9s|%7ld   |   %5d    |\n", z[i].f,z[i].i,z[i].o,
z[i].tel,z[i].voz);

printf("-----------------------------------------------------\n");
int k=0;abonent x;
for (int i=0;i<n;i++)
  {if(z[i].voz<25) // поиск абонента  моложе 25 лет
   y[k++]=z[i];
  }
for(int i=1;i<k;i++)  //сортировка списка абонентов моложе 25 лет
```

```
    for(int j=k-1;j>=i;j--)
     if(y[j].f[0]<y[j-1].f[0])
      {x=y[j];
      y[j]=y[j-1];
      y[j-1]=x;}
    printf("mologe 25\n");
printf("-----------------------------------------------------\n");
printf("| Фамилия |  Имя   | Отчество| Телефон |  Возраст |\n");
printf("-----------------------------------------------------\n");
for (int i=0;i<k;i++) // вывод отсортированного списка
 {printf("|%9s|%8s|%9s|%7ld  |  %5d   |\n", y[i].f,y[i].i, y[i].o,
y[i].tel,y[i].voz);
 }
printf("-----------------------------------------------------\n");
return 0;
   }
```

Полем структурной переменной может быть переменная любого типа, в том числе другая структурная переменная. Поле, представляющее собой структуру, называется вложенной структурой.

2.15. Файлы данных

Понятие файла

В большинстве своем *файлы* представляют собой именованные области внешней (дисковой) памяти, с которыми программы могут обмениваться информацией. Необходимость в таких обменах, во-первых, возникает, когда объем оперативной памяти недостаточен для хранения нужной информации. Во-вторых, программа может воспользоваться данными, полученными ранее другой программой и предусмотрительно записанными на диск.

Файл – это информация, размещенная на каком-либо носителе (диске) или в буфере ввода/вывода устройства (клавиатура). Файлы предназначены только для хранения информации, а обработка этой информации осуществляется программами.

Для обмена данными файл должен быть открыт, по завершении этого процесса – закрыт.

Поток – это логический канал, предназначенный для выполнения операций ввода/вывода. Каждому файлу при его открытии ставится в соответствие поток.

В языке Си существуют стандартные потоки:

stdin – стандартный консольный ввод (клавиатура по умолчанию);
stdout – стандартный консольный вывод (монитор по умолчанию);

Стандартные потоки открываются при каждом запуске программы.

Работа с файлами

Для работы с файлами в программах на С используется заголовочный файл stdio.h, в котором объявлен специальный тип данных – структура FILE, предназначенная для хранения атрибутов (параметров) файлов (указатель

текущей позиции файла, признак конца файла, флаги индикации ошибок, сведения о буферизации и др.).

Поля структуры типа FILE доступны с помощью специальных С-функций. Для организации работы с файлом используется определенная последовательность действий.

Объявление переменной-указателя на структуру типа FILE, в которой будут храниться атрибуты файла

```
FILE *fl;
```

где *fl – указатель на файл.

Открытие файла

```
fl=fopen("путь к файлу","режим работы файла");
```

Параметр "путь к файлу" указывает размещение файла на диске. Он обязательно содержит имя файла и может содержать имя логического диска и путь к нему по папкам.

Параметр "режим работы файла" показывает, как будет использоваться файл:

"w" – для записи данных (вывод);
"r" – для чтения данных (ввод);
"a" – для добавления данных к существующим записям.
Примеры открытия файлов:

