КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ФИЗИКИ

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

Практикум по программированию на языке Си

для физиков и радиофизиков

Часть 2

Учебно-методическое пособие

КАЗАНЬ – 2013

УДК 681.924

Печатается по решению Редакционно-издательского совета ФГАОУВПО «Казанский (Приволжский) федеральный университет»

Учебно-методического совета Института физики КФУ
Протокол № 2 от 13 мая 2013 г.

заседания кафедры радиоастрономии
Протокол № 12 от 23 мая 2013 г.

Рецензент
доктор физико-математических наук
О.Н. Шерстюков

Журавлев А.А., Ильдиряков В.Р., Мамедова Л.Э., Стенин Ю.М., Фахртдинов Р.Х., Хуторова О.Г.

Практикум по программированию на языке Си для физиков и радиофизиков. Часть 2. 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. – 45 с.

Данное учебно-методическое пособие предназначено для проведения занятий по практическому курсу программирования, обязательного к изучению в Институте физики. Пособие содержит теоретический материал по методам численного решения задач, некоторые начальные сведения по программированию на языке Си и практические задания для самостоятельной реализации.

Последовательность изложения теоретического материала и практических заданий представлена в порядке выполнения практических работ по курсам «Информатика», «Программирование», «Численные методы и математическое моделирование» студентами Института физики КФУ, обучающихся по направлениям физика и радиофизика.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данное учебно-методическое пособие является продолжением [1] и предназначено для выполнения практических работ в курсе информатики студентов первого курса второго семестра Института физики. Предполагается освоение студентами особенностей программирования задач, связанных с обработкой строк, использованием структурного типа данных, созданием и изменением текстовых и бинарных файлов, использованием динамической памяти. Решается задача построения графика функции на плоскости. В качестве среды программирования предполагается использование Visual Studio. Часть предлагаемых задач взята из пособия [2], ориентированного на выполнение заданий в среде Турбо-Паскаль.

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 в данном примере 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. Например, литерал `"*"` обозначает два байта: первый байт содержит код символа звездочки, второй байт содержит ноль. Константа `'*'` обозначает один байт, содержащий код звездочки.

Строку приведённого выше примера обычно записывают с помощью литерала:

```
char str[15]=”Hello!”;
```

Инициализировать строку можно и так:

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

Размер строки как 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.

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

Копировать 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);
```

Присоединить 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);
```

Сравнить 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);
```

Определить 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.

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 длину самого короткого слова.
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 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 "*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 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.

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, которым присваивается описанный тип. Вот почему в приведенном выше описании структуры после закрывающей фигурной скобки стоит точка с запятой; она завершает пустой список.

Но можно, используя указанное выше описание **`Zurnal`,** с помощью отдельной строки, например такой:

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

описать структурные переменные `a0, a1, a2`, каждая из которых строится по шаблону структурного типа `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 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, в этом случае за определением следует список начальных значений элементов:

```
Zurnal a0 = {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.number = 12;
strcpy(a0.name, "Semenov V.U.");
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 `Zurnal`, применив описание

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

Теперь в программе можно выполнять, например, следующие действия:

```
strcpy(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

