КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ИНСТИТУТ ФИЗИКИ
Кафедра радиоастрономии

Е.Ю. ЗЫКОВ, А.А. КОЛЧЕВ, О.Г. ХУТОРОВА

ОБРАБОТКА ДАННЫХ НА ЯЗЫКЕ СИ

Учебно-методическое пособие

Казань – 2019

УДК 681.924

Печатается по решению учебно-методической комиссии
Института Физики КФУ
Протокол № 3 от 22 ноября 2018 г.

Принято на заседании кафедры радиоастрономии
Протокол № 1 от 29 августа 2018 г.

Рецензент:
кандидат физико-математических наук,
доцент кафедры радиоэлектроники КФУ **И.А. Насыров**

Зыков Е.Ю., Колчев А.А., Хуторова О.Г., Обработка данных на языке Си / Е.Ю.Зыков, А.А. Колчев, О.Г. Хуторова. – Казань: Казан. ун-т, 2019. – 59 с.

Данное учебно-методическое пособие предназначено для методической поддержки при проведении практических занятий по дисциплинам, связанным с введением в информационные технологии, обязательным к изучению в Институте физики КФУ. Пособие содержит теоретический материал по методам представления и обработки данных с использованием языка программирования Си и практические задания для самостоятельной работы. Теоретический материал иллюстрируется примерами по составлению программ.

Последовательность изложения теоретического материала и практических заданий представлена в порядке выполнения практических работ по соответствующим курсам студентами Института физики КФУ.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Важной составляющей научных исследований является обработка экспериментальных данных. Вопросам представления текстовых, числовых и смешанных данных в языке Си и посвящено пособие. Оно предназначено для методического обеспечения при проведении практических занятий по дисциплинам, связанным с введением в информационные технологии, преподаваемым на 1-2 курсах Института Физики КФУ.

Пособие включает в себя разделы, описывающие работу с текстовой информацией (строки), со смешанной информацией (структуры), а также с информацией, записанной в файлы (как в текстовые, так и в бинарные). Дополнительно включены разделы, связанные с возможностью оптимизации вычислительных процессов в Си (динамическая память), а также представления полученных результатов в графическом виде.

Теоретический материал иллюстрируется большим количеством примеров практической реализации соответствующих алгоритмов представления и обработки данных на языке Си.

1. РАБОТА СО СТРОКАМИ

В языке Си не существует встроенного строкового типа данных. Строка в языке Си представляет собой одномерный массив символов (тип char), последним элементом которой является символ конца строки – NULL (**\0** - является отметкой конца строки).

Обычно для записи строковых констант в программе используются литералы. Строковый литерал - это последовательность букв, цифр и символов, заключенная в двойные кавычки. Примеры:

```
"This is a string literal";
"0123456789";
"*";
```

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

Таким образом, объявить строку в языке Си можно с помощью объявления массива символов (не забывая добавить место для завершающего нуля):

```
char s[40+1];
```

или присваивая строковой переменной начальное значение (при этом длину строки компилятор может вычислить сам):

```
char s[] = " This is a string literal";
```

Справа от знака присваивания записана строковая константа. В конце строки автоматически добавляется ноль (‘\0’). Константы символьных строк помещаются в класс статической памяти.

Поскольку литерал – 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";
```

Ввод строковой переменной с клавиатуры можно сделать с помощью функции getch(), осуществляющей посимвольный ввод данных:

```
#include <stdio.h>
int main ()
{  printf("vvod stroki\n");
   int i;
 char str[20];
 for (i=0; (str[i] = getchar()) != '\n'; i++);
 str[i] = '\0';
 printf("\n%s\n", str);
 return 0;}
```

В заголовке цикла `getch()` возвращает символ, далее записываемый в очередную ячейку массива. После этого элемент массива сравнивается с символом новой строки '\n'. Если они равны, то цикл завершается. После цикла символ '\n' в массиве "затирается" символом '\0'.

Однако удобней работать со строками с помощью функций стандартной библиотеки ввода-вывода `gets()` и `puts()`, которые получают строку из стандартного потока и выводят в стандартный поток. Буква s в конце слов gets и puts является сокращением от слова string (строка). В качестве параметров обе функции принимают указатель на массив символов (либо имя массива, либо указатель).

Простейший пример использования этих функций выглядит так:

```
#include <stdio.h>
int main ()
{char str[200];
  gets(str);
  puts(str);
  return 0;}
```

Итак, если вы работаете со строками, а не другими типами данных, при этом нет необходимости выполнять их посимвольную обработку, то удобнее пользоваться функциями `puts()` и `gets()`.

Ниже приведен пример более сложной программы, подсчитывающей число вхождений символа sym в строку s. Строка и символ вводятся с клавиатуры.

```
#include <stdio.h>
#include <stdlib.h>
int main() {
  char s[80], sym;
  int count, i;
  printf("Input string: ");
  gets(s);
  printf("Input symbol: ");
  sym = getchar();
  count = 0;
  for(i=0; s[i]!='\0'; i++)
  {
    if(s[i]==sym)
     count++;
  }
  printf("In string\n");
  puts(s);       // Вывод строки
  printf("symbol ");
  putchar(sym); // Вывод символа
  printf(" is found %d times",count);
  return 0;
}
```

Передача строки в функцию не отличается от передачи туда массива чисел:

```
#include <stdio.h>
int main ()
{char str[200];
 gets(str);
 change (str);
 puts(str);
 return 0;}
 void change (char *s)
{
for ( ;*s != '\0'; s++)
 (*s)++;
}
```

В этом примере функция `change()` принимает в качестве параметра указатель на символ. В теле функции значение указателя инкрементируется, указывая на следующий символ массива. В теле цикла инкрементируется значение, которое находится по адресу, который содержит указатель.

Набор строк можно представить как двумерный массив, т.е. массив, состоящий из одномерных массивов, где каждый одномерный массив — это строка символов:

```
char str[][10] = {"Hello", "World", "First", "Second"};
```

Для преобразования строк в языке Си предусмотрена библиотека string. Каждая из функций имеет свой формат записи (прототип).

Наиболее используемые функции приведены в таблице 1.

Таблица1.

Функции для работы со строками и символами в Си	
Функция	**Пояснение**
`strlen(имя_строки)`	определяет длину указанной строки, без учёта нуль-символа
`strcpy(s1,s2)`	выполняет побайтное копирование символов из строки s2 в строку s1
`strncpy(s1,s2, n)`	выполняет побайтное копирование n символов из строки s2 в строку s1. Возвращает значения s1
`strcat(s1,s2)`	дописывает строку s2 в конец строки s1. Результат сохраняется в s1
`strncat(s1,s2,n)`	дописывает n символов строки s2 в конец строки s1. Результат сохраняется в s1
`strcmp(s1,s2)`	сравнивает строку s1 со строкой s2 в лексикографическом порядке (по алфавиту) и возвращает результат типа int: 0 –если строки эквивалентны, >0 – если s1<s2, <0 – если s1>s2 с учётом регистра
`strncmp(s1,s2)`	сравнивает n символов строки s1 со строкой s2 и возвращает результат типа int: 0 –если строки эквивалентны, >0 – если s1<s2, <0 – если s1>s2 с учётом регистра
`stricmp(s1,s2)`	сравнивает строку s1 со строкой s2 и возвращает результат типа int: 0 –если строки эквивалентны, >0 – если s1<s2, <0 – если s1>s2 без учёта регистра
`strnicmp(s1,s2)`	сравнивает n символов строки s1 со строкой s2 и возвращает результат типа int: 0 –если строки эквивалентны, >0 – если s1<s2, <0 – если s1>s2 без учёта регистра

`isalnum(c)`	возвращает значение true, если с является буквой или цифрой, и false в других случаях
`isalpha(c)`	возвращает значение true, если с является буквой, и false в других случаях
`isdigit(c)`	возвращает значение true, если с является цифрой, и false в других случаях
`islower(c)`	возвращает значение true, если с является буквой нижнего регистра, и false в других случаях
`isupper(c)`	возвращает значение true, если с является буквой верхнего регистра, и false в других случаях
`isspace(c)`	возвращает значение true, если с является пробелом, и false в других случаях
`toupper(c)`	если символ с является символом нижнего регистра, то функция возвращает преобразованный символ с в верхнем регистре, иначе символ возвращается без изменений
`strchr(s,c)`	поиск первого вхождения символа с в строке s. В случае удачного поиска возвращает указатель на место первого вхождения символа с. Если символ не найден, то возвращается ноль
`strcspn(s1,s2)`	определяет длину начального сегмента строки s1, содержащего те символы, которые не входят в строку s2
`strspn(s1,s2)`	возвращает длину начального сегмента строки s1, содержащего только те символы, которые входят в строку s2
`strprbk(s1,s2)`	Возвращает указатель первого вхождения любого символа строки s2 в строке s1
`atof(s1)`	преобразует строку s1 в тип double

`atoi(s1)`	преобразует строку s1 в тип int
`atol(s1)`	преобразует строку s1 в тип long int

Рассмотрим более подробно некоторые функции.

1. `int strcmp(const char * _Str1, const char * _Str2);` - функция сравнения строк.

Если символы строки _Str1 раньше встречаются в алфавите, то _Str1 больше, чем _Str2, и результат функции будет меньше нуля. Если они равны, то результат равен нулю. Если символы строки _Str2 раньше встречаются в алфавите, то _Str2 больше, чем _Str1, и результат функции будет меньше нуля.

```
#include <stdio.h>
#include <string.h>
int main( )
{
  char *szString1 = "stroka";
  char *szString2 = "stroka";
  printf("\n\tString Comparing\n\n");
  int compareResult = strcmp(szString1,szString2);
  printf("compareResult\t=\t%d\n\n", compareRsult);
  return 0; }
```

Функция `strcmp` различает большие и маленькие буквы.

2. `int stricmp(const char * _Str1, const char * _Str2);` - функция сравнения строк, игнорирует разницу больших и маленьких букв, в остальном такая же, как и strcmp

3. `size_t strlen(const char * _Str)` - возвращает длину строки С в виде количества байтов.

```
#include <stdio.h>
#include <string.h>
int main( )
{
  char *szString1 = "Stroka";
  printf("szString1 = %s\n", szString1);
  int stringLength = strlen(szString1);
  printf("String length = %d bytes\n\n", stringLength);
  return 0; }
```

4. `char * strcpy(char * _Dst, const char * _Src);` - функция копирования строк.

```
#include <stdio.h>
#include <string.h>
int main( )
{
  char cMassiv[100];
  char *szStr = "String1";
  strcpy(cMassiv, szStr);
  printf("\ncMassiv = %s\n\n", cMassiv);
  return 0; }
```

5. `char * strncpy(char * _Dst, const char * _Src, size_t _MaxCount);` - функция копирования строк с указанием количества копируемых символов.

6. `char * strcat(char * _Dst, const char * _Src);` - функция конкатенации строк, т.е. присоединение одной строки к другой.

```
#include <stdio.h>
#include <string.h>
int main( )
{
  char cMassiv[100];
  char *szStr = "String1_";
  strcpy(cMassiv, szStr);
  char *szStr2 = "Noventa";
  strcat(cMassiv, szStr2);
  printf("\ncMassiv = %s\n\n", cMassiv);
  return 0;}
```

7. `char * strncat(char * _Dst, const char * _Src, size_t _MaxCount);` - функция конкатенации строк с указанием количества символов.

8. `int isdigit( int character );` - функция проверяет аргумент, передаваемый через параметр character, является ли он десятичной цифрой.

Значение отлично от нуля (т.е. истинно), если аргумент функции character – это десятичная цифра, ноль (т.е. ложь) в противном случае.

В примере подсчитывается количество цифр в строке str:

```
#include<stdio.h>
#include <string.h>
int main()
{
  int count = 0;
  int i;
  char str[40];
```

```
    puts("Enter string with digits:");
    gets(str); /*ввод строки */
    for (i=0; i<strlen(str);i++)
      {
        if (isdigit(str[i])) count++;
        /*если str[i] цифра, увеличить счетчик */
      }
    printf("Number of digits = %d\n",count);
    return 0;
  }
```

Чтобы проверить, является ли текущий символ цифрой, используется функция `isdigit`.

9. `char *strtok(char *str1, const char *str2);` - функция ищет в строке str1 лексемы, выделенные символами из второй строки. Здесь str1 – указатель на разбиваемую строку, str2 – указатель на строку, содержащую набор символов разделителей. Возвращаемое значение: NULL – если строку str невозможно разделить на части, указатель на первый символ выделенной части строки.

Функция `strtok` выделяет очередную часть строки, на которую указывает аргумент str1, отделенную одним из символов разделителей, указанных в строке, на которую указывает аргумент str2. Последовательный вызов функции strtok приводит к разбиению строки str на части (лексемы).

Пример использования функции `strtok` для выстраивания слов в строке по количеству символов в слове.

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

```
int main( )
{
  int i;
  char str[101];
  printf("Enter: ");
  gets(str); /* Ввод строки */
  char array[20][20];
  int count = 0;
  char *ptr = strtok(str,"\t ");
  for(count=0; ptr!=NULL; count++)
    {
      strcpy(array[count],ptr);
      /* Строка разбивается на слова*/
      ptr = strtok(NULL,"\t ");
    }

int flag = 1;  /* Сортировка слов по длине*/
 while(flag){
    flag = 0;
    for(i = 0;i<count-1;i++)
      if(strlen(array[i])>strlen(array[i+1]))
        {
          char buffer[11];
          strcpy(buffer,array[i]);
          strcpy(array[i],array[i+1]);
          strcpy(array[i+1],buffer);
          flag = 1;
        }
  }
 strcpy(str,"");
```

```
  for( i=0;i<count;i++)
    {
      strcat(str,array[i]);
      strcat(str,"    ");
    }
  printf("Result: ");
  puts(str);
  return 0;
}
```

Строка вводится с клавиатуры. Она преобразуется в двумерный массив, состоящий из слов. Далее слова сортируются по возрастанию количества букв в слове.

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, содержащие «i» и «a»; если таких слов нет, то вывести 0.
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. Вывести все слова, читающиеся одинаково слева направо и наоборот, справа налево. Если таких слов нет, то вывести 0.
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. Заменить все первые в слове строчные буквы на прописные, а прописные на строчные.

7. Найти слова, первый и последний символы которых совпадают, и вывести эти слова и их количество.

8. Дан текст, в словах которого могут встречаться цифры и знаки "+", "–", "*", "/". 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 "*a*".

17. Подсчитать количество букв "*a*" 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 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 дальнейшего использования или обработки.

Как и массив, она представляет собой совокупность данных. Отличием является то, что к ее элементам необходимо обращаться по имени и что различные элементы структуры не обязательно должны принадлежать одному типу.

Объявление структуры осуществляется с помощью ключевого слова struct, за которым идет ее тип и далее список элементов, заключенных в фигурные скобки:

```
struct тип { тип элемента_1 имя элемента_1;
             .....
           тип элемента_n имя элемента_n;  };
```

Именем элемента может быть любой идентификатор. В одной строке можно записывать через запятую несколько идентификаторов одного типа.

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*/
    floatmid_mark; /* средний балл*/
    char comment[80]; /* поле для комментария*/
};              /* конец блока структуры*/
```

Описание структуры начинается с ключевого слова **struct** и состоит из заключенного в фигурные скобки списка описаний. За словом **struct** может сле-

довать 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.**

В структуре Zurnal находятся данные различных типов, но они объединены в логическую связь. Данные в структуре должны иметь уникальные имена, но в различных структурах можно использовать одинаковые названия.

