Казанский Федеральный Университет

Журавлёв А.А., Мамедова Л.Э., Стенин Ю.М.,
Хуторова О.Г., Фахртдинов Р.Х.

ПРАКТИКУМ
ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ СИ
ЧАСТЬ 2

Казань 2011

СОДЕРЖАНИЕ

1. РАБОТА СО СТРОКАМИ

В языке Си не существует встроенного строкового типа данных. Текстовая строка представляется в виде одномерного символьного массива (тип `char`), заканчивающегося нулевым байтом.

Пример непосредственного посимвольного ввода строки:

```
char str[15];
str[0] = 'H';
str[1] = 'e';
str[2] = 'l';
str[3] = 'l';
str[4] = 'o';
str[5]= '!';
str[6] = 0;
```

В массиве `str` записана строка `"Hello!"`. При этом используются только 7 из 15 элементов массива.

Обычно для записи строковых констант в программе используются литералы. Литерал – последовательность символов, заключенная в двойные кавычки. Примеры:

```
"Это строка"
"0123456789"
"*"
```

Заметим, что символ, заключенный в двойные кавычки, отличается от символа, заключенного в апострофы. Литерал `"*"` обозначает два байта: первый байт содержит символ звездочки, второй байт содержит ноль. Константа `'*'` обозначает один байт, содержащий знак звездочки.

Строку приведённого выше примера обычно записывают с помощью литерала:

```
char str[15]= "Hello!";
```

Инициализировать строку можно и так:

```
char digits[] = "0123456789ABCDEF";
```

Размер строки (одномерного массива) можно явно не задавать, он определяется согласно инициализирующему литералу, в данном случае 17 (16 символов плюс нулевой байт).

При работе со строками часто используется связь между массивами и указателями. Поскольку литерал, по сути, – массив байтов, то его значение есть адрес первого байта (указатель на начало строки), и его можно присвоить указателю на char:

```
char *str = "Message for you";
```

В следующем примере функция `CopyString` копирует первую строку во вторую:

```
#include<stdio.h>
#include<conio.h>
void CopyString(char src[], char dst[])
{  int i=0;
   while(src[i]!='\0')
      {dst[i]=src[i];i++;};
   dst[i]='\0';
}
void main()
{
   char frst[]="Hello, I love you!",scnd[80];
   CopyString(frst,scnd);
   puts(scnd);
   getch();
}
```

Копирование здесь осуществляется как перепись элементов одного массива в другой с добавлением в конце символа окончания строки. Та же функция, явно использующая указатели, выглядит следующим образом:

```
void CopyString(char*src, char*dst)
{
     while(*dst++=*src++);
     *dst = 0;
}
```

Здесь используется свойство указателя по операции ++ автоматически смещаться в массиве на величину, равную размеру элемента массива. Присвоения *dst=0 и *dst='\0' эквивалентны.

Для вывода строки на дисплей можно использовать функцию puts(), а для ввода строки с клавиатуры – функцию gets() (входят в состав заголовочного файла <stdio.h>).

Еще один пример: нужно подсчитать количество цифр в строке str:

```
#include <ctype.h>
#include<stdio.h>
void main()
{
   int count = 0;
   char*str;
   puts("Enter string with digits:");
 gets(str);
 while (*str != 0) {
    /*признак конца строки – ноль*/
     if (isdigit(*str++)) count++;
    /*проверить байт, на который
     указывает str, и сдвинуть
    указатель на следующий байт */
 printf("number of digits=%d\n",count);
}
```

При выходе из цикла while переменная count содержит количество цифр в строке str, а сам указатель str указывает на конец строки – нулевой байт. Чтобы проверить, является ли текущий символ цифрой, используется функция isdigit, входящая в состав заголовочного файла <ctype.h>. Это одна из многих стандартных функций языка, предназначенных для работы с символами и строками.

С помощью функций, входящих в состав заголовочных файлов <string.h> и <ctype.h>, реализованы многие часто используемые операции над символьными строками. В большинстве своем в качестве строк они воспринимают указатели. Приведем ряд наиболее употребительных функций.

```
char*strcpy(char*target, char*source);
```

Копирует строку `source` по адресу `target`. Функция предполагает, что памяти, выделенной по адресу `target`, достаточно для копируемой строки. В качестве результата функция возвращает адрес первой строки.

```
char*strcat(char*target, char*source);
```

Присоединяет вторую строку с конца первой. На место завершающего нулевого байта первой строки переписывается первый символ второй строки. В результате, по адресу `target` получается строка, образованная слиянием первой со второй. В качестве результата функция возвращает адрес первой строки.

```
int strcmp(char*string1, char*string2);
```

Сравнивает две строки в лексикографическом порядке (по алфавиту). Если первая строка должна стоять по алфавиту раньше, чем вторая, то результат функции меньше нуля, если позже – больше нуля, и ноль, если две строки равны.

```
strlen(char*string);
```

Определяет длину строки в байтах, не считая завершающего нулевого байта.

В следующем примере, использующем приведенные функции, в массиве `target` будет образована строка `"22-го июня, ровно в 4 часа"`:

```
char target[80];
char *date = "22 июня";
char *time = "ровно в 4 часа";
strcpy(target, date);
strcat(target, ", ");
strcat(target, time);
```

Как видно из этого примера, литералы можно непосредственно использовать в выражениях.

Поскольку символы строки как элементы массива располагаются в памяти по возрастающим адресам, это позволяет обращаться к строке не только целиком, но и начиная с любого её символа, для чего можно использовать, например, оператор ссылки &. Следующий пример иллюстрирует эту возможность, как и различие

между обращением к строке по ссылке, начиная с третьего символа, и обращением к этому символу как элементу массива:

```
#include<stdio.h>
void main()
{
 char str[]="Mary has a little lamb";
/*или char *str="Mary has a little lamb";*/
printf("stroka:\"%s\",simvol:'%c'\n",&str[2],
str[2]);
}
```

Результат выполнения этой программы:

```
stroka:"ry has a little lamb", simvol:'r'
```

Определить массив строк можно с помощью двумерного массива:

```
char StrMassiv[][20]={ "John","Paul","Ringo",
                       "George"};
```

или через массив указателей:

```
  char *StrArray[4]={ "one", "two", "three",
"four"};
```

1.1. Задания на работу со строками

Дана строка длиной n символов, содержащая слова, т.е. группы символов, разделенные пробелами и другими разделителями (знаками препинания) и не содержащие пробелов внутри себя:

1. Вывести все слова строки в алфавитном порядке.
2. Вывести все слова, отличные от последнего слова, предварительно преобразовав их по следующему правилу: удалить из каждого слова все последующие вхождения первой его буквы.
3. Найти и вывести самое короткое слово в строке и егодлину.
4. Вывести те слова, которые отличаются от последнего слова и удовлетворяют условию, что в слове нет повторяющихся букв.