```
Zurnal *uk;
```

значит, что `uk` – указатель на структуру типа `Zurnal`. Выборка элемента структуры здесь может выполняться с помощью операции «стрелка» `'->'` (набирается двумя символами), например: `uk->number`. Таким образом, следующие обращения к полю структуры эквивалентны:

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

В выражении `(*uk).number` скобки обязательны, так как приоритет операции выделения элемента `"."` выше, чем у `"*"`.

Если указатель связывается с массивом структур, то 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.

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 издание, 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.

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, работавших в среде операционной системы DOS. Последним из них явился популярный компилятор Borland C++ 3.1 (1992). Графические функции здесь хранятся в библиотеке graphics.lib, а прототипы (объявления) этих функций находятся в файле graphics.h.

В 1997 году 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 BGI2008.zip (а также на BGI2010.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 (решение Microsoft Visual Studio 2008 (2010)). В результате, в среде Visual Studio создаётся проект bgi, в который и следует включать файлы, работающие с графической библиотекой WinBGIm.

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

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).

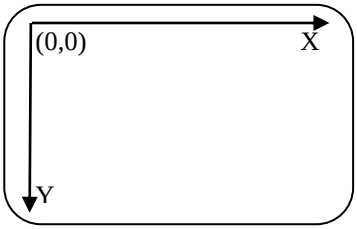


Рис. 3.1. Система графических (экранных) координат

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 инициализации 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	DARKGRAY	12	LIGHTRED
1	BLUE	5	MAGENTA	9	LIGHTBLUE	13	LIGHTMAGENTA
2	GREEN	6	BROWN	10	LIGHTGREEN	14	YELLOW
3	CYAN	7	LIGHTGRAY	11	LIGHTCYAN	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, yr**)** – черчение прямоугольника, который задаётся координатами верхней левой вершины (xl, yl) и правой нижней вершины (xr, yr);

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

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

setbkcolor(цвет**)** – 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;

delay(time**)** – функция приостанова вычислений на время time, выраженное в мс (1 с = 1000 мс).

Полный список графических функций библиотеки WinBGIm Graphics Library, пригодных для использования совместно с Visual Studio, 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:

$$\begin{aligned} Xgr &= a1 \cdot x + a2; \\ Ygr &= b1 \cdot f(x) + b2. \end{aligned} \qquad (3.1)$$

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

$$\begin{aligned} a1 &= (xr - xl) / (Xmax - Xmin); \\ a2 &= xl - a1 \cdot Xmin; \end{aligned} \qquad (3.2)$$

$b1$ = (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 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;
10) рисуется окно вывода изображения функции (rectangle());
11) рисуется график функции (точками (putpixel()) или отрезками прямой (lineto()));
12) чертятся оси координат (если они попадают на отображаемый участок функции), наносятся их обозначения и стрелки (графические координаты осей нетрудно получить из системы (3.1));

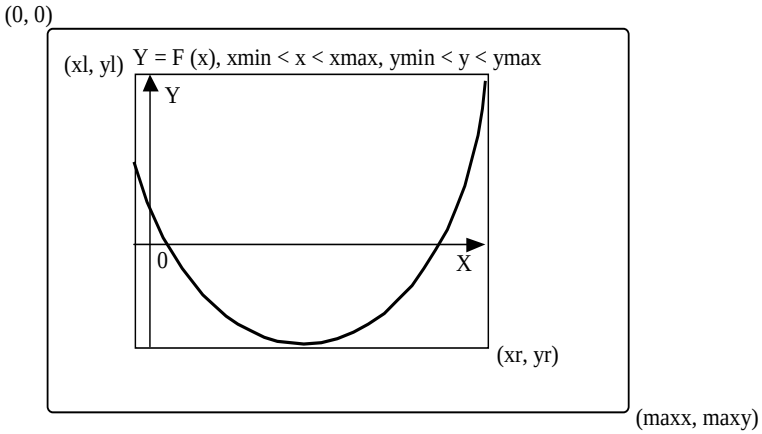


Рис. 3.2. Изображение на экране в заданном координатами (xl, yl), (xr,yr) окне участка (xmin, xmax) графика функции F(x), содержащего точку начала координат

13) над графиком выводится формула изображаемой функции, а также пределы изменения аргумента и функции на выбранном участке;

14) организуется выход 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 переменных и массивах 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. 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 объёме данных;
- программа может самостоятельно помещать данные в файл.

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

Также многие компиляторы СИ позволяют открывать файлы одновременно и для чтения и для записи. Для этого в аргументе функции `fopen()` указываются следующие параметры:

`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 (в случае существования файла – для дозаписи в его конец).

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`, или `r+b`. В некоторых компиляторах текстовый режим обмена обозначается буквой `t`, т.е. записывается `a+t` или `rt`.

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

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.
- **fwrite**(указатель на буфер хранения данных, размер элемента, количество элементов, указатель на файл) – запись заданного количества блоков данных определѐнной длины из буфера в файл.
- **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.
- **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 языке Си определены два стандартных файла ввода-вывода, к которым можно обращаться с помощью следующих указателей:

- stdin – указатель на стандартный поток ввода. По умолчанию ему ставится в соответствие клавиатура.

- stdout – указатель на стандартный поток вывода. По умолчанию ему соответствует экран дисплея.

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)
{
  fprintf(stdout, "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)
{
  fprintf(stdout, "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);
  fputc(ch,stdout);
}
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];
 FILE *fp;
 fp = fopen("input.dat","rb");
 n = fread(A, sizeof(int), N, fp); /*чтение
  блока данных в массив*/
 fclose(fp);
 fp = fopen("output.dat","wb");
 for(i=0;i<N;i++)
 {
    if(A[i]>0)
       fwrite(&A[i],sizeof(int),1,fp); /*запись
  данных по одному элементу из массива*/
 }
 fclose(fp);
}
```

4.1. Задания на работу с бинарными файлами

1. Создать файл, содержащий 10 одномерных целочисленных массивов *a*1, *a*2,..., *a*10, 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

Следующим шагом в овладении программированием на Си является использование указателей. Указатели 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. Любую серьёзную программу на языке Си (и не только на языке Си, но и на любом другом языке программирования) очень трудно 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 *. Примеры оператора объявления переменных *a* и *b* типа `int` и переменной `p` – указателя на величину типа `int` приведены на рис. 5.1. В результате последующих операций указателю `p` с помощью операции & (см. п. 5.2) присваивается значение адреса переменной `a` (`0xf34`), хранящей значение 3.

