Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Воронежский государственный архитектурно-строительный университет»

О.В. Курипта, О.В. Минакова, Д.К. Проскурин

ОСНОВЫ ПРОГРАММИРОВАНИЯ И АЛГОРИТМИЗАЦИИ

Практикум

Воронеж 2015

УДК 004.4
ББК 32.973.3
К931

Рецензенты:
кафедра информационных технологий моделирования и управления Воронежского государственного университета инженерных технологий;
Н.Б. Горбачева, канд. физ - матем. наук, доцент Российского экономического университета им. Г.В. Плеханова (Воронежский филиал)

К931 **Курипта, О.В.**
ОСНОВЫ ПРОГРАММИРОВАНИЯ И АЛГОРИТМИЗАЦИИ : практикум / О.В. Курипта, О.В. Минакова, Д.К. Проскурин; Воронежский ГАСУ. – Воронеж, 2015. – 132 с.

ISBN 978-5-89040-575-3

Приводятся основные теоретические сведения, примеры программ, описание выполнения лабораторных заданий, индивидуальные задания необходимые для получения навыков разработки алгоритмов и программ на языке Си, а также контрольные задания для самостоятельной работы.

Практикум предназначен для студентов, обучающихся по направлению подготовки 09.03.02. - «Информационные системы и технологии», а также для студентов вузов, магистрантов, интересующихся программированием.

Ил. 17. Табл. 8. Библиогр.: 15 назв.

УДК 004.4
ББК 32.973.3

Печатается по решению учебно - методического совета Воронежского ГАСУ

ISBN 978-5-89040-575-3

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Получение навыков программирования на языке высокого уровня, является неотъемлемой часть подготовки ИТ-специалиста. Начиная с первого курса необходимо не только научить студента использованию основных языковых конструкций, но и привить хороший стиль программирования. Под стилем программирования подразумевается набор приемов или методов программирования, которые позволяют писать правильные, эффективные, удобные для применения и легкочитаемые программы. Реализации этой цели служит данный практикум.

Каждая тема включает необходимый теоретический материал, приемы и правила эффективного использования рассматриваемых языковых конструкций и структур данных, примеры программ, практические задания для отработки теоретического материала и контрольные задания для закрепления полученных в ходе аудиторных занятий навыков. Структуризация материала практикума направлена на получение практических навыков разработки программ на языке Си, начиная от анализа условия задачи и заканчивая оформлением документации по реализованной программе.

Материал практикума охватывает изучение основ языка Си – переменные, типы данных, арифметические и логические операции, операторы управления и цикла, функции и передача параметров, включает организацию ввода-вывода, структурную декомпозицию, механизмы передачи параметров и использования структур данных – массивов, записей, строк.

В ходе подготовки к выполнению работы изучение теоретического материала не достаточно, следует анализировать и понимать поведение представленных примеров программ, стараться их модифицировать и расширить.

Выполнение каждой лабораторной работы ставит перед студентом ряд задач – спроектировать, реализовать, протестировать и отладить программу,

которая использует рассматриваемую конструкцию программирования, метод или структуру данных. В ходе решения поставленной задачи необходимо выбирать подходящие операторы и типы данных, применять типовые приемы и структуры данных, использовать эффективные методы построения кода.

При решении кажой посьавленной здачи рекомендуется следующая последовательность выполнения работ:

– анализ условия задачи и выбор пути решения;

– пошаговая разработка решения (словесная детализация) или описание алгоритма решения в виде блок-схемы;

– кодирование алгоритма на языке Си;

– отладка программы и демонстрация ее работы на тестовом наборе данных.

По результатам выполнения работы оформляется отчет, который должен содержать:

– тему и цель работы;

– условия задачи;

– пошаговое описание или блок-схему алгоритма решения;

– листинг программы с комментариями;

– скриншот консоли с результатами выполнения программы на тестовом наборе данных.

При защите отчета студент должен продемонстрировать:

– понимание текста программы и умение внести изменения в программу с учетом дополнительных условий;

– навыки редактирования и отладки программы в выбранной инструментальной среде программирования.