5. Удалить все символы в строке, не являющиеся буквами, а также заменить множественные пробелы одним.
6. Выделить в строке отдельные слова и вывести их «в столбик» по одному слову.
7. Найти слова, первый и последний символы которых совпадают, и вывести эти слова и их количество.
8. Дан текст, в котором могут встречаться знаки арифметических операций `'+'`, `'–'`, `'·'`, `'/'`. Если с обеих сторон от знака расположены буквы, то каждый знак `'+'`, заменить цифрой `'1'`, знак `'–'` заменить цифрой `'2'`, знак `'·'` заменить цифрой `'3'`, а знак `'/'` – цифрой `'4'`. Иначе оставить текст без изменений.
9. Найти число слов, которые оканчиваются той же буквой, что и последнее слово. Вывести их на экран.
10. Дана последовательность английских слов. В словах, которые оканчиваются сочетанием букв "*ing*", заменить это окончание на "*ed*".
11. Вывести все слова, отличные от последнего слова, предварительно удалив из слов нечетной длины среднюю букву.
12. Дано натуральное число n ($-1000 \le n \le 1000$). Написать программу вывода такого числа в словесной форме («два», «семнадцать», «минус сорок пять», «сто двадцать два» и т.д.).
13. Для каждого слова указать, сколько раз оно встречается среди всех слов данного текста.
14. Известно, что в начале строки находится группа не более чем из 40 латинских букв, за которой следуют пробелы. Вывести эту строку, предварительно удалив все вхождения сочетания "*th*" (после удаления текст сомкнуть).
15. Заменить все малые буквы в русских словах одноименными большими.
16. Найти и вывести все слова, начинающиеся на букву *'а'*.
17. Подсчитать количество букв *'а'* в каждом четном слове.
18. Строка символов содержит, кроме букв и знаков, действительные числа (например, "*а*·*в*+0,973–1,1"). Получить строку, в которой в

таких числах после запятой оставлены только две значащие цифры (т.е. "*а*·*в*+0,97–1,10")

19. Строка содержит два действительных числа. Получить сумму этих чисел и записать в другую строковую переменную в виде выражения (например, "27,8 + 17,3 = 45,1").
20. Вывести те слова строки, которые отличны от последнего слова и совпадают с начальным отрезком латинского алфавита (*a, ab, abc* и т. д.). В каждом слове от 1 до 8 литер.

2. СТРУКТУРЫ

Структура в языке Си – это совокупность логически связанных переменных, возможно, различных типов, сгруппированных под одним именем для удобства дальнейшей обработки.

Пример описания структуры из пяти полей для хранения сведений об успеваемости студентов:

```
struct  Zurnal    /* Zurnal – имя нового типа*/
{                 /* начало блока структуры*/
 int number;      /* поле номера студента*/
 char name[20];   /* имя студента*/
 int num_gr;      /* номер группы*/
 float mid_mark;  /* средний балл*/
 char comment[80]/* поле для комментария*/
};                /* конец блока структуры*/
```

Эта запись называется **описанием** структуры. Она начинается с ключевого слова **struct** и состоит из заключенного в фигурные скобки списка описаний. За словом **struct** может следовать необязательное имя, которое называется **именем типа** структуры (в примере - `Zurnal`). Этот идентификатор именует структуру и в дальнейшем может использоваться для сокращения подробного описания. Переменные, упоминающиеся в структуре, называются **элементами** или **полями.** Следом за правой фигурной скобкой, заканчивающей список элементов, может следовать список переменных, так же, как и в случае базисных типов. Вот почему в приведенном выше описании структуры после закрывающей фигурной скобки стоит точка с запятой; она завершает пустой список.

После определения нового типа данных можно объявлять переменные. Например, используя указанное выше описание **`Zurnal`,** можно с помощью строки

```
struct Zurnal  a0, a1 ,a2;
```

описать структурные переменные `a0, a1, a2`, каждая из которых строится по шаблону, введенному структурой `Zurnal`. Любая

переменная a0, a1, a2 содержит в строго определенном порядке элементы number, name, num_gr, mid_mark, comment.

Графически структуру можно представить в виде строки таблицы, где вся строка имеет имя, соответствующее имени структуры, а ее отдельные столбцы – именам полей:

Zurnal

number	name	num_gr	mid_mark	comment
int	char[20]	int	float	char[80]

Так же, как и переменные других типов, переменную структурного типа можно инициализировать, в этом случае за определением следует список начальных значений элементов:

```
struct Zurnal b0 = { 11, "Ivanov O.O.",
                     667, 87.5,
                     "budget"};
```

Операция присваивания определена для переменных одного и того же структурного типа, например, исходя из вышеприведённого описания, будет справедлива операция a0=a1; (при этом выполняется поэлементное копирование). В большинстве других случаев приходится оперировать непосредственно с полями структуры.

Обращение к отдельному полю структуры осуществляется с помощью точечной нотации (селектора поля), то есть, указываются имя структуры и имя поля, разделенные точкой. Например, мы можем с учетом введенных обозначений присвоить полям структуры значения:

```
a0.name = "Semenov V.U.";
a0.number = 12;
a0.num_gr = 602;
a0.mid_mark = 60.6;
```

Строку, как и другие элементы структуры, можно вводить с клавиатуры:

```
scanf("%s", a0.comment);
scanf("%d %f", &a0.number, &a0.mid_mark);
```

На практике структурные переменные обычно появляются в виде массива или списка. Например, описанные выше переменные `a0`, `a1`, `a2` можно объединить в массив, состоящий из элементов типа `struct Zurnal`, применив описание

```
struct Zurnal a[3];.
```

Теперь в программе можно выполнять, например, следующие действия:

```
a[0].name ="Petrov K.K.";/*Присвоение значения*/
...
if(a[i].number>a[i+1].number)/*Сравнение полей*/
   {
   p=a[i].number;
   a[i].number =a[i+1].number;
   a[i+1].number = p;
   }
```

Структуры можно передавать в функцию целиком, а также возвращать в качестве значения функции.

Операции над структурами можно выполнять и с помощью указателей. Так, описание

```
struct Zurnal *uk;
```

значит, что `uk` - указатель на структуру типа `Zurnal`. Выборка элемента структуры в этом случае производится с помощью операции `'->'`, например: `uk->number`. Другая возможность: поскольку `uk` есть указатель на структуру `Zurnal`, то к элементу `number` можно обращаться и так:

```
(*uk).number = 12;
```

Указатель установлен на начало массива структур. Имя массива, как обычно, эквивалентно адресу его начального элемента, и при добавлении к указателю на структуру или вычитании из него целого

числа размер структуры учитывается автоматически. Так, оператор `uk=a;` устанавливает указатель на первый элемент массива структур, а запись `++a;` обеспечивает автоматический переход к следующему элементу. В выражении `(*uk).number` скобки обязательны, так как приоритет операции выделения элемента `" . "` выше, чем у `"*"`.