Следом за правой фигурной скобкой, заканчивающей список элементов, может следовать список переменных, которым присваивается описанный тип:

struct Zurnal {...} a0, a1, a2;

В приведенном выше описании структуры после закрывающей фигурной скобки стоит точка с запятой; она завершает пустой список.

Введенное имя типа позже можно использовать для объявления структуры, например:

struct Zurnal NEW;

Теперь переменная NEW имеет тип Zurnal.

Графически структуру можно представить в виде строки таблицы, где вся строка имеет имя, соответствующее имени структуры, а ее отдельные столбцы содержат имена полей и типы хранящейся в них информации:

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.

Инициализация структуры в языке Си происходит аналогично тому, как инициализируются массивы. Ниже представлен пример инициализации структуры:

```
Zurnal a0 = {11,"Ivanov O.O.",667,87.5,"budget"};
```

Мы создаем переменную с именем a0 типа Zurnal и присваиваем всем пяти полям, которые определенны в структуре, значения.

Обращаться в языке Си к элементам структуры можно двумя способами.

1. ***Через операцию-точку***

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.number = 12;
strcpy(a0.name, "Semenov V.U.");
a0.num_gr = 621;
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);
```

Пример. Структура ДАННЫЕ ПРИБОРА (название прибора, показание прибора, единица измерения).

```
#include <stdio.h>
int main()
{ struct DATA { char name [20];
  float result; char unit[20]; };
  /* объявление структуры* /
  struct DATA a = {"SMB 100A", 2.5 , "dBm"};
  /* инициализация структуры*/
  printf ("result = %f\n",a.result); /* вывод на
```

```
    экран значения элемента result структуры DATA*/
  a.result = 4.6 ; /*присваиваем элементу
    новое значение*/
  printf ("name = %s ; result = %f ; unit = %s
\n",a.name,a.result,a.unit);
  /* вывод на экран всех  элементов*/
  return 0;}
```

2. Через операцию-стрелку

Обозначение стрелки: '->'. Различия заключаются в том, что операция точка используется, когда имеем дело с переменной, а операция стрелка – когда имеем дело с указателем на переменную (см. раздел 5.1). Например:

```
  struct DATA *ptr = &a;
  printf ("%f", ptr-> result);
```

Здесь необходимо ставить знак взятия адреса (&) перед именем структуры, так как имя структуры не является указателем. Также можно обратиться еще и следующим образом:

printf ("%.f", (*ptr). result);

При таком обращении необходимо ставить скобки (*ptr), т.к. операция доступа имеет более высокий приоритет по сравнению с операцией разыменования.

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

Пример. Здесь определяется указатель p_Sergey на переменную Sergey, которая является структурой вида peson (возраст, имя). Используя указатель, можем получить или изменить значения элементов структуры.

```
#include <stdio.h>
struct person
{ int age; char name[20];};
 /* объявление структуры*/
int main( )
{
  struct person Sergey = {18, "Sergey"};
  /* инициализация структуры*/
  struct person * p_Sergey = &Sergey;
  /* указатель на структуру*/
  char * name = p_Sergey->name;
  int age = (*p_Sergey).age;
  /* элементы структуры */
  printf("name = %s ; age = %d \n", name, age);
  /* печать элементов структуры*/
  p_Sergey->age = 19;  /* изменим элемент age в
    структуре*/
  printf("name = %s \t age = %d \n", p_Sergey->name,
Sergey.age);
  /* печать новых элементов структуры*/
  return 0;}
```

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 Zurnal, применив описание Zurnal a[3];

Пример сортировки массива из четырех структур типа Zurnal по возрастанию среднего балла. Инициализация массива структур осуществляется так же, как и инициализация обычного двумерного массива.

```
#include <stdio.h>
int main()
{  int size = 4; int i;
struct  Zurnal  {int number;  char name[20];
 int num_gr;  float mid_mark;
 char comment[80];};
struct  Zurnal bufer; /* вспомогательная структура */
/* ввод данных в массив из четырех структур */
struct Zurnal tel[] =
{{1,"Ivanov O.O.",621,87.5,"budget"},
{2,"Petrov A.S.",621,77.5,"budget"},
{3,"Smirnov P.S.",621,63.4,"not budget"},
{4,"Sidorov G.I.",621,58.9,"not budget"}};
/* Сортировка структур по возрастанию среднего
  балла*/
  int flag = 1;
  while(flag){ flag = 0;
     for(i = 0;i<size-1;i++)
     { struct  Zurnal bufer;
     if(tel[i].mid_mark>tel[i+1].mid_mark)
        {
        bufer=tel[i+1];
        tel[i+1]=tel[i];
        tel[i]=bufer;
        flag = 1;}
  }}
/* Вывод на экран полей структур после сортировки*/
  for(i = 0;i<size;i++)
    { tel[i].number = i+1;
```