Систематическое последовательное выполнение представленных работ будет способствовать формированию профессиональных компетенций у студентов направления «Информационные систем и технологии» и создаст базу для успешного освоения специальных дисциплин.

ЛАБОРАТОРНАЯ РАБОТА № 1
СОЗДАНИЕ, ОТЛАДКА И ЗАПУСК КОНСОЛЬНОГО ПРИЛОЖЕНИЯ

Цель работы: Написание консольного приложения на языке Си
Программные средства: MICROSOFT VISUAL STUDIO

1.1 Теоретические сведения

Решение любой задачи с использованием ЭВМ требует выполнения следующих шагов:

1. Постановка задачи – необходимо сформулировать, что дано и что требуется найти, и определить полный набор исходных данных, необходимых для получения решения.

2. Формализация задачи – составление математической формулы или разработка математической модели.

3. Построение алгоритма и составление программы на языке программирования.

5. Отладка и тестирование программы.

6. Проведение расчетов и анализ полученных результатов.

С помощью языка программирования создается текст программы (его называют исходным кодом или source code), описывающий выбранную или построенную математическую модель. Чтобы написанная на языке Си программа была выполнена, надо весь ее текст перевести в машинный код и затем передать на исполнение процессору. Это действие и выполняет специальная программа — компилятор.

В ходе обработки текста программы современные компиляторы осуществляют поиск синтаксических ошибок, выполняют семантический анализ и только затем, если текст программы в точности соответствует правилам языка, его автоматически переводят (транслируют) на машинный язык. Нередко при этом выполняется оптимизация с помощью набора методов, позволяющих повысить быстродействие программы. В ходе процесса компиляции генерируют объектный код или object code. Сгенерированный объектный код обрабатывается специальной программой – сборщиком или редактором связей (компоновщик – linker), который производит его связывание с кодами заданных библиотек. В ходе процесса компоновки формируется готовый к исполнению EXE-файл или исполнимый код с расширением bin, его можно сохранить в памяти компьютера или на диске (рис. 1.1). Этот файл имеет самостоятельное значение и может работать под управлением той операционной системы, для которой была выполнена компиляция и компоновка. Его можно перенести на другие компьютеры с процессором, поддерживающим соответствующий машинный код или соответствующей архитектурой.