Основное достоинство структур в том, что они позволяют хранить вместе данные разных типов, и их совокупность, как правило, представляет собой модель какого-либо реального объекта: человека, механизма, книги и т.д.

2.1. Задания на работу со структурами

1. Описать два комплексных числа и проделать над ними операции сложения, вычитания, умножения и деления.
2. Имеется база данных, содержащая числители и знаменатели дробных чисел. Например, последовательность чисел 5/18, –7/13, 9/8, … хранится в виде:

Номер структуры	1	2	3	…
Числитель	5	–7	9	…
Знаменатель	18	13	8	…

Найти и вывести номера структур, содержащих числа больше заданного (оно вводится с клавиатуры в десятичной форме), и сами числа (тоже в десятичной форме).

3. Информация об итогах сдачи сессии каждым студентом представлена в следующем порядке: Фамилия И.О., номер группы, экзаменационные оценки по четырём предметам. Определить процент студентов, сдавших экзамены на 4 и 5.
4. Создать массив структур для учета занятости аудитории: день недели, время учебной пары, аудитория, название предмета. Реализовать поиск периодов времени, когда выбранная аудитория свободна.
5. Даны стоимости двух товаров в рублях и копейках. Найти суммарную стоимость покупки и рассчитать сдачу.

6. Даны два отсчета времени в часах, минутах и секундах. Найти величину временного интервала в секундах.
7. Дано пять различных дат в виде: число, месяц, год. Вывести их на экран в порядке возрастания
8. Имеется список следующих сведений об экспорте из страны: наименование товара, страна-импортёр, объём поставки. Определить страны, в которые экспортируется конкретный товар и общий объём его экспорта.
9. Создать в программе массив структур, описывающих обозначение поля шахматной доски 8×8 (*a*5, *h*8 и т.п.). Вывести на экран координаты клеток возможных ходов конем с указанной позиции.
10. Ведомость содержит следующие сведения о сдавших вступительные экзамены: Ф.И.О., оценки (баллы) по отдельным дисциплинам. Например:

Name	Mathematic	Physics	Informatics
Petrov V.I.	88	76	73

Вывести на экран фамилии абитуриентов, имеющих средний балл 51 и выше, и их количество.

11. В расписании рейсов вылетов самолётов на определённый день содержится следующая информация: номер рейса, тип самолёта, пункт назначения, время вылета. Например:

U124	Airbus 90	London	13:46

Определить, какие самолёты и когда летят до заданного пункта назначения.

12. В бюро занятости имеется список вакансий рабочих мест, содержащий следующие сведения: наименование организации, должность, требуемая квалификация (образование, разряд), стаж работы, заработная плата. Клиент, введя сведения о своей квалификации и требованиях, должен получить список возможных рабочих мест.
13. Список книг содержит следующую информацию: фамилии авторов, название книги, год издания. Найти все книги, в названии которых имеется определённое слово, например, «волны».

14. Список имеющихся в продаже автомобилей содержит следующие сведения: марка автомобиля, цвет, стоимость, мощность двигателя, расход бензина на 100 км. Вывести перечень автомобилей, удовлетворяющих определённым требованиям клиента, таким например, как стоимость в диапазоне 300000 – 500000 руб., расход бензина в пределах 8 – 10 л и т.п.
15. Ведомость успеваемости студентов курса содержит следующую информацию: номер группы, фамилию, средний балл за последнюю сессию. Составить список студентов в порядке возрастания их номеров групп.
16. Регистратура больницы имеет список больных, структура о каждом из которых содержит: фамилию и инициалы, возраст, диагноз, номер палаты. Получить список больных, занимающих конкретную палату и имеющих возраст в определенном диапазоне.
17. Список жителей города содержит следующую информацию: фамилия и инициалы, улица, дом, квартира. Получить Ф.И.О. жителей, проживающих по одному и тому же адресу.
18. В бюро знакомств имеются списки мужчин и женщин, содержащие следующие сведения: 1) шифр, 2) пол, 3) возраст, 4) образование, 5) рост. Получить список возможных пар (шифров) с учётом требований кандидатов.
19. Создать структуры, содержащие требования на книгу (сведения о книге и заказчике): шифр, автор, название; сведения о студенте: номер читательского билета, фамилия, дата заказа. Вывести на экран названия книг, которые заказал определенный студент.
20. Создать структуры, содержащие сведения о книге: шифр, автор, название, год издание, количество страниц. Вывести на экран названия книг года издания ранее, чем указанный, и объемом не менее указанного числа страниц.

3. РАБОТА С ФАЙЛАМИ

Хранение данных в переменных и массивах является временным, так как все эти данные теряются при завершении работы программы. При сохранении любая информация, введенная в память компьютера, записывается в файлы для долговременного хранения. Программы, которые Вы пишете в редакторе, при сохранении на диске также становятся файлами. Файлы имеют имя (состоящее из названия и возможного расширения, отделяемого точкой) и хранятся на дисках и других внешних запоминающих устройствах. Без файлов работа компьютера немыслима, потому что и операционная система, и все выполняемые программы также хранятся в файлах. В файлах может храниться и любая другая информация, нужная пользователю.

Использование файлов предоставляет следующие возможности и удобства для программиста:

- сохранение информации;
- поддержание связи между программами, когда в одной создается, а в другой обрабатывается информация;
- ввод информации осуществляется не во время работы программы, а заранее, что очень удобно при большом объеме данных;
- программа может помещать данные в файл без присутствия пользователя.

Язык СИ рассматривает любой файл как последовательность байтов. Каждый файл оканчивается или маркером конца файла (EOF)

байт 0	байт 1	байт 2	...	EOF

или особым байтом, определенным в работающей с файлом программе.

Работа с файлом начинается с объявления указателя на структуру FILE (определенную в `<stdio.h>`).

Пример такого объявления:

```
FILE *file1;
```

где `file1` – имя указателя.

Если Вы собираетесь использовать несколько файлов, то Вам нужны указатели для каждого из них.

Далее необходимо установить связь между программой и физическим файлом (открыть файл), присвоив конкретное значение указателю. Для этого служит функция `fopen(…,…)`, которой передаются два аргумента: имя файла и режим доступа к файлу.

Существуют следующие режимы доступа и соответствующие им параметры:

`r` – открывает уже существующий файл для чтения;

`w` – создает и открывает новый файл для записи;

`a` – открывает для добавления (дозаписи) информации в конец файла.

Многие компиляторы СИ позволяют открывать файлы одновременно и для чтения и для записи. Для этого в описание режима доступа добавляется символ `'+'`. Так,

`r+` – открывает уже существующий файл для чтения и для записи;

`w+` – создает и открывает новый файл для чтения и записи.