```
FILE *fin,*out;
fin= fopen("My_file1","r");
out=fopen("My_file2","w");
```

Функция fopen() возвращает значение указателя на структуру типа файл. Если файл по каким-либо причинам не открывается, функция fopen() возвращает значение NULL.
Рассмотрим особенности режимов открытия файлов. Если файл открывается в режиме записи данных "w", то указатель текущей позиции устанавливается на начало файла. Если указанный в функции fopen() файл не существует, то он создается. Необходимо помнить, что открытие существующего файла в режиме "w" приводит к уничтожению его старого содержания.

Открытие файла для чтения в режиме "r" возможно только для созданного ранее файла, при этом указатель текущей позиции устанавливается на начало файла. Если открываемый на чтение файл не существует, функция fopen() возвращает пустой указатель со значением NULL.

Если файл открывается в режиме добавления данных "a", то указатель текущей позиции устанавливается на конец файла. Данные, ранее помещенные в файл, остаются без изменений. Если указывается несуществующий файл, то он создается заново.

В С файл можно открыть для чтения и/или записи в текстовом или бинарном (двоичном) режиме. Поэтому можно указать дополнительные условия режима открытия файла:

"b" – двоичный поток;
"t" – текстовый поток;
"+" – обновление файла.
Пример:
"r+" – чтение файла с обновлением, т. е. возможна перезапись данных с усечением;
"w+" – запись в файл и одновременно чтение;
"a+" – добавление данных и чтение.
Для поочередного выполнения чтения и записи в режиме "+" необходимо ручное позиционирование курсора.

Обработка открытого файла

Каким образом можно прочитать уже открытый файл или записать в него информацию? Для этого в языке С существуют специальные функции:

Чтение (ввод)	Запись (вывод)
fgetc()	fputc()
fscanf()	fprintf()
fgets()	fputs()
fread()	fwrite()

При каждой операции ввода/вывода указатель текущей позиции файла смещается на одну позицию в сторону конца файла.

Проверка признака конца файла

Так как при каждой операции ввода/вывода происходит перемещение указателя текущей позиции в файле, в какой-то момент указатель достигает конца файла. Структура типа FILE имеет поле – индикатор конца файла. Функция feof() проверяет состояние индикатора конца файла и возвращает значение 0, если конец файла не был достигнут, или значение, отличное от нуля, если был достигнут конец файла. Функция имеет единственный аргумент – указатель на FILE. Вызов функции в команде if:

```
if (! feof(fin))…
```

проверяет, что конец файла еще не достигнут.

Закрытие файла

После завершения обработки файла его следует закрыть с помощью функции fclose(). При этом разрывается связь указателя на файл с внешним набором данных. Освободившийся указатель можно использовать для другого файла. Формат вызова функции:

```
fclose(fin);
```

При нормальном завершении программы в большинстве операционных систем все открытые файлы закрываются автоматически, но рекомендуется закрывать все файлы, дальнейшая обработка которых в программе не предполагается, при помощи функции fclose().

Функции ввода/вывода

Простейший способ выполнить чтение из файла или запись в файл – использовать функции fgetc() или fputc().

Функция getc() выбирает из файла очередной символ; ей нужно только знать указатель на файл, например,

```
char Symb=fgetc(fin);
```

Если при обработке достигается конец файла, то функция getc() возвращает значение EOF(end of file).

Функция putc() заносит значение символа Symb в файл, на который указывает out. Формат вызова функции:

```
fputc(Symb,out);
```

Пример 1. Текст из файла my_char.txt выводится на экран. Если файл не найден, на экран выводится сообщение "File not found!":

```
#include <stdio.h>
int main()
{
FILE *ptr;      //описание указателя на файл
unsigned char ch; //открытие файла для чтения
if ((ptr=fopen("my_char.txt","r"))!=NULL)
{
ch=fgetc(ptr);   //чтение первого  символа из файла
while (!feof(ptr))   //цикл до конца файла
{
printf("%c",ch);       //вывод символа, взятого из файла
ch=fgetc(ptr);          //чтение  следующего символа  из файла
}
fclose(ptr);          //закрытие файла
}
else printf("\nFile not found!");
return 0;}
```

В этом примере для чтения файла используется указатель ptr. При открытии файла производится проверка. Если переменной ptr присвоено значение NULL, то файл не найден; на экран выводится соответствующее сообщение, и программа завершается. Если ptr получил ненулевое значение, то файл открыт. Далее выполняется чтение символов из файла до тех пор, пока не будет достигнут конец файла (!feof(ptr)). Прочитанный символ помещается в переменную ch, а затем выводится на экран.

Пример 2. Записать в файл буквы, вводимые с клавиатуры. Ввод продолжается до нажатия клавиши F6 или CTRL/z на новой строке (ввод символа EOF – конца файла):

```
#include <stdio.h>
int main(void)
```

```
{
char c;
FILE *out;  // описание указателя на файл
out=fopen("Liter.txt","w");  //открытие файла для записи
while (  (c=getchar( ))!=EOF)  //пока не будет введен символ конца
fputc(c,out);  //  запись введенного символа в файл
fclose(out);   //закрытие файла
return 0;
}
```

Функции fscanf() и fprintf() выполняют форматированный ввод/вывод.
Чтение из файла выполняет функция fscanf():

```
fscanf(fin,["строка формата"],[список адресов переменных]);
```

Функция возвращает количество введенных из файла значений или EOF.

Запись в файл осуществляет функция fprintf():