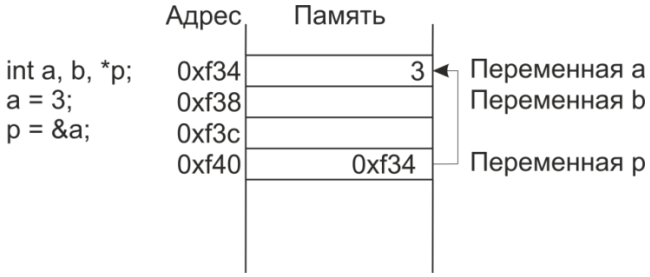


Рис. 5.1. Пример объявления и использования переменной-указателя (`p`)

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 0x23f4. Значение
           переменной увеличится не на 1, а на 4 – количество байт,
           занимаемое типом int в машинной памяти. Т. е., при
           сложении указателя с целым числом происходит не простое
           арифметическое сложение, а перемещение указателя на
           данное типа  int, отстоящее от nurse на количество
           таких же по типу данных, равное этому целому числу. */
pin++;/* Переменная pin изменит свое значение с 0x23f4 на 0x23f8,
         т.е. будет указывать на следующее данное типа int. */
```

Иллюстрация выполненных операций сложения и инкремента приведена на рис. 5.2.

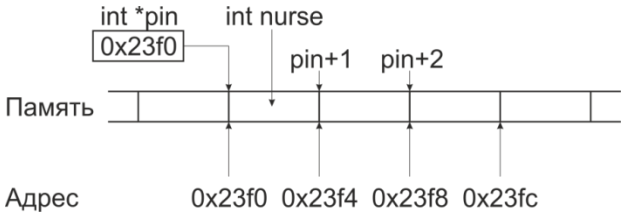


Рис. 5.2. Перемещение указателя при сложении и инкременте

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 функции, возвращающей значение 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 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:

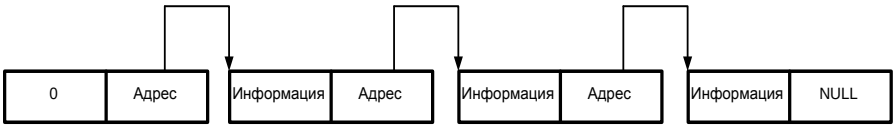
```
int *VectXP[100];
```

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:

$T_0(x) = 1$;

$T_1(x) = x$;

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

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 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 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. Хуторова О. Г., Стенин Ю. М., Фахртдинов Р. Х., Зыков Е. Ю., Журавлев А. А. Практикум по программированию на языке Си. Часть 1. - Казань: Казанский университет, 2012. – 46 с.
2. Журавлев А. А., Стенин Ю. М., Хуторова О. Г. Практикум по информатике. Часть 2. - Казань: КГУ, 2007. – 28 с.
3. Подбельский В. В., Фомин С. С. Программирование на языке Си. - М.: Финансы и статистика, 2003. - 600 с.
4. Подбельский В. В. Язык C++. - М.: Финансы и статистика, 2003. - 562 с.
5. Александров Э. Э. Программирование на языке С в Microsoft Visual Studio 2010 / Э. Э. Александров, В. В. Афонин. – Саранск: Изд-во Мордов. ун-та, 2010. – 428 с.
6. Пахомов Б. И. C/C++ и Visual C++ 2010 для начинающих. – СПб.: БХВ-Петербург, 2011. – 736 с.
7. Сайт: http://www.cs.colorado.edu/~main/bgi/visual/.
8. Сайт: http://www.cs.colorado.edu/~main/bgi/doc/.
9. Абрамов С. А., Гнездилова Г. Г., Капустина Е. Н., Селюн М. И. Задачи по программированию. - М.: Наука, 1988. – 224 с.
10. Оверленд Брайан. C++ без страха. - М.: Изд-во Триумф, 2005. – 432 с.
11. Дейтел, Х. М. Как программировать на С / Х.М. Дейтел, П. Дж. Дейтел. - М.: Изд-во Бином, 2000.
12. 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.
13. Жешке Р. Толковый стандарт языка Си. - СПб.: Питер, 1994. - 223 с.
14. Керниган Б. Язык программирования Си./ Керниган Б., Ритчи Д. - М.: «Вильямс», 2007. – 304 с.
15. Белецкий Ян. Энциклопедия языка СИ./ Под ред. Ф.Ф.Пащенко. – М: Мир, 1992, - 686 с.
16. Р. Хазфилд, К. Лоуренс и др. Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста / Р. Хэзфилд, Л. Кирби и др. – М.: Изд. ДиаСофт, 2001. - 736 с.
17. Шилдт Г. Полный справочник по С. - М.: Вильямс, 2004. - 704 с.