`a+` – открывает для чтения и записи (добавляет в случае существования файла).

Пример:

```
file1=fopen("data.dat", "w");
```

(указатель `file1` связывается с файлом `data.dat`, который будет открыт или создан для записи).

! Открытие уже существующего файла в режимах «w» и «w+» приведет к уничтожению без предупреждения всей хранящейся в нем информации.

Чаще всего в пользовательских файлах хранятся либо тексты, либо числа (данные), поэтому в языке Си выделяют два вида файлов – текстовые файлы и бинарные файлы.

Текстовый файл – файл, содержащий текст, разбитый на строки парой специальных кодов: «возврат каретки» (0x13) и «перевод строки» (0x10). Если файл открыт в текстовом режиме, то при чтении из такого файла комбинация этих кодов преобразуется в один символ

'\n' — переход к новой строке. При записи в файл осуществляется обратное преобразование. Все указанные выше параметры режимов открывают текстовые файлы.

Бинарный файл – файл, из которого байты считываются и выводятся в первоначальном виде без каких-либо преобразований. Если требуется указать на такой файл, то к параметру добавляется буква b. Например: rb, или wb, или r+b. В некоторых компиляторах текстовый режим обмена обозначается буквой t, т.е. записывается r+t или rt.

При открытии файла с ним связывается специальная область оперативной памяти – буфер, по мере заполнения которого и идёт ввод–вывод информации. После окончания работы с файлом его рекомендуется закрыть, то есть прервать связь между файлом и программой. Для этого служит функция fclose(…), которой необходимо передать указатель на закрываемый файл. При таком явном закрытии происходит выгрузка содержимого буфера на внешний носитель, что позволяет избежать потери данных при выводе. Во время выполнения программы такую выгрузку буфера можно выполнить с помощью функции fflush().

Пример:

```
fclose (file1); /* закрывает файл data.dat, связанный с указателем file1 */).
```

Функции чтения из файла и записи в файл:

- fputc(переменная типа char, указатель на файл) – посимвольная запись данных в файл.
- fgetc(указатель на файл) – посимвольное чтение из файла.
- fputs(переменная типа строка, указатель на файл) – построчная запись данных в файл. Записывает в файл строку, но в конце не добавляет символ окончания строки.
- fgets(переменная типа строка, длина, указатель на файл) – построчное чтение данных из файла. Читает строку целиком до символа новой строки, если ее длина не превышает значения параметра «длина» минус один символ. Параметр «длина» является целым числом или целочисленной переменной,

указывающей максимально возможное количество символов в строке.

- `fprintf(указатель на файл, строка формата, список переменных)` – форматированный вывод символов, строк или чисел в файл.
- `fwrite(указатель на буфер хранения данных, размер элемента, количество элементов, указатель на файл)` – запись заданного количества блоков данных определённой длины из буфера в файл.
- `fscanf(указатель на файл, строка формата, список переменных)` – форматированный ввод символов строк или чисел из файла.
- `fread(указатель на буфер размещения данных, размер элемента, количество элементов, указатель на файл)` – чтение блоков данных заданного размера в указанном количестве из файла в буфер.

Другие функции работы с файлами:

- `feof(указатель на файл)` – функция определяет, достигнут ли конец файла. Если текущая позиция является концом файла (`EOF`), то функция возвращает ненулевое значение, в противном случае возвращается 0.
- `fflush(указатель на файл)` – принудительная очистка буфера вывода путем передачи содержимого на ВЗУ.
- `remove(имя файла)` – удаляет файл. Функция `remove()` возвращает 0, если файл успешно удален.
- `rename(старое имя, новое имя)` – переименовывает файл или директорию, указанную в параметре «старое имя», и присваивает имя, указанное в параметре «новое имя». Также может применяться для перемещения файла.
- `fseek(указатель на файл, количество байт, начало отсчёта)` -- устанавливает указатель текущей позиции в файле. Количество байт отсчитывается от значения параметра «начало отсчета», оно определяет новое значение указателя текущей позиции, а начало отсчёта - это один из следующих макросов: начало файла (`SEEK_SET`), текущая позиция (`SEEK_CUR`), конец

файла (SEEK_END). Обычно данная функция применяется только для бинарных файлов.

Пример работы с текстовыми файлами: необходимо открыть имеющийся файл first.txt, посимвольно считать его содержимое и записать это содержимое в новый файл second.txt. Ниже приведена программа, которая позволяет произвести все эти действия.

```
#include <stdio.h>
#include <stdlib.h>
void main( )
{
FILE *f1,*f2;
int ch;
if ((f1 = fopen("first.txt","r")) == NULL)
 {/* корректный выход из программы в случае ошибки
     открытия файла */
  fprintf(stdout, "Error opening file c_rec");
  exit(1);
 }
if ((f2 = fopen("second.txt","w")) == NULL)
 {
  fprintf(stdout, "Error opening file b_rec");
  exit(1);
 }
while (feof(f1)==0)
  {/*посимвольное чтение из файла first.txt и запись в
     файл second.txt, пока не будет достигнут конец файла*/
   ch = fgetc(f1);
   fputc(ch,f2);
   fputc(ch,stdout);
  }
fclose(f1);
fclose(f2);
}
```

Пример программы работы с бинарным файлом: необходимо записать в бинарный файл first.dat информацию по двум книгам

(название, имя автора, год издания) в виде структуры, а затем считать записанную информацию.

```
#include <stdio.h>
#include <stdlib.h>
#define N 2
struct tag_book          /*описание структуры */
{
char name[100];
char author[100];
int year;
} books[N];

void  main()
{
FILE *fp;
int i;
for( i=0; i < N; i++)
  {
  puts("Enter book's name:");
  scanf("%s", books[i].name);
  puts("Enter author's name:");
  scanf("%s", books[i].author);
  puts("Enter year of edition:");
  scanf("%d", &books[i].year);
  }
if ((fp = fopen("first.dat", "wb+"))== NULL)
 {
  fprintf(stdout, "Error opening file");
  exit(1);
 }
fwrite(books, sizeof(books),1, fp);     /* запись
в файл first.dat массива books, состоящего из двух
элементов */
fflush(fp);
fseek(fp,0,SEEK_SET);   /*обращение к началу файла*/

fread(books, sizeof(books),1, fp);   /*чтение из
файла first.dat массива books, состоящего из двух
```

```
  элементов*/
  fclose(fp);
  printf("Reading of file:\n");
  for(i=0; i < N; i++)
   {/* вывод информации о книгах на экран монитора */
    puts(books[i].name);
    puts(books[i].author);
    printf("%d\n",books[i].year);
   }
}
```

3.1. Задания на работу с файлами

1. Создать текстовый файл, содержащий информацию о студентах (фамилия, имя, номер группы, средний балл). Затем создать бинарный файл и переписать в него информацию о студентах определенной группы (номер группы считать с клавиатуры).