```
fprintf(out,["строка формата"],[список переменных, констант]);
```

Возвращает количество выведенных в файл байт (символов) или EOF.

Строка формата функций fscanf() и fprintf() формируется так же, как было описано ранее в главе, посвященной консольному вводу/выводу и функциям printf() и scanf().

Следует заметить, что вызов функции

```
fscanf(stdin,[строка формата],[список адресов переменных]);
```

эквивалентен вызову

```
scanf([строка формата],[список адресов переменных]);
```

Аналогично,

```
fprintf(stdout, [строка формата],[список переменных, констант]);
```

эквивалентно

```
printf([строка формата],[список переменных, констант]);
```

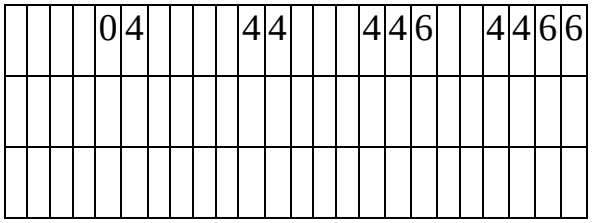
Рассмотрим примеры программ, использующих эти функции.

Пример 3. В программе создается массив, состоящий из четырех целых чисел. Вывести массив в файл:

```
#include <stdio.h>
int main()
{const int n=4;
int array[]={4,44,446,4466}; //описание и инициализация массива
FILE *out;                   //описание указателя на файл
out=fopen("num_arr.txt","w"); //открытие файла для записи
for(int i=0;i<n;i++)
fprintf(out,"%6d",array[i]);  //запись в файл элемента массива
```

```
fclose(out);                          //закрытие файла
return 0;
}
```

В результате выполнения программы в файл num_arr.txt будет помещена следующая информация:



Пример 4. Имеется файл данных, содержащий целые числа, разделенные пробелами. Количество чисел в файле неизвестно. Требуется найти среднее арифметическое значение этих чисел:

```
#include <stdio.h>
int main()
{
int S=0, count=0, numb;   //описание переменных
FILE *in;                 //описание укателя на файл
if((in=fopen("num_arr.txt","r"))!=NULL)/*открытие файла для чтения*/
{
while (!feof(in))         //пока не конец файла
{
fscanf(in,"%d",&numb);    //читать из файла число в переменную numb
S+=numb;                  // добавить numb  в сумму
count++;                  //увеличиваем счетчик на 1  количество
printf("%d\n", numb);     //выводим значение numb на экран
}
double aver=(double)S/count;   //считаем среднее значение
printf("Average=%lf\n", aver); //вывод среднего значения
fclose(in);                    //закрыть файл
}
else
printf("\nФайл не найден!");
return 0;
}
```

Чтение чисел из файла выполняется в переменную numb до тех пор, пока не будет достигнут конец файла. Одновременно ведется подсчет количества прочитанных чисел в переменной count и накопление суммы прочитанных чисел в переменной S. Переменные S и count целые, поэтому для правильного вычисления среднего арифметического, необходимо выполнить преобразование одной из этих переменных в формат double.

Рассмотрим другие библиотечные функции, используемые для работы с файлами (все они описаны в файле stdio.h):

1. Функция fputs( ) записывает строку символов в файл. Она отличается от функции puts( ) тем, что в качестве второго параметра должен быть записан указатель на переменную файлового типа.

Например:

```
fputs("Example", fp);
```

При возникновении ошибки возвращается значение EOF.

2. Функция fgets( ) читает строку символов из файла. Она отличается от функции gets( ) тем, что в качестве второго параметра должно быть указано максимальное число вводимых символов плюс единица, а в качестве третьего - указатель на переменную файлового типа. Строка считывается целиком, если ее длина не превышает указанного числа символов, в противном случае функция возвращает только заданное число символов.

Рассмотрим пример:

```
fgets(string, n, fp);
```

Функция возвращает указатель на строку string при успешном завершении и константу NULL в случае ошибки либо достижения конца файла.

3. Функция fread( ) предназначена для чтения блоков данных из потока. Имеет прототип:

```
unsigned fread(void *ptr,unsigned size,unsigned n,FILE *fp);
```

Она читает n элементов данных, длиной size байт каждый, из заданного входного потока fp в блок, на который указывает указатель ptr. Общее число прочитанных байтов равно произведению n*size. При успешном завершении функция fread( ) возвращает число прочитанных элементов данных, при ошибке - 0.

4. Функция fwrite( ) предназначена для записи в файл блоков данных. Имеет прототип:

```
unsigned fwrite(void *ptr,unsigned size,unsigned n, FILE *fp);
```

Она добавляет n элементов данных, длиной size байт каждый, в заданный выходной файл fp. Данные записываются с позиции, на которую указывает указатель ptr. При успешном завершении операции функция fwrite( ) возвращает число записанных элементов данных, при ошибке - неверное число элементов данных.

5. Функция fseek( ) позволяет выполнять чтение и запись с произвольным доступом и имеет следующий прототип:

```
int fseek(FILE *fp, long count, int access);
```

Здесь fp - указатель на файл, возвращенный функцией fopen( ), count - номер байта относительно заданной начальной позиции, начиная с которого будет выполняться операция, access - способ задания начальной позиции.

Переменная access может принимать следующие значения:

0 - начальная позиция задана в начале файла;
1 - начальная позиция считается текущей;
2 - начальная позиция задана в конце файла.

При успешном завершении возвращается нуль, при ошибке - ненулевое значение.

6. Функция ferror( ) позволяет проверить правильность выполнения последней операции при работе с файлами. Имеет следующий прототип:

```
int ferror(FILE *fp);
```

В случае ошибки возвращается ненулевое значение, в противном случае возвращается нуль.

7. Функция remove( ) удаляет файл и имеет следующий прототип:

```
int remove(char *file_name);
```

Здесь file_name - указатель на строку со спецификацией файла. При успешном завершении возвращается нуль, в противном случае возвращается ненулевое значение.

8. Функция rewind( ) устанавливает указатель текущей позиции в начало файла и имеет следующий прототип:

```
void rewind(FILE *fp);
```

Работа с текстовыми файлами

Файлы бывают текстовые (в которых можно записывать только буквы, цифры, скобки и т.п.) и двоичные (в которых могут храниться любые символы из таблицы). В текстовых файлах не употребляются первые 31 символ кодовой таблицы ASCII (управляющие), а символы конца строки 0x13 (возврат каретки, CR) и 0x10 (перевод строки LF) преобразуются при вводе в одиночный символ перевода строки \n (при выводе выполняется обратное преобразование). Эти символы добавляются в конце каждой строки, записываемой в текстовый файл. При обнаружении в текстовом файле символа с кодом 26 (0x26), т.е. признака конца файла, чтение файла в текстовом режиме заканчивается, хотя файл может иметь продолжение.

Создать текстовый файл можно с помощью текстового редактора и с помощью программы.

Обработка бинарных файлов

Если файл открыт в бинарном режиме, его можно записывать или считывать побайтно. Функция fseek() позволяет обращаться с бинарным файлом как с массивом и переходить к любой позиции в файле, обеспечивая возможность произвольного доступа. Если текстовые файлы являются файлами

с последовательным доступом, то к бинарным файлам может применяться произвольный доступ.

Составим программу создания нового файла с информацией о городах: код, название, численность жителей.

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>
struct city
    { int kod;
       char name[10];
       long  c;
} town;

int main(){
 SetConsoleCP(1251);
 SetConsoleOutputCP(1251);
char c;
  FILE *f;

 f=fopen("file1.dat","wb"); //открытие бинарного файла для записи
  printf("\n Ввод информации о городе ");
  do
 { printf("\nКод: "); scanf("%d", &town.kod);
   printf("\nназвание: "); scanf("%s", town.name);
   printf("\nколичество жителей: "); scanf("%ld", &town.c);
   fwrite(&town, sizeof(town), 1, f); //запись в файл одной
структуры t
   printf("\n END Закончить?  y/n  ");

 }  while (getch() != 'y');
  fclose(f);
}
```

Выполнение этой программы приведет к созданию бинарного файла с информацией о городах.

Рассмотрим еще одну программу, которая будет читать из файла информацию о городах и выводить на экран список городов, количество жителей в которых превышает миллион.

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>
struct city
    { int kod;
       char name[10];
       long  c;
};

int main(){
 SetConsoleCP(1251);
 SetConsoleOutputCP(1251);
city t;
  FILE *f;
```

```
f=fopen("file1.dat","rb"); //открытие бинарного файла для чтения
 fread(&t, sizeof(t), 1, f); //чтение из файла одной структуры t
 while (!feof(f))
 { if(t.c>1000000)
  printf("\n%3d название: %10s   количество жителей: %ld",t.kod,
t.name, t.c);
  fread(&t, sizeof(t), 1, f);
 }
  fclose(f);
}
```