```
        printf("number = %d; name is: %s; mid_mark =
%.1f\n", tel[i].number, tel[i].name, tel[i].mid_mark); }
    return 0; }
```

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

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. Ведомость содержит следующие сведения о сдавших вступительные экзамены: Ф.И.О., оценки 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.

2. Имеется база данных, содержащая числители и знаменатели дробных чисел. Например, последовательность 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 (вводится с клавиатуры), и сами числа (результат деления).

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. 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.

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. Создать структуры, содержащие требования на заказ книги в библиотеке: шифр, автор, название; сведения о студенте: номер читательского билета, фамилия, дата заказа. Вывести на экран названия книг, которые заказал определенный студент.

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. Список жителей города содержит следующую информацию: фамилия и инициалы, улица, дом, квартира. Получить Ф.И.О. жителей, проживающих по одному и тому же адресу.

10. Список книг содержит следующую информацию: фамилии авторов, название книги, год издания. Найти и вывести информацию о книгах, в названии которых имеется определённое слово.

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. Описать два комплексных числа и проделать над ними операции сложения, вычитания, умножения и деления.

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. Создать массив структур для учета занятости аудитории: день недели, время пары, аудитория, название предмета. Реализовать поиск периодов времени, когда заданная аудитория свободна.

16. Ведомость успеваемости студентов курса содержит следующую информацию: номер группы, фамилию, средний балл за последнюю сессию. Составить список студентов в порядке возрастания их номеров групп.

17. Создать в программе массив структур, описывающих обозначение поля шахматной доски 8×8 ($a5$, $h8$ и т.п.). Вывести на экран координаты клеток возможных ходов конем с указанной позиции.

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. Дано пять различных дат в виде: число, месяц, год. Вывести их на экран в порядке возрастания

20. Создать структуры, содержащие сведения о книге: шифр, автор, название, год издания, количество страниц. Вывести на экран названия книг года издания ранее, чем указанный, и объемом не менее указанного числа страниц.

3. РИСОВАНИЕ ГРАФИКА ФУНКЦИИ

Изображение на экране монитора формируется из отдельных элементов – пикселов (Pixel – Picture Element – элемент изображения). Видеосистема компьютера поддерживает два режима вывода: текстовый и графический. В текстовом режиме на экран могут выводиться только символы, каждый из которых представляет собой определённую матрицу пикселов. В графическом режиме возможен доступ к отдельным пикселам, что позволяет формировать любые изображения.

Качество изображения на экране монитора зависит как от самого монитора, так и от типа используемого в видеосистеме видеоадаптера. Для получения изображения на экране монитора необходимо поместить в видеопамять определенную информацию. Графическое программирование на этом уровне является непростой и трудоемкой задачей, требующей знания особенностей того или иного видеоадаптера.

Стандарт ANSI C не определяет функции для работы с текстом или графикой в первую очередь потому, что имеется широкое разнообразие аппаратных средств, чем затрудняется стандартизация.

Чтобы сделать процесс создания графических изображений более простым и эффективным, фирма Borland в свое время разработала специальную графическую библиотеку, а также набор графических драйверов, позволяющих работать с различными типами видеоадаптеров. Библиотека, получившая название BGI (Borland Graphic Interface), использовалась в компиляторах языков Pascal и C/C++ фирмы Borland. Графические функции здесь хранятся в библиотеке graphics.lib, а прототипы (объявления) этих функций находятся в файле graphics.h.

В дальнейшем появился визуальный вариант среды для языка C++, который также был создан компанией Borland и получил название C++ Builder. Позже свой взгляд на визуализацию C++ был предложен компанией Microsoft, которая выпустила среду разработки Visual C++.

В настоящее время во всех современных компьютерах применяется графический режим вывода информации, а операционная система Windows не знает символьного режима. Если программа выполняется в среде ОС Windows, то компилятор использует функции графического интерфейса, являющегося частью операционной системы. Точнее, ОС Windows имеет набор библиотечных функций, предназначенных для разработки различных приложений. Эти функции в совокупности называются интерфейсом API (Application Program Interface). Графические функции API Windows объединены в отдельную группу – подсистему GDI (Graphic Device Interface). Достоинством Visual C++ является тесное взаимодействие с операционной системой Windows через функции Win32 API. Пример разработки графического приложения (рисование графика функции) в среде Visual Studio приведён в [6]. При этом используются формы и классы, т.е. такое решение требует хорошего знания C/C++, а также основ объектно-ориентированного программирования.

Для начального освоения программирования графики здесь предлагается использование графической библиотеки WinBGIm (Borland Graphics Interface for Windows) вместе с Microsoft Visual C, как это описано на сайте [7]. В источнике имеется ссылка на архив BGIXXXX.zip, с помощью которого создаётся пустой проект – решение Microsoft Visual C с подключенной библиотекой. Для работы нужно распаковать этот архив в свою рабочую папку, зайти в полученный каталог bgi и двойным щелчком открыть файл bgi.sln. В результате, в среде Visual Studio создаётся проект bgi, в который и следует включать файлы, работающие с графической библиотекой WinBGIm.

При использовании бесплатной среды разработки для C и C++ Code::Blocks алгоритм подключения графической библиотеки будет иным:

1) Скачайте portable версию с сайта *http://codeblocks.codecutter.org*. Распакуйте архив. Для работы запускайте файл "CbLancher.exe" (НЕ "codeblocks.exe"!)

2) Загрузите файлы `graphics.h` и `winbgi.h`, поместите его в папку компилятора MinGW `include` (например `C:\User\CodeBlocks\MinGW\include`)
3) Загрузите файл `libbgi.a` и поместите его в папку ***lib*** (`C:\User\CodeBlocks\MinGW\lib`)
4) В Code::Blocks откройте вкладку *Settings -> Compiler and debugger -> Linker settings*
5) Нажмите кнопку *Add* в *Link libraries* и загрузите файл `libbgi.a`
6) В правой части (*Other linker options*) вставьте команды: `-lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32`
7) Нажмите *OK*

А для того, чтобы подключить модуль для работы с графикой в компиляторе Dev-C++, сделайте следуюшее:

1. Скачайте архив, содержащий файлы `graphics.h` (заголовочный файл) и `libbgi.a` (библиотека).
2. Скопируйте файл `graphics.h` в папку `C:\Dev-Cpp\include`.
3. Скопируйте файл `libbgi.a` в папку `C:\Dev-Cpp\lib`.
4. Запустите оболочку *Dev-C++* и войдите в меню *Сервис -> Параметры компилятора*.
5. Перейдите на вкладку *Компилятор*, включите флажок *Добавить эти команды к командной строке компоновщика* и добавьте в окно под этим флажком строчку:
   `-lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32`

Подключаемая библиотека `graphics.h` содержит графические функции, константы и переменные управления экраном, графические шрифты и др.