2. Создать текстовый файл *ft*, содержащий *n* строк по $2 \cdot m$ целых чисел. Считать числа из файла *ft*, сформировать два массива размерностью $n \times m$ из четных и нечетных столбцов и записать эти массивы в бинарный файл.

3. Создать в программе бинарный файл *fd*1, содержащий массив $n \times m$ целых чисел. Из файла *fd*1 переписать транспонированный массив в бинарный файл *fd*2. Создать также текстовый файл, в котором в строках будут записаны строки матрицы из файла *fd*1, знаки "=" и числа, равные сумме элементов строк матрицы.

4. Создать в программе текстовый файл, содержащий сведения о студентах первого курса: Фамилия И.О., группа, оценки за экзаменационную сессию. Затем информацию о студентах, имеющих задолженности, переписать в бинарный файл.

5. Создать текстовый файл *ft*, содержащий *n* строк по $2 \cdot m$ целых чисел. Считать числа из файла *ft*, сформировать два массива размерностью $n \times m$ из четных и нечетных столбцов и записать эти массивы в бинарный файл *fd*.

6. Создать текстовый файл *ft*, содержащий *n* строк по *m* целых чисел в строке (числа в строке записать с 1-ой позиции через пробел). Переписать числа из текстового файла *ft* в бинарный файл с данными *fd* без пробелов.

7. Создать текстовый файл *ft*, компонентами которого являются целочисленные массивы *a*1,.., *a*10, заполненные случайными числами.

Затем считать массивы из файла и записать в бинарный файл *fd*, расположив элементы четных массивов в обратном порядке.

8. Создать текстовый файл *ft*, компонентами которого являются целочисленные массивы *a*1,.., *a*10, заполненные случайными числами, как положительными, так и отрицательными (–20…20). Затем считать массивы из файла, преобразовать их и записать в бинарный файл *fd*. Преобразовать массивы следующим образом: положительные элементы заменить на 1, отрицательные – на –1, а нулевые оставить без изменения.

9. В программе создать текстовый файл *ft*, содержащий матрицу *A* действительных чисел размерностью 4×4. Из матрицы *A* получить матрицу *B* по следующему рекуррентному соотношению *B*(*m*, *n*) = *A*(*m*, *n*)∗(*m*–*n*). Записать матрицу *B* в бинарный файл *fd* и вывести ее на экран.

10. В текстовый файл *ft* записать буквы латинского алфавита и цифры. Занести в бинарный файл *fd*1 только латинские буквы, а в бинарный файл *fd*2 – только цифры. Предусмотреть в программе просмотр бинарных файлов *fd*1 и *fd*2.

11. Создать текстовый файл *ft*, содержащий *n* строк с текстом. Считать строки и вывести в бинарный файл *fd* *n* чисел, соответствующих количеству слов в каждой строке.

12. В программе создать текстовый файл *ft*, содержащий *n* случайных целых чисел. Считать числа из *ft*, подсчитать среднее значение всех чисел и записать в бинарный файл *fd* все числа меньше этого среднего значения.

13. Создать бинарный файл *fd*, содержащий n случайных чисел от 0 до 100. Считать числа и записать в текстовый файл *ft* все числа, делящиеся на три.

14. Создать текстовый файл *ft*, содержащий *n* случайных чисел от 0 до 9. Провести частотный анализ получившейся последовательности, т.е. указать (в %), сколько раз встречается та или иная цифра. Результаты вывести в бинарный файл *fd*.

15. Создать текстовый файл *ft*, содержащий *n* строк с текстом. Прочитать все строки и вывести в бинарный файл *fd* значения длин всех строк.

16. Создать текстовый файл *ft*, содержащий последовательность чисел Фибоначчи (*F*1=0, *F*2=1,…*Fn*=*Fn*–1+*Fn*–2). Считать

последовательность и переписать в бинарный файл *fd* все числа в обратном порядке.

17. Создать текстовый файл *ft*, компонентами которого являются случайные целые числа. Найти процентное соотношение четных и нечетных чисел в файле и вывести результат в бинарный файл *fd*.

18. Создать текстовый файл, содержащий случайные целые числа (от −10 до 10). Переписать в бинарный файл *fd* сначала все положительные, а затем все отрицательные числа, не меняя исходный порядок в каждой группе чисел.

19. Создать текстовый файл *ft*, содержащий матрицу целых случайных чисел (от 0 до 10). Заменить нулями элементы матрицы, стоящие на пересечении строк и столбцов, в которых имеется хотя бы по одному нулю. Получившуюся матрицу записать в бинарный файл *fd* и вывести на экран.

20. Создать текстовый файл *ft*, компонентами которого являются целочисленные массивы *a*1,.., *a*10, заполненные случайными числами. Затем считать массивы из файла и записать в бинарный файл *fd*, расположив элементы массивов в порядке возрастания.

4. РАБОТА В ГРАФИЧЕСКОМ РЕЖИМЕ

Язык программирования C/C++ не содержит встроенных средств работы в графическом режиме. Однако ряд реализаций языка позволяет использовать дополнительную библиотеку графических средств, которую можно подключить к программе с помощью заголовочного файла `graphics.h`. Библиотека содержит графические функции, константы и переменные управления экраном, графические шрифты и др., информацию о которых можно получить через `help`. Рассмотрим средства, необходимые для рисования графиков функций.

В графическом режиме используется следующая система координат. Пиксел (pixel) с координатами (0,0) находится в левом верхнем углу экрана, при этом ось 0x направлена традиционно (слева направо), а вот ось 0y – сверху вниз. Таким образом, координаты графических объектов могут иметь только неотрицательные значения. Разрешение используемого режима работы видеосистемы (количество точек экрана по горизонтали и по вертикали) можно определить с помощью функций `getmaxx()` и `getmaxy()`.

Вследствие различий в управлении текстовым и графическим экраном в программе необходимо указать, в каком режиме она (или ее часть) будет работать. Инициализацию графического режима работы видеосистемы можно организовать с помощью следующей последовательности:

```
int gr_driver=detect; gr_mode; /*константа detect запускает автоматическое обнаружение типа видеоадаптера и режима его работы*/
initgraph(&gr_driver, &gr_mode, "C:\\BC\\BGI"); /*здесь последним аргументом является строка, указывающая путь по файловой системе к каталогу BGI языка Си, содержащему драйверы управления работой видеоадаптера в графическом режиме. На Вашем компьютере путь может отличаться от указанного в этом примере*/
```

В конце блока программы, работающего в графическом режиме, размещают функцию `closegraph()`, возвращающую видеосистему в текстовый режим.