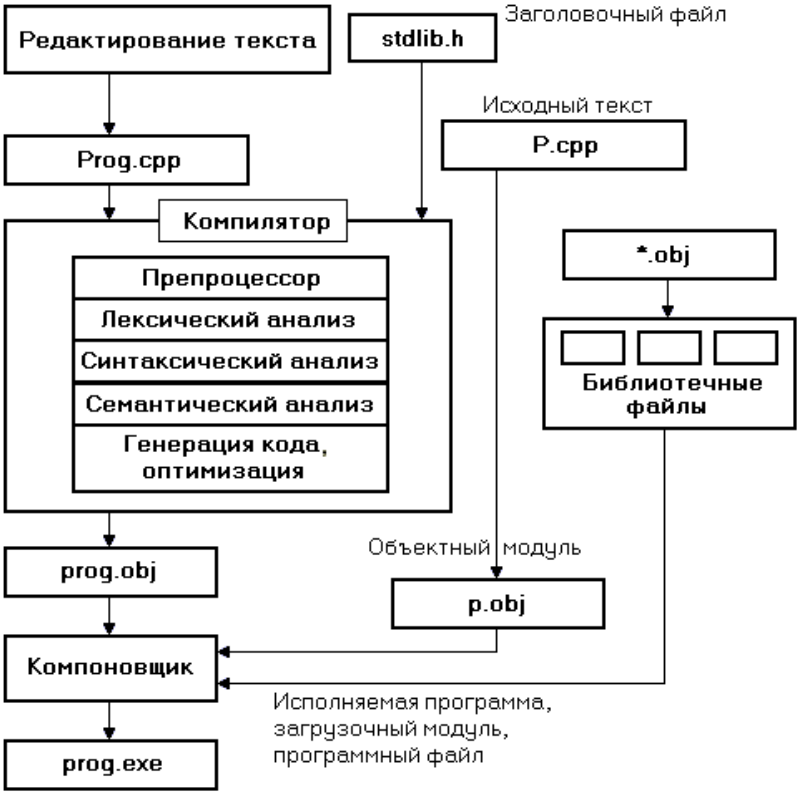
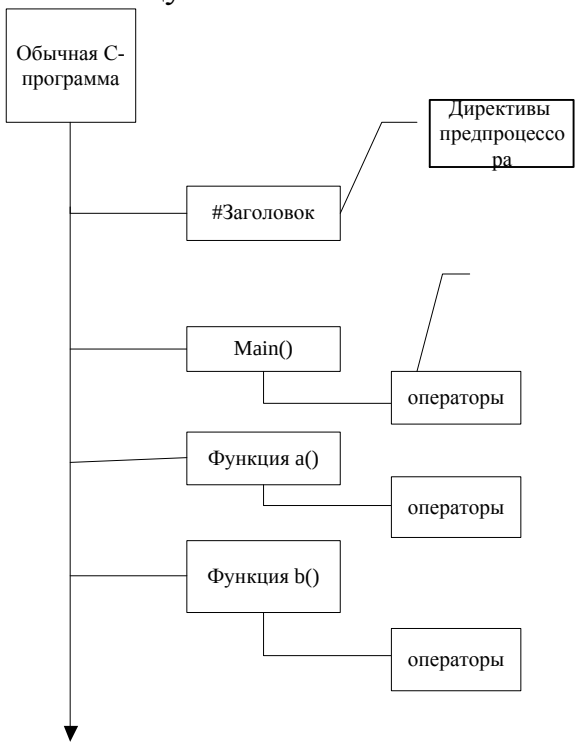


Рис. 1.1. Процесс создания программы

Текст программы на языке Си имеет определенную структуру (рис. 1.2), включающую:



1. заголовок;
2. включение необходимых заголовочных файлов;
3. определение используемых макроконстант;
4. объявление глобальных переменных;
5. определение обязательной функции main();
6. описание других функций используемых в программе.

Рис. 1.2. Структура программы

Для отображения результатов и изменения исходных данных необходимо организовать ввод/вывод. Наиболее простой способ – использование экранного

буфера или консоли. Консоль – это объект операционной системы, создаваемый по запросу приложения и состоящий их входного буфера для записи информации о событиях нажатия и отпускания клавиш и активного экранного буфера, представляющего двухмерных массив из 80 на 25 символов. Для работы с консолью используются функции языка Си, предназначенные для работы со стандартными потоками, детально рассматриваемые в следующих темах.

1.2 Приемы, используемые при отладке программ

В некоторых ситуациях необходима приостановка выполнения программы. Задержку можно осуществить следующими способами:

1. Использование функции gethchar(), которая ожидает ввода символа с консоли. Ввод любого символа приведет к переходу на следующую строку кода, что продолжит выполнение программы. А пока программа ждет ввода, ее выполнение как бы останавливается. Пример представлен в листинге 1.

2. Вызов служебной функции операционной системы с командой задержки выполнения:

```
system("pause");
```

Прототип описан в stdlib.h, поэтому для корректной работы требуется включение заголовочного файла #include <stdlib.h >. Вызовможет быть осуществлен передзавершение программы – оператором return или функции exit().

Если нет необходимости останавливать программу, то можно использовать функцию exit(), прототип которой определен в заголовочном файле stdlib.h. Выполнение оператора exit(0); немедленно завершает программу, закрывает все открытые файлы и выполняет некоторые другие завершающие действия. Значение в круглых скобках возвращается ОС или другой вызывающей программе для анализа и использования.