Центральной концепцией для функций работы с текстом и графикой служит концепция окна, то есть активной области экрана, в пределах которой осуществляется вывод данных. Окно может быть размером с целый экран, что и задается по умолчанию, либо же иметь такие размеры, какие необходимы.

Borland C++ использует слегка различную терминологию для текстовой и графической систем с целью сохранять эти системы раздельно друг от друга. Текстовые функции работают с окнами, в то время как графическая система использует область просмотра (viewport). Однако концепция в обоих случаях одна и та же. Вся выводимая информация содержится в активном окне. Это означает, что часть изображения, расположенная за пределами окна или области просмотра, автоматически отсекается.

В графическом режиме используется следующая система координат. Пиксел (pixel) с координатами (0,0) находится в левом верхнем углу экрана, при этом ось 0*x* направлена традиционно (слева направо), а вот ось 0*y* – сверху вниз (рис. 3.1).

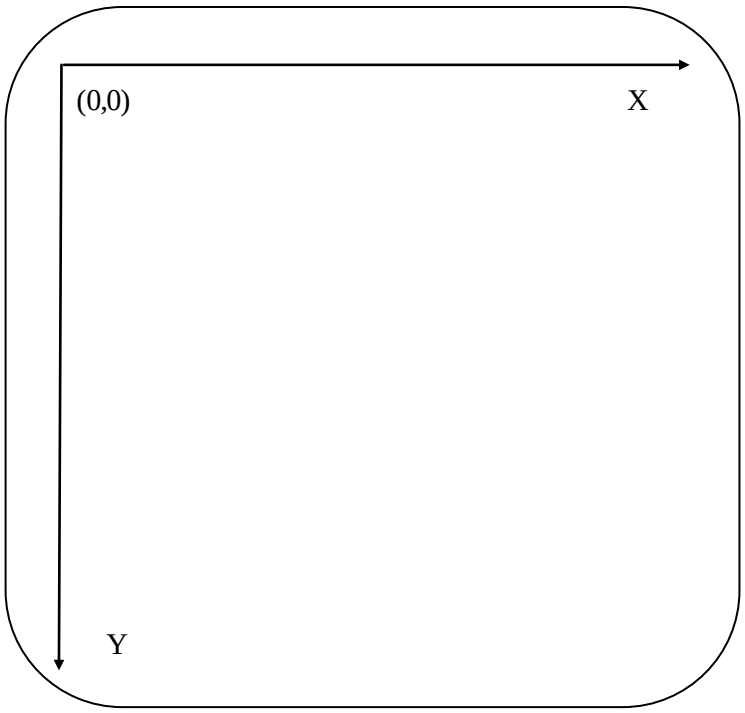


Рис. 3.1. Система графических (экранных) координат

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 `getmaxwidth()` и `getmaxheight()`. Особенностью этих функций является то, что к ним можно обращаться до инициализа-

ции режима рисования, которая осуществляется с помощью функции `initwindow(width, heght)`.

Результатом выполнения этой функции является появление окна отображения графики шириной width и высотой height – все в пикселах; при этом на экране сохраняется и окно с текстовой информацией. Для корректного отображения результатов работы графических функций значения width и height не должны превышать значений, полученных с помощью функций `getmaxwidth()` и `getmaxheight()` соответственно.

В конце блока программы, работающего в режиме рисования, размещают функцию `closegraph()` выхода из режима.

Ниже приводятся некоторые другие графические функции:

`clearviewport()` – функция очистки графического окна;

`putpixel(X, Y, цвет)` – вывод пиксела с координатами X и Y. Цвет пиксела задаётся числом от 0 до 15 или названием цвета:

0	BLACK	4	RED	8	DARK-GRAY	12	LIGHT-RED
1	BLUE	5	MAGENTA	9	LIGHT-BLUE	13	LIGHT-MAGENTA
2	GREEN	6	BROWN	10	LIGHT-GREEN	14	YELLOW
3	CYAN	7	LIGHT-GRAY	11	LIGHT-CYAN	15	WHITE

`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);

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

`rectangle(Xl, Yl, Xr, Xr)` – черчение прямоугольника, который задаётся координатами верхней левой вершины (Xl, Yl) и правой нижней вершины (Yr, Yr);

`circle(X, Y, radius)` – вычерчивание окружности радиусом radius с центром в точке (X, Y);

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

`setbkcolor(color)` – функция, задающая цвет фона формируемого изображения (корректно выполняется после очистки графического окна с помощью функции `clearviewport()`);

`setlinestyle(linestyle, upattern, thickness)` – установка типа и толщины линий и линейных фигур;

Тип линии (linestyle)		Толщина линии (thickness)		
Name	Value	Name	Value	Description
SOLID_LINE	0	NORM_WIDTH	1	1 пиксел
DOTTED_LINE	1	THICK_WIDTH	3	3 пиксела
CENTER_LINE	2			
DASHED_LINE	3			
USERBIT_LINE	4			

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

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

`settextstyle(font, direction, charsize)` – установка стиля текста, выводимого функциями **outtext**(), **outtextxy()**:

Шрифт (font)	Value	Шрифт (font)	Value
DEFAULT_FONT	0	SIMPLEX_FONT	6
TRIPLEX_FONT	1	TRIPLEX_SCR_FONT	7
SMALL_FONT	2	COMPLEX_FONT	8
SANS_SERIF_FONT	3	EUROPEAN_FONT	9

GOTHIC_FONT	4	BOLD_FONT	10
SCRIPT_FONT	5		

direction (направление):

0 – стандартное (слева направо);

1 – вертикально вверх;

2 – справо налево зеркально;

3 – вертикально вниз;

charsize – величина символов – возрастающий размер шрифта задаётся целым числом от 0 до 10;

Полный список графических функций библиотеки WinBGIm Graphics Library можно посмотреть на англоязычном сайте [8].

Отметим, что и координаты пикселов, и большинство других параметров графических функций являются целочисленными по своей сути, и это следует помнить при выполнении влияющих на них математических преобразований.

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 и самой функции.

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

$$Xgr = a1 \cdot x + a2; \qquad (3.1)$$

$$Ygr = b1 \cdot f(x) + b2.$$

Коэффициенты $a1$, $a2$, $b1$, $b2$ находятся исходя из графических координат окна, в котором изображается график. Исходя из поставленной задачи максимального масштабирования, нетрудно получить, что

$$a1 = (xr - xl) / (Xmax - Xmin);$$

$$a2 = xl - a1 \cdot Xmin; \qquad (3.2)$$

$b1$ = (yl – yr) / (Ymax – Ymin);

$b2$ = yl – $b1$·Ymax.

Здесь Xmin и Xmax – границы диапазона изменения аргумента, Ymin и

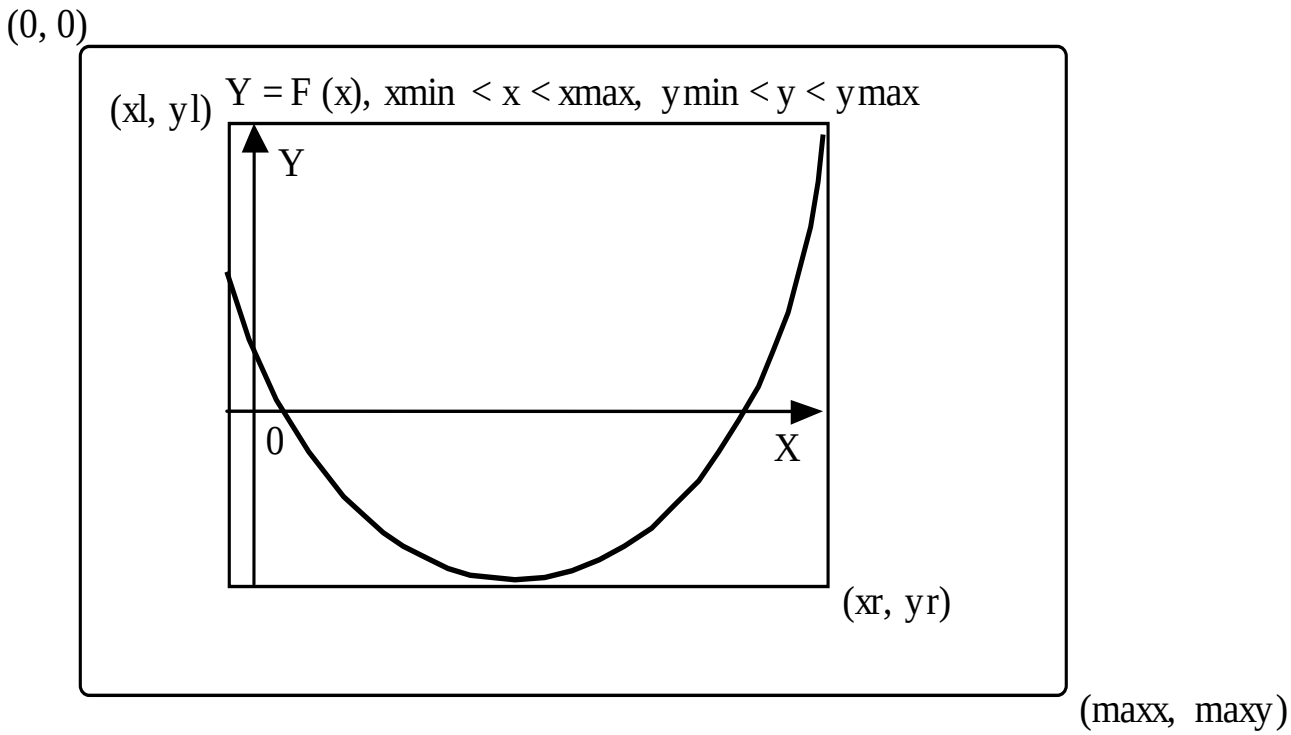


Рис. 3.2. Изображение на экране в заданном координатами (xl, yl), (xr,yr) окне участка (xmin, xmax) графика функции F(x), содержащего точку начала координат

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 отображаемой функции;
3) ввод пределов изменения аргумента функции: Xmin и Xmax;
4) определение с помощью функций `getmaxwidth()` и `getmaxheight()` максимальных графических координат дисплея;

5) вывод на экран максимальных графических координат дисплея;

6) ввод координат прямоугольного окна, в котором должна быть изображена функция. Окно задается экранными координатами верхнего левого угла (xl, yl) и правого нижнего угла (xr, yr);

7) определение (отдельной функцией) максимального и минимального значений функции (Ymin, Ymax) на выбранном участке с учётом того, что максимально возможное число изображаемых точек (пикселов) графика равно xr–xl+1;

8) выполнение действий, связанных с переходом от математических координат к экранным (получение коэффициентов линейного преобразования согласно системы (3.2));

9) инициализация режима рисования (`initgraph());`

10) организация окна вывода изображения функции (`rectangle());`

11) построение графика функции (точками (`putpixel()`) или отрезками прямой (`lineto()`)));

12) вычерчивание оси координат (если они попадают на отображаемый участок функции), нанесение их обозначений и стрелок (графические координаты осей нетрудно получить из системы (3.1));

13) вывод над графиком формулы изображаемой функции, а также пределов изменения аргумента и функции на выбранном участке;

14) организация выхода из режима рисования (`closegraph()`).

Примерный вид результата выполнения программы рисования графика функции приведен на рисунке 3.2.

3.1. Задания на построение графика функции

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. $y = x\cdot\exp(x^{1/2})$;
13. $y = \ln(1- x)$;
14. $y = (1+2x^2) \cdot \exp(x^2)$;
15. $y = \ln((1+x)/(1 - x))$;
16. $y = \sin(x)/x$;
17. $y = 10\cdot\cos(x - 1) + |x|$;
18. $y = x - 2\cdot\sin(5x)$;
19. $y = x - \cos(x) - 1$;
20. $y = 2\cdot|x| + \ln(x+2{,}5) - 5{,}5$.

4. РАБОТА С ФАЙЛАМИ

Хранение данных в оперативной памяти является временным, так как все эти данные теряются при завершении работы программы. При сохранении информация, имеющаяся в памяти компьютера, записывается в файлы для долговременного хранения. Файлы имеют имя (состоящее из названия и возможного расширения, отделяемого точкой) и хранятся на дисках и других внешних запоминающих устройствах. Операционная система и все выполняемые программы также хранятся в файлах, как и любая другая информация, нужная пользователю.

Использование файлов предоставляет следующие возможности и удобства:

- сохранение информации;
- поддержание связи между программами, когда в одной создается, а в другой обрабатывается информация.

Компьютер рассматривает любой файл как последовательность байтов. Каждый файл оканчивается маркером конца файла (EOF).

байт 0	байт 1	байт 2	…			EOF

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

Пример.