Ниже приводятся некоторые другие графические функции:

`cleardevice()` – функция очистки графического экрана;

putpixel(X, Y, цвет) – вывод пиксела с координатами X и Y. Цвет пиксела задаётся числом от 0 до 15;

line(X1, Y1, X2, Y2) – черчение отрезка прямой от точки (X1, Y1) до точки (X2, Y2);

moveto(X, Y) – перемещение графического курсора в точку с координатами X,Y;

lineto(X, Y) – черчение линии от текущего положения графического курсора до точки с координатами (X, Y);

rectangle(xl ,yl, xr, yr) – черчение прямоугольника, который задаётся координатами верхней левой вершины (xl ,yl) и правой нижней вершины (xr, yr);

setcolor(цвет) – функция, задающая цвет изображаемых линий;

setbkcolor(цвет) – функция, задающая цвет фона формируемого изображения;

settextstyle(…,…,…) – установка стиля текста, выводимого функцией outtext();

outtext(textstring) – вывод текста текущим графическим шрифтом, начиная с текущего положения курсора.

outtextxy(X,Y, textstring) – вывод текста начиная с точки с координатами (X, Y);

restorecrtmode() – функция восстановления исходного текстового режима;

setgraphmode(getgraphmode()) – функция возврата в текущий графический режим.

Решаемая в данном разделе задача предусматривает вывод фрагмента графика функции в заданное экранное окно, при этом график представляется с максимально возможным разрешением, т. е. растягивается во всё окно, как по вертикали, так и по горизонтали. Должно быть предусмотрено изображение осей координат с соответствующими обозначениями, если они попадают на изображаемый участок. Над графиком изображается строка с названием функции и пределами изменения аргумента.

Для перехода от математических координат x, f(x) к экранным координатам Xgr, Ygr требуется выполнить линейное преобразование по формулам:

```
Xgr = a1·x +a2;
Ygr = b1·f(x) + b2.
```

Коэффициенты `a1`, `a2`, `b1`, `b2` находятся исходя из графических координат окна, в котором изображается график. Нетрудно получить, что

```
a1 = (xr – xl) / (Xmax – Xmin);
a2 = xl – a1·Xmin;
b1 = (yl – yr) / ( Ymax – Ymin );
b2 = yl – b1·Ymax.
```

Здесь `Xmin` и `Xmax` – границы диапазона изменения аргумента, `Ymin` и `Ymax` – минимальное и максимальное значения функции f(x) на отображаемом участке.

Алгоритм рисования графика функции f(x) должен содержать следующую последовательность действий:

1) подключение к программе заголовочного файла `graphics.h`;
2) задание изображаемой функции;
3) ввод пределов изменения аргумента функции: Xmin и Xmax;
4) переход в графический режим;
5) определение с помощью функций `getmaxx()` и `getmaxy()` максимальных графических координат дисплея;
6) вывод на экран максимальных графических координат дисплея;
7) задание координат прямоугольного окна, в котором должна быть изображена функция. Окно задается экранными координатами верхнего левого угла (xl, yl) и правого нижнего угла (xr, yr);
8) определение (отдельной функцией) максимального и минимального значений функции (Ymin, Ymax) на выбранном участке с учётом того, что максимально возможное число изображаемых точек графика равно xr–xl+1;
9) выполнение действий, связанных с переходом от математических координат к экранным;

10) рисуется график функции (точками (`putpixel()`) или отрезками прямой (`lineto()`));
11) чертятся оси координат (если они попадают на изображаемый участок) и наносятся их обозначения;
12) над графиком выводится формула изображаемой функции, а также пределы изменения аргумента на исследуемом участке;
13) организуется возвращение в текстовый режим (`closegraph()`).

Пример результата работы программы рисования графика функции приведен на рисунке 4.1.

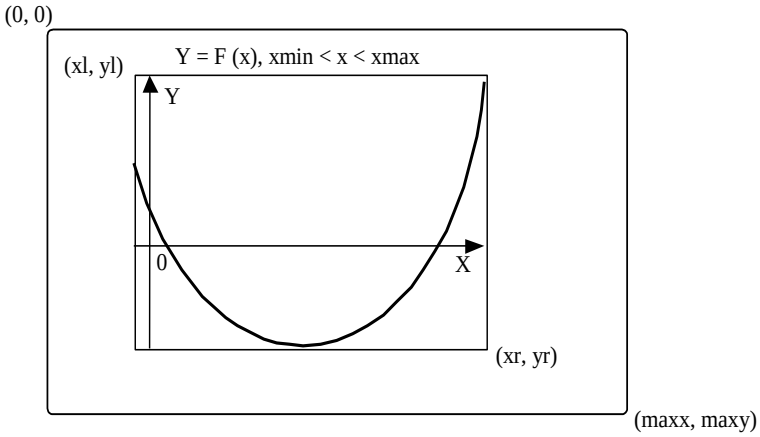


Рис. 4.1. Изображение на экране в заданном координатами (xl, yl), (xr,yr) окне участка (xmin, xmax) графика функции F(x), содержащего точку начала координат

4.1. Задания на построение графиков

1. $y = \sin(x)$;
2. $y = \cos(x + 0{,}8)$;
3. $y = \exp(x)$;
4. $y = 0{,}5 - \pi \cdot |\sin(x)| / 4$;
5. $y = \exp(-x^2)$;
6. $y = 2 \cdot (\cos(x) - 1)$;

7. $y = \cos(x)$;
8. $y = \exp(3x)$;
9. $y = \sin(x + 0{,}5)$;
10. $y = \exp(x \cdot \ln 5)$;
11. $y = \ln(x)$;
12. $y = x \cdot \exp(x^{0,5})$;
13. $y = \ln(1 - x)$;
14. $y = (1 + 2x^2) \cdot \exp(x^2)$;
15. $y = \ln((1 + x)/(1 - x))$;
16. $y = \sin(x)/x$.

5. РАБОТА С УКАЗАТЕЛЯМИ И ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

5.1. Общие сведения об указателях

Следующим шагом в овладении программированием на Си является использование указателей. Указатель – это переменные, которые отличаются от обычных переменных. Точно так же как значением переменной типа `int` является целое число, значением переменной типа указатель служит адрес некоторой величины. Адрес – это, проще говоря, номер байта в памяти, где располагаются переменные и функции программы. При помощи указателей Вы можете самостоятельно распоряжаться памятью компьютера. Ни одну серьёзную программу на языке Си (и не только на языке Си, но и на любом другом языке программирования) нельзя написать без применения указателей. Однако этими возможностями можно эффективно пользоваться при условии, что Вы знаете средства управления и сами можете контролировать правильность их применения в программе. Поскольку указатели хранят адреса переменных, используемых в программе, а все переменные должны иметь тип, то и указатели должны иметь соответствующие типы.

Указатели объявляются среди обычных переменных. При объявлении переменной типа указатель перед именем переменной ставится знак *. Вот пример оператора объявления переменной i типа `int` и переменной - указателя `pin` на величину типа `int`:

```
int i, *pin;
```

Аналогичным образом могут быть объявлены переменные–указатели для любого типа. При необходимости, можно ввести нетипизированные указатели. Это указатели на тип `void`:

```
void *pv;
```

5.2. Операции над указателями

1. Операция получения адреса переменной `&`.

Когда за знаком `&` следует имя переменной, результатом операции является адрес указанной переменной. Пример:

```
int *pin, nurse;
```

```
pin=&nurse;   /* переменной pin присваивается адрес
переменной nurse. */
```

2. Операция присвоения указателей.

Указателю можно присвоить значение указателя того же типа или типа `void`. В последнем случае требуется использовать приведение к типу. Указателю на тип `void` может быть присвоен указатель любого типа. Тогда также требуется использовать приведение к типу. Пример:

```
int *ip,*pin, nurse;
void *pv;
double *pd;
pin=&nurse;
ip=pin;
pv=( void *) ip; /* приводим значение указателя ip
                 к типу void */
pd=(double*)  pv; /*  приводим значение указателя pv к типу
                      double.  В итоге, указатель на double
                      ссылается на то же место, что и указатель на
                      int. Автор программы должен четко
                      представлять, для чего это ему нужно. */
```

3. Операция косвенной адресации * (операция разыменования указателя).

Когда за знаком * следует указатель на переменную, результатом операции является величина, помещенная в ячейку с указанным адресом. Пример:

```
int val,*ptr,nurse;
nurse = 22;
ptr = &nurse;
val = *ptr; /*переменной val присваивается значение переменной
nurse. */
```

4. Операции сложения (+) и инкремента (++).

Указатели нельзя складывать друг с другом, но можно сложить указатель и целую величину. Как происходит выполнение операций сложения и инкремента, можно видеть из следующего примера:

```
int *pin, nurse;
pin=&nurse; /* Пусть после этого переменная pin примет значение
        0x23f0 */
pin+=1; /* Переменная pin примет значение 0x23f2. Значение
        переменной увеличится не на 1, а на 2 (для windows- и
        linux-приложений на 4). При сложении целого числа с
        указателем происходит не арифметическое сложение, а
        указатель начинает «указывать» на величину, отстоящую от
        переменной nurse на число ячеек типа int, равное целой
        величине, использованной в операции сложения*/
pin++;/* Переменная pin изменит свое значение с 0x23f2 на 0x23f4.
        Указатель начинает «указывать» на величину, следующую
        за величиной, на которую «указывал» ранее */
```

5. Операции вычитания (–) и декремента (– –).

Можно находить разность двух указателей. Разность двух указателей есть целое число, равное количеству элементов типа, соответствующего типу указателя, которые уместились бы между двумя адресами, значения которых хранятся в этих указателях.

Разность указателя и целой величины, а также декремент трактуются аналогично сложению и инкременту, с той лишь разницей, что в данном случае значения адресов уменьшаются.

6. Указатели можно передавать как параметры в функции. Указатель может быть возвращен функцией. Примером функций, возвращающих значения типа указатель, может быть функция `fopen()`.

5.3. Массивы и указатели

Язык Си сконструирован так, что имя массива является константой типа указатель, значение которой равно адресу первого элемента этого массива.

Поэтому имя массива может использоваться как указатель, и наоборот. Пример объявления указателей и использования их в программе как массивов:

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

```
int i, *ip, c[5];
for(i=0;i<=4;i++) *(c+i)=i; /* используем имя массива
                                как указатель*/
ip=c;
for(i=0;i<=4;i++) printf("%d  ",ip[i]);
            /*используем указатель как массив */
return(0);
}
```

```
  0     1     2     3     4
```

5.4. Динамические переменные

Для того, чтобы рационально использовать память, выделенную программе, вы можете размещать данные в памяти и удалять их, высвобождая память, в процессе выполнения программы, т.е. **динамически**. При этом в программе используется не имя переменной, а адрес области памяти. Размещаемый объем данных будет ограничен только свободной памятью компьютера во время выполнения программы (для Windows он может достигать 4 Гб). Сам указатель помещается в том же блоке, что и обычные переменные, занимает объём, равный четырем байтам (для приложений MS DOS размер указателя может быть равен двум байтам), и содержит адрес области свободной памяти, начиная с которого можно размещать соответствующие данные. Например, в программе Вы используете два указателя, занимающие в памяти 8 байт, а во время работы программы с их помощью можно разместить и обработать данные, занимающие, скажем, 100 Кбайт.

Для запроса памяти, чтобы размещать переменные в языке Си можно использовать две стандартные функции `malloc()` и `calloc()`.

Функция `malloc()`. Прототип функции может быть записан следующим образом:

```
void *malloc(unsigned n);
```

Аргументом функции `malloc()` является количество запрашиваемой памяти в байтах. Функция возвращает значение адреса

начала выделенной области памяти, если выделение памяти произошло успешно, и NULL – если произошла ошибка.

Функция calloc(). Прототип функции может быть записан следующим образом:

```
void *calloc(unsigned size, unsigned n);
```

Аргументами функции calloc() являются две целые неотрицательные величины: размер блока памяти в байтах size и количество запрашиваемых блоков памяти n. Функция возвращает значение адреса начала выделенной области памяти, если выделение памяти произошло успешно, и NULL – если произошла ошибка.

Для освобождения запрошенной памяти используется функция free().

Прототип функции может быть записан следующим образом:

```
void free(void *);
```

Аргументом функции free() является начальный адрес памяти, ранее запрошенной функциями malloc() или calloc().

Для работы всех трех функций необходимо в начале текста программы добавить #include<stdlib.h> или #include<alloc.h>.

Пример объявления указателей и использования их в программе:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main( )
{
 unsigned char *str; /* указатель на символьную
                        переменную */
str = (unsigned char *)malloc(10); /* запрашиваем
    область памяти размером в 10 байт для размещения строки */
strcpy(str,  "Hello");/* копируем строку "Hello" в
                         выделенную область памяти */
printf("String is %s\n", str); /* печатаем строку из
        выделенной области памяти, используя указатель как
                                  имя массива*/
```

```
 free(str); /* освобождаем запрошенную память */
 return(0);
}
```

Для выполнения заданий достаточно будет этих трех функций процедур. С другими функциями работы с выделяемой памятью Вы можете ознакомиться в рекомендованной литературе.

5.5. Задания на указатели и динамическую память

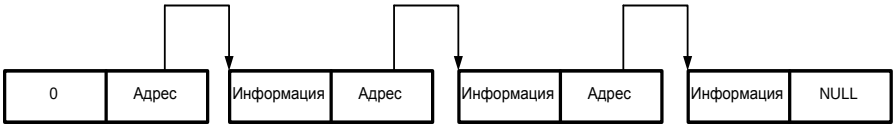
А. ***Указатели.*** Имеется массив указателей на целые числа (вектор **XP**), заданный следующим образом:

```
int *VectXP[100];
```

Разместить в памяти *n* (от 10 до 100) чисел, на которые будут ссылаться элементы вектора **XP**.

1. Написать функцию *max*() для нахождения наибольшего из чисел *n*, на которые ссылаются элементы вектора **XP**. Вывести это число.
2. Написать функцию *unique*(), которая в векторе **XP** все элементы, ссылающиеся на равные числа, заменяет на первый из этих элементов. Вывести элементы вектора до и после работы созданной процедуры.

Б. ***Списки.*** Цепочку данных можно представить в виде списка, где каждый предыдущий элемент содержит поле с указателем на последующий:



3. Вычислить значения *y* некоторой функции *F*(*x*) в *n* точках, результаты вычислений *y* вместе со значениями соответствующих *x* поместить в список. Вывести на экран элементы списка, содержащие значения *x* в интервале от *a* до *b*, и соответствующие им значения *y*.

4. В программе сгенерировать *n* вещественных чисел, поместить их в список в порядке неубывания и распечатать тот список.

5. Написать программу, которая считывает в список значения из файла любой длины.

6. Последовательность случайных вещественных чисел случайной длины записывается в очередь. Найти среднее арифметическое этой последовательности и поместить это число в середину списка.

В. ***Многочлен***. Представить многочлен $S(x)=52\cdot x^{40}-3\cdot x^{8}+x$ в виде списка, где каждый элемент указывает на последующий. Если коэффициент $a(i) = 0$, то звено не включается в список. При создании списка использовать структурные переменные с тремя полями: для хранения соответственно коэффициента, показателя степени и указателя на следующую запись.

7. Написать функцию *equal*(), проверяющую на равенство многочлены *p* и *q*. Протестировать при *p* = *q*.

8. Написать функцию *value*(), вычисляющую значения многочлена *p* в заданной целочисленной точке *x*.

9. Написать функцию *printP*(), которая выводит на экран многочлен *p*. Например, 52*x*^40 – 3*x*^8 + *x*.

10. Создать список, содержащий элементы многочлена Чебышева $T_n(x)$, определяемого формулами:

$T_0(x) = 1;$
$T_1(x) = x;$
$T_k(x) = 2x\cdot T_{k-1}(x) - T_{k-2}(x) \qquad (k = 2, 3,...).$

Вывести многочлен Чебышева $T_n(x)$ в виде, представленном, например, в задании 9.

Г. ***Текст***. Для удобства работы с длинным текстом на экране необходимо разделить его на строки, не превышающие длины экрана (80 символов). Одна из возможных реализаций такого разбиения – это разделить текст на строки ограниченной длины и создать массив указателей на эти строки. Строки при этом разместятся в массивах типа `unsigned char` следующим образом:

```
const unsigned len = 80; /*длина строки <=80*/
```

```
const unsigned num = 100; /* максимальное число строк
100*/
unsigned char*str[100];
int i;
       /* Создание массивов */
for(i=0; i<=num; i++)
      str[i]=(unsigned char*) malloc(len);
```

Для удобства отладки программ рекомендуется взять в качестве редактируемого текста файл, содержащий программу на языке Си. Если строка больше 80 символов, то для упрощения программы их можно отсечь. Сделать `num` > числа строк в обрабатываемом файле; при этом последним элементам массива `str`, не указывающим на строки, присвоить значение `NULL`. Разместить в памяти, используя массив указателей, преобразованные строки исходного текста программы и вывести их на печать.

11. Написать функцию *numberstring*( ) для подсчета числа строк в тексте. Напечатать это число.

12. Написать функцию *replacment*( ), заменяющую *i*-ю строку текста на копию *j*-й строки.

13. Написать функцию *rearrangment*( ), меняющую *i*-ю и *j*-ю строки текста.

14. Написать функцию *remove*( ), удаляющую *i*-ю строку из текста.

Д. **Стек.**

15.Последовательность случайных вещественных чисел случайной длины записывается в стек. Провести сортировку последовательности методом пузырька, работая со стеком.

ЛИТЕРАТУРА

1. М.Болски. Язык программирования Си. Справочник: Пер. с англ - М.: Радио и связь, 1988. – 96с., ил.
2. Р.Берри. Язык Си. Введение для программистов. / Р.Берри, Б.Микинз. - М.: Финансы и статистика, 1988.
3. М.Дансмур. ОС UNIX и программирование на языке Си./М.Дансмур, Г.Дейвис. - М.: Радио и связь, 1989.
4. Джехани Н. Программирование на языке СИ./Пер. с англ. И.Г.Шестакова под ред. Б.А.Кузьмина. –М.: Радио и связь, 1988, - 270 с.
5. Демидович Б.П. Основы вычислительной математики./ Демидович Б.П. , Марон И.А.; — М.: изд. Лань, 2009. – 664 с.
6. Дейтел, Х. М. Как программировать на С / Х.М. Дейтел, П. Дж. Дейтел. - М.: Изд-во «Бином», 2000.
7. Дуглас Т. Программирование на языке СИ для персонального компьютера IBM PC/Пер. с англ. Б.А.Кузьмина, под ред. И.В.Емелина, - М.: Радио и связь, 1991, - 428 с.
8. Жешке Р. Толковый стандарт языка Си: Пер. с англ. СПб.: Питер, 1994. 223с.
9. Керниган Б. Язык программирования Си./ Керниган Б., Ритчи Д. — 2-е изд. — М.: «Вильямс», 2007. - С. 304
10. Белецкий Ян. Энциклопедия языка СИ./Пер.с пол. Под ред. Ф.Ф.Пащенко. –М: Мир, 1992, -686 с.
11. Самарский А.А. Численные методы: Учеб. пособие для вузов./Самарский А.А., Гулин А.В. - М.:Наука, 1989. 432 с.
12. Турчак Л.И. Основы численных методов./Турчак Л.И., Плотников П.В. М.:ФИЗМАТЛИТ, 2002. 304 с.
13. М.Уэйт. Язык Си. Руководство для начинающих./М.Уэйт, С.Прата, Д.Мартин.- М.: Мир, 1988. – 512 с.
14. Р. Хазфилд, К. Лоуренс и др. Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста: Пер. с англ./Р. Хэзфилд, Л. Кирби и др. – М.: Изд. «ДиаСофт», 2001. - 736 с.
15. Л.Хэнкок. Введение в программирование на языке Си./ Л.Хэнкок, М.Кригер. - М.: Радио и связь, 1986. - 192 с.
16. Шилдт Г. Полный справочник по С. — М.: «Вильямс», 2004. - С. 704.