```
FILE *file1;        // 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.

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 особо выделяют два вида файлов – текстовые файлы и бинарные файлы. Текстовый файл – файл, содержащий текст, разбитый на строки специальными кодами «возврат каретки» и «перевод строки». Если файл открыт в текстовом режиме, то при чтении из такого файла комбинация символов «возврат каретки – перевод строки» преобразуется в один символ \n – переход к новой строке. При записи в файл осуществляется обратное преобразование. Бинарный файл – файл, из которого байты считываются и выводятся в первоначальном виде без каких-либо преобразований.

Все указанные выше параметры режимов открывают текстовые файлы. Если требуется указать на двоичный файл, то к параметру добавляется буква b. Например: rb, или wb.

Пример.

```
// Объявляем указатели на файлы
FILE * data1, text2;
```

```
// Открываем бинарный файл для записи
data1 = fopen("c:\\data\\data.bin", "wb");
// Открываем текстовый файл для чтения
text2 = fopen("d:\\device\\input.bin", "r");
```

После окончания работы с файлом его требуется закрыть, то есть прервать связь между файлом и программой. Для этого служит функция **`fclose(…)`**, которой необходимо передать указатель на закрываемый файл.

Пример:

```
fclose(data1); //закрывает файл, связанный
 // с указателем data1.
```

Функции работы с файлами:

- `fputc(переменная типа char, указатель на файл)` – посимвольная запись данных в файл.
- `fgetc(указатель на файл)` – посимвольное чтение из файла.
- `fputs(переменная типа string, указатель на файл)` – построчная запись данных в файл. Записывает в файл строку, но в конце не добавляет символ окончания строки.
- `fgets(переменная типа string, 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)` – форматированная запись символов, строк или чисел в файл.
- `fscanf(указатель на файл, строка формата, список переменных)` – форматированный ввод символов строк или чисел из файла.

- `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.
- `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 отсчета)` – 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.

В качестве примеров работы с текстовыми и бинарными файлами рассмотрим следующие задачи:

Задача 1. Необходимо открыть существующий текстовый файл first.txt, посимвольно считать его содержимое и записать это содержимое в новый текстовый 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)
{
  printf("Error opening file c_rec");
  exit(1); /* корректный выход из программы в
    случае ошибки открытия файла first.txt*/
}
if ((f2 = fopen("second.txt","w")) == NULL)
{
  printf("Error opening file b_rec");
  exit(1); /* корректный выход из программы в
    случае ошибки открытия файла second.txt */
}
while(feof(f1)==0)
{
  ch = fgetc(f1); /* посимвольное чтение из
   first.txt и запись в second.txt, пока не
   будет достигнут конец файла*/
  fputc(ch,f2);
  putc(ch);
}
fclose(f1);
fclose(f2);
}
```

Задача 2. В исходном бинарном файле input.dat записаны 30 целых чисел типа int, значения которых лежат в диапазоне от -20 до 20. Необходимо считать данные из файла input.dat и записать все положительные числа в бинарный файл output.dat. Ниже приведена программа, которая позволяет произвести все эти действия.

```
#include <stdio.h>
#define N 30
void main()
{
  int i, n, A[N];
    /*объявляем и открываем файл для чтения*/
  FILE *fp;
  fp = fopen("input.dat","rb");
    /*чтение  блока данных в массив*/
  n = fread(A, sizeof(int), N, fp);
    /*закрываем файл */
  fclose(fp);
    /*объявляем и открываем файл для записи*/
  fp = fopen("output.dat","wb");
    /*запись данных по одному элементу из
    массива*/
  for(i=0;i<N;i++)
{
  if(A[i]>0)
  fwrite(&A[i],sizeof(int),1,fp);
}
/*закрываем файл */
fclose(fp);
}
```

4.1. Задания на работу с бинарными файлами

1. Создать файл, содержащий 10 одномерных целочисленных массивов $a1$, $a2$,..., $a10$, заполненных случайными числами от –50 до 50. Переписать в другой файл те массивы, у которых сумма элементов больше 0.
2. Создать файл, содержащий сведения только о студентах первого курса: Фамилия И.О., группа, оценки за экзаменационную сессию. Затем написать

программу, которая оставляет в файле сведения о тех студентах, которые успешно сдали все экзамены, и записывает в другой файл фамилии студентов, имеющих задолженности.

3. Создать файл **F1**, компонентами которого являются целочисленные массивы *a*1,.., *a*10, заполненные случайными числами, как положительными, так и отрицательными, закрыть его. Затем считать массивы из файла, преобразовать их и записать в файл **F2**. Преобразовать массивы следующим образом: положительные элементы заменить на 1, отрицательные на -1, а нулевые оставить без изменения.
4. Создать файл **F1**, компонентами которого являются целочисленные массивы *a*1,.., *a*10, закрыть его. Затем считать массивы из файла и записать в файл **F2**, расположив элементы четных массивов в обратном порядке.
5. Создать файл **f**, компонентами которого являются 10 целочисленных массивов *a*1,.., *a*10, заполненных случайными числами. Требуется преобразовать каждый из массивов, заменив наибольший элемент нулем. Полученные массивы должны быть записаны в тот же самый файл **f**. Разрешается использовать вспомогательный файл **g.**
6. Создать файл **f**, содержащий сведения о книгах: фамилию автора, название, год издания. Другой программой найти название книг данного автора, изданных с 1960 г., определить, имеется ли книга с названием, например, "Информатика", если да, то сообщить фамилию автора и год издания. В случае если таких книг несколько, сообщить сведения обо всех этих книгах.
7. Создать файл **f**, компонентами которого являются целочисленные массивы *a*1,.., *a*10, заполненные случайными числами. Требуется преобразовать каждый из массивов, поменяв местами максимальный и минимальный элементы. Полученные массивы должны быть записаны в тот же самый файл **f.** Разрешается использовать вспомогательный файл **g.**
8. Создать файл, содержащий сведения о студентах: фамилию, номер курса, номер группы. Затем написать программу, которая вычисляет, на сколько

человек на 1 курсе больше, чем на втором. Записать в файл **gr*.dat** сведения о студентах * группы (вместо * поставьте номер Вашей группы).

9. Создать файл **f**, содержащий сведения о веществах в следующей последовательности: название вещества, температура кипения, удельный вес. (Для отладки программы достаточно записать сведения о нескольких веществах). Используя сведения из файла **f**, записать в файл **g** расширенные сведения в таком порядке: фазовое состояние (газ, жидкость, твердое тело), название вещества, удельный вес, температура кипения.
10. Создать файл **f**, который содержит целые числа (для их генерации используйте функцию rand()). Затем написать программу, которая суммирует каждый восьмой элемент файла, начиная с первого.
11. Файл **Finp** содержит матрицу из $m \times n$ целых чисел. Записать в файл **Fout** k-ю строку матрицы (значения m, n, k определить в начале программы через const). Примечание: решить задачу с минимальным количеством операций считывания из файла.
12. Файл **Finp** содержит матрицу A действительных чисел размерностью 4×4. Из матрицы A получить матрицу B по следующему правилу: $b_{00}=a_{00}$, $b_{01}=(a_{00}+a_{01})/2$, $b_{02}=a_{01}$, $b_{03}=(a_{01}+a_{02})/2$, $b_{04}=a_{02}$, $b_{05}=(a_{02}+a_{03})/2$, $b_{06}=a_{03}$ (для последующих строк аналогично, меняется только первый индекс: $b_{10}=a_{10}$,...). Записать матрицу B в файл **Fout**.
13. Файл **Finp** содержит матрицу A действительных чисел размерностью 8×4. Из матрицы A получить матрицу B по следующему правилу: $b_{00}=(a_{00}+a_{01})/2$, $b_{01}=(a_{02}+a_{03})/2$, $b_{02}=(a_{04}+a_{05})/2$, $b_{03}=(a_{06}+a_{07})/2$ (для последующих строк аналогично, меняется только первый индекс). Записать матрицу B в файл **Fout**.
14. Файл **Finp** содержит матрицу A действительных чисел размерностью 8×8. Из матрицы A получить транспонированную матрицу B. Записать матрицу B в файл **Fout**.
15. Создать файл **Finp.dat**, который содержит 3 матрицы из $m \times n$ случайных целых чисел. Записать в файл **Fout** k-е столбцы матрицы (значения m, n, k определить в начале программы через const).

16. Создать файл с информацией занятости аудиторий: номер аудитории, время, предмет. Переписать эту информацию в другой файл, дополнив каждую запись фамилией преподавателя.
17. Создать файл **f**, содержащий имена людей с их адресами. Обеспечить в программе следующий сервис: при наборе имени человека на экране распечатывается его адрес.

4.2. Задания на работу с текстовыми файлами

1. Дан текстовый файл **f**, содержащий целые числа от 1 до 100. Подготовить новый файл для печати этих чисел в две колонки. В левой колонке должны быть размещены числа от 1 до 50, в правой колонке – числа от 51 до 100. Выровнять числа по левому краю в каждой колонке.

2. Дан текстовый файл **f**, содержащий сведения о сотрудниках учреждения, записанные по следующему образцу: Фамилия Имя Отчество. Записать эти сведения в файле **g**, используя образцы: 1) Имя Отчество Фамилия; 2) Фамилия И.О. Выбор варианта преобразования задаётся в начале работы программы.

3. Дан текстовый файл **f**, содержащий произвольный текст. Слова в тексте разделены пробелами и знаками препинания. Получить 10 наиболее часто встречающихся слов и число их появлений.

4. Даны два текстовых файла **f1** и **f2**. Файл **f1** содержит произвольный текст. Слова в тексте разделены пробелами и знаками препинания. Файл **f2** содержит два слова. Первое слово считается заменяемым, второе слово – заменяющим. Найти в файле **f1** все случаи употребления первого слова и заменить его на второе слово. Преобразованный текст поместить в файл **g.**

5. В текстовый файл вывести первые 12 строк «треугольника Паскаля». В этом «треугольнике» крайние числа равны 1, а каждое внутреннее – сумме двух ближайших расположенных над ним.

					1					
				1		1				
			1		2		1			
		1		3		3		1		
	1		4		6		4		1	
1		5		10		10		5		1
	6		15		…		…		…	
…		…		…		…		…		…

6. В текстовый файл вывести картинку, изображающую умножение «столбиком» двух заданных натуральных чисел. Возможный пример:

```
39624
      ×
 8503
----------
  118872
 + 198120
 + 316992
------------------
 336922872
```

7. Дан текстовый файл **f**, содержащий программу на языке СИ. Проверить текст программы построчно на несоответствие числа открывающих и закрывающих круглых скобок, вывести на экран номера строк с несоответствием и сами строки.

8. Дан текстовый файл **f.** Переписать строки файла **f** в файл **g** в перевернутом («зеркальном») виде. Порядок строк в файле **g** должен совпадать с порядком исходных строк в файле **f.**

9. Дан текстовый файл **f**. Записать строки файла **f** в файл **g** в перевернутом («зеркальном») виде. Порядок строк в файле **g** должен быть обратным по отношению к порядку строк исходного файла.

10. Даны текстовый файл **f** и строка **s**. Получить все строки файла **f**, содержащие в качестве фрагмента строку **s.**

11. Даны два текстовых файла **f1** и **f2**. В текстовый файл **f3** поместить текст из файла **f1** со вставкой текста из файла **f2** после *n*-й строки файла **f1.**

12. Создать с помощью редактора СИ текстовый файл **ft**, содержащий *n* строк по *m* целых чисел в строке (числа в строке записать с 1-й позиции через пробел). Переписать числа из текстового файла **ft** в файл с данными **fd.**

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

14. Дан текстовый файл **f**. Создать текстовый файл **g**, в котором будут сохранены только те слова из файла **f**, которые встречаются в нем впервые. Таким образом, в файле **g** все слова будут разными.

15. Используя в качестве внешнего текстового файла файл, содержащий программу на языке СИ, составить программу выравнивания текста «по ширине» и вывести его в другой текстовый файл. Принять длину строки в 40 символов.

16. Используя в качестве внешнего текстового файла файл с программой на языке СИ, составить программу, которая подсчитывает в тексте количество вхождений заданного оператора и выводит номера содержащих его строк.

17. Используя в качестве внешнего текстового файла файл с программой на языке СИ, составить программу выравнивания текста «по центру» (вставляя пробелы) и вывести его в другой текстовый файл. Принять длину строки в 40 символов.

5. РАБОТА С УКАЗАТЕЛЯМИ И ДИНАМИЧЕСКОЙ ПАМЯТЬЮ

5.1. Общие сведения об указателях

Указатель – это переменная, значением которой служит адрес некоторой величины. Адрес – это номер байта в памяти, где располагаются переменные и функции программы.

При помощи указателей вы можете самостоятельно распоряжаться памятью компьютера, при условии, что вы 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;

Основная операция над указателями - операция получения адреса переменной &.

Пример:

```
int *pin, nurse;

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

```
//переменной nurse
```

5.2. 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. */
```

Указатели часто используются как параметры в функции. Указатель может быть возвращен функцией. Примером функции, возвращающей значение типа указатель, может быть функция fopen().

5.2. Массивы и указатели

В языке Си имя массива является константой типа указатель, значение которой равно адресу первого элемента этого массива. Это позволяет передавать массив в функцию через указатель, не копируя его в стековую память. Поэтому имя массива может использоваться как указатель, и наоборот.

Пример передачи массива в функцию через указатель.

```
#include <stdio.h>
// Функция вычисления суммы с параметрами – целым числом
// элементов dim и указателем на массив arr
float total( int dim, float* arr)
```

```
{
      int i;
      float sum;
      for (sum=0, i=0; i<dim; i++)
      sum += arr[i];
      return sum;
}
//+++++++++++++++++++++++++++++++++++++++++++
int main()
{
      float a[4]={1.,1.,1.,1.}, total_a;
      float b[3]={1.,1.,1.}, total_b;
/* вызываем функцию для вычисления суммы 4-х элементов
массива a */
      total_a = total(4, a);
/* вызываем функцию для вычисления суммы 3-х элементов
массива b */
      total_b = total(3, b);
      printf("%f, %f", total_a, total_b);
      return 0;
}
```

Когда указатель связан с массивом, можно пользоваться операциями инкремента ++ или декремента --. Прибавляя к указателю единицу (или вычитая ее), мы получаем адрес последующего (предыдущего) элемента массива.

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.

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
```

5.3. 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>.

Пример объявления указателей и использования их в программе:

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

```
int main()
{
  int n;
  int *a; // объявляем переменную-указатель
  //вводим размер динамического массива
  printf("Input size of array: ");
  scanf("%d", &n);
  //выделяем память для динамического массива
  a = (int *) malloc(n * sizeof(int));
  //заполняем динамический массив данными
  for (int i=0; i<n;i++)
   {a[i]=i;}
  //освобождаем память
  free(a);
  return 0;
}
```

Для выполнения заданий достаточно будет этих трех функций процедур. С работой остальных функций по работе 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:

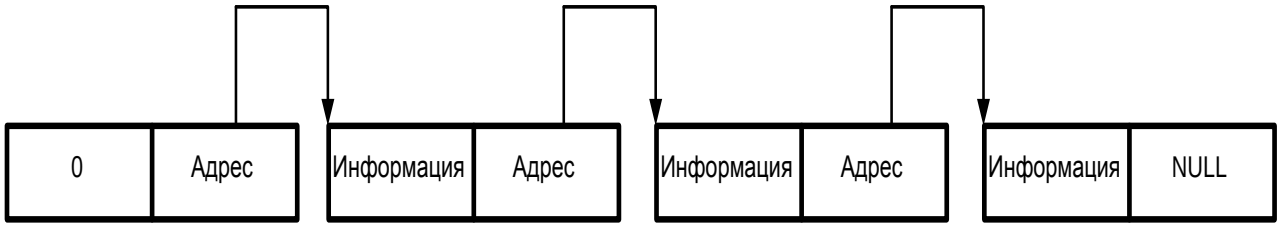
int *VectXP[100];

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 (например, $S(x)=52\cdot x^{40}-3\cdot x^{8}+x$) в виде однонаправленного списка. Если какой-то из коэффициентов $a_i = 0$, 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, в следующем виде: P(x)=52x^40-3x^8+x.

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

$T_0(x) = 1$;

$T_1(x) = x$;

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

по способу, предложенному в задании 3. Вывести вид произвольного *i*-го (где $0 <= i < n$) многочлена Чебышева $T_n(x)$.

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 = 70;
/*длина строки <= 80*/
const unsigned num = 20;
/* 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 больше числа строк в обрабатываемом файле; при этом последним элементам массива str, не указывающим на строки, присвоить значение NULL. Разместить в памя-

ти, 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 местами.

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. Практикум по программированию на языке Си для физиков и радиофизиков. Учебно-методическое пособие / Журавлев А.А., Ильдиряков В.Р., Мамедова Л.Э., Стенин Ю.М., Фахртдинов Р.Х., Хуторова О.Г. – Казань: Казанский университет, 2013.
2. Царев, Р. Ю. Программирование на языке Си [Электронный ресурс] : учеб. пособие / Р. Ю. Царев. – Красноярск : Сиб. федер. ун-т, 2014. – 108 с. - Режим доступа: http://znanium.com/catalog/product/510946
3. Язык Си: кратко и ясно: Учебное пособие [Электронный ресурс] / Д.В. Парфенов. - М.: Альфа-М: НИЦ ИНФРА-М, 2014. - 320 с. - Режим доступа: http://znanium.com/bookread2.php?book=459254
4. Программирование на СИ#: Учебное пособие / Медведев М.А., Медведев А.Н., - 2-е изд., стер. - М.:Флинта, Изд-во Урал. ун-та, 2017. - 64 с.
5. Язык Си++ : учебное пособие для студентов высших учебных заведений, обучающихся по направлениям "Прикладная математика" и "Вычислительные машины, комплексы, системы и сети" / В. В. Подбельский . 5-е изд. Москва : Финансы и статистика, 2008 . 559 с.
6. Бахвалов, Н.С. Численные методы. [Электронный ресурс] : учеб. пособие / Н.С. Бахвалов, Н.П. Жидков, Г.М. Кобельков. М. : Издательство 'Лаборатория знаний', 2015. 639 с. – Режим доступа: http://e.lanbook.com/book/70767
7. Демидович Б.П. Основы вычислительной математики./ Демидович Б.П. , Марон И.А.; – М.: изд. Лань, 2009. – 664 с.
8. Керниган Б. Язык программирования Си./ Керниган Б., Ритчи Д. — 2-е изд. – М.: «Вильямс», 2007. – 304 с.
9. Турчак Л.И. Основы численных методов. / Турчак Л.И., Плотников П.В. М.:ФИЗМАТЛИТ, 2002. – 304 с.