1.3 Пример разработки консольного приложения – вывода текста

Программа, написанная на языке С, состоит из одной или нескольких функций, одна из которых обязательно имеет идентификатор (имя) ***main*** – основная, главная. Ее назначение – управление всей работой программы (проекта).

Наиболее ощутимые отличия консольного приложения от всех других – это организация ввода-вывода данных.

Для вывода информации в консольном приложении чаще всего используются функции puts(…) и printf(…), которым в качестве аргумента передается строка текста, заключенная в двойные кавычки. Описания функции puts(char *) не требуется, так как она объявлена в заголовочном файле stdio.h и находится в стандартной библиотеке языка Си.

Следовательно, основные шаги по реализации задачи вывода текста на

консоль следующие:

1. Включение заголовочного файла stdio.h.

2. Написание функции main(), включающей единственное действие – вывод заданной строки с использованием функции puts(..).

В результате программа имеет вид:

`#include <stdio.h>`	Включение заголовочных файлов
`void main()`	Главная функция программы
`{`	Начало главной программы-функции
`puts ("Моя программа");`	Вывод строки на консоль путем вызова функции `puts(…)` с параметрами `"моя программа"` типа строка
`}`	Конец главной программы-функции

Полный текст программы с локализацией поддержки русского язык, представлен на листинге 1.

Листинг 1.– Программа вывода сообщения на консоль

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

void main()
{

 setlocale(LC_ALL, "RUS"); // для переключения русской кодировки
 puts ("Моя программа"); // вывод строки
 getchar(); // задержка экрана
}
```

1.4 Контрольные задания

Напишите программу, которая выводит тему лабораторной работы и информацию об ее исполнителе (группа, студент).

Реализуйте форматирование темы работы по центру консоли, а информации об исполнителе – слева к краю.

Окружите текст рамкой из символов -* или – или >.

ЛАБОРАТОРНАЯ РАБОТА № 2
ФОРМАТИРОВАННЫЙ ВВОД/ВЫВОД ЧИСЕЛ

Цель работы: Получение навыков написания консольного приложения с использованием функций вывода данных

Программные средства: MICROSOFT VISUAL STUDIO

2.1 Средства вывода и ввода в консоль

Вывод символов на консоль осуществляется в языке Си помощью функции printf(...).Эта функция может иметь переменное число аргументов, но первым всегда является управляющая строка. Она содержит:

–обычные символы, которые непосредственно выводятся на консоль;

– управляющие символьные константы (таблица 2.1), которые определяют расположение выводимых элементов;

– спецификаторы, начинающиеся с символа % и заканчивающиеся некоторым символом, задающим преобразования (таблица 2.2), которые вызывают вывод на консоль значения очередного аргумента из последующего списка переменных.

Таблица 2.1

Управляющие символьные константы

\n	Новая строка, перевод строки
\r	Возврат каретки
\t	Горизонтальная табуляция
\v	Вертикальная табуляция
\"	Двойная кавычка
\\	Обратная косая черта
\?	Знак вопроса

Общий вид спецификатора формата вывода следующий:

%[флаг][ширина][.точность][h|l|L]тип,

где **флаг** – символ, указывающий на особенности заполнения полей вывода (таблица 2.3);

ширина– десятичное число, которое указывает минимальную ширину поля (включая знак для чисел);

точность – число, которое указывает:

– на минимальное количество символов, которое должно появиться при обработке типов d, i, o, u, x, X;

– на минимальное количество символов, которое должно появиться после десятичной запятой (точки) при обработке типов a, A, e, E, f, F;

– максимальное количество значащих символов для типов g и G;

– максимальное число символов, которые будут выведены для типа s.

тип – символьная константа, указывающая на типы выводимой переменной (таблица 2.2).

Для форматированного ввода данных с консоли используется функция scanf (..), которая сканирует клавиатуру, определяя какие клавиши нажаты. А затем интерпретирует вводимые данные в соответствии с указанной управляющей строкой и списком параметров (аргументов функции). Аргументы функции scanf должны быть указателями на соответствующие значения (перед именем записывается символ &). Управляющая строка содержит спецификации преобразования (рис. 2.1), используемые для

установления количества и типов аргументов. В нее могут включаться: пробелы, символ табуляции и перехода на новую строку.

Таблица 2.2

Спецификация преобразования переменных

%c	символ
%d или %i	целое десятичное число со знаком
%e или %E	десятичное число в виде x.xx e+xx
%f или %F	десятичное число с плавающей запятой xx.xxxx
%g или %G	%f или %e, с выбором по принципу, что короче
%o	восьмеричное число
%s	строка символов
%u	беззнаковое десятичное число
%x или %X	шестнадцатеричное число
%lf или %lg	длинное десятичное число

Таблица 2.3

Спецификатор вывода флаг

Знак	Значение	В отсутствие этого знака
-	выводимое значение выравнивается по левому краю в пределах минимальной ширины поля	по правому
+	всегда указывать знак (плюс или минус) для выводимого десятичного числового значения	только для отрицательных чисел
□*	помещать перед результатом пробел, если первый символ значения не знак	вывод может начинаться с цифры.
0	дополнять поле до ширины, указанной в поле *ширина* управляющей последовательности, символом 0	дополнять пробелами

*)символ, означающий пробел

На время выполнения функции scanf (..) выполнение приостанавливается, и программа ожидает ввода указанных параметров. Завершение ввода осуществляется нажатием клавиши Enter.

Функция scanf (..) может быть использована для одновременного ввода нескольких параметров, но если они будут введены не в правильной последовательности, то введенная информация может быть проигнорирована, поэтому рекомендуется для каждого ввода вызвать отдельную scanf (..).

Прототипы функций printf и scanf описаны в заголовочном файле **stdio.h**.

2.2 Приемы форматированного вывода информации

Использование функции printf позволяет оформить вывод значений

переменной, выражений и даже других функций. Так, например, следующая функция выводит значения переменных и результат операции над ними:

```
printf ("\tОстаток от деления %d на %d равен \n %d ",a, b, a%b);
```

Аргументами этой функции являются управляющая строка, переменные a и b, выражение – a%b.

Управляющая строка											Список параметров
"\t	Остаток деления	от	%d	на	%d	равен	\n	%d "			,a, b, a%b
табуляция	Обычный текст		Преобразование целого типа	текст	Преобразование целого типа	текст	Перевод строки	Преобразование целого типа			1) a, 2) b, 3) a%b - -

Рис.2.1. Использование функции printf

Можно использовать эту функцию для вывода по частям.

Только вывод строки:

```
printf ("Остаток от деления \n");
```

Результаты вычисления выражения **a%b** в виде целого числа:

```
printf ("%d", a%b);
```

Совмещение вывода строки и двух значений переменных a и b в виде десятичного числа с одной и двумя цифрами после запятой:

```
printf ("Умножение %.1fна %.2f", a, b);
```

2.3 Примеры форматированного вывода информации

Пример 1. Вывести сумму и произведение двух целых чисел.

```
#include<stdio.h>                    Включение заголовочных файлов
voidmain()                           Главная функция программы
{                                    Начало главной программы-функции
int a, b, res;                       Объявление переменных целого типа для данных и результата
puts (введите два числа");           Вывод приглашения к вводу данных
scanf ("%d %d", &a, &b);             Считывание введенных пользователем значений по адресу переменных a и b
res=a+b;                             Присваивания переменной res значения результата операции сложения чисел a и b
printf("Сумма равна: %d\n", res);    Вывод строки, содержащей значение переменной res
printf("Произведение %d на %d равно %d\n",a, b, a*b);   Вывод строки, включающей значение переменных a и b, а также результата операции умножения a и b
}                                    Конец главной программы-функции
```

Пример 2. Использование различных спецификаторов в функции printf()

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

/* выводит два раза строку "Это текст" в 20-символьном поле с выравниванием по левому краю */

```
printf("%-20s%-20s\n", "Это текст", "Это текст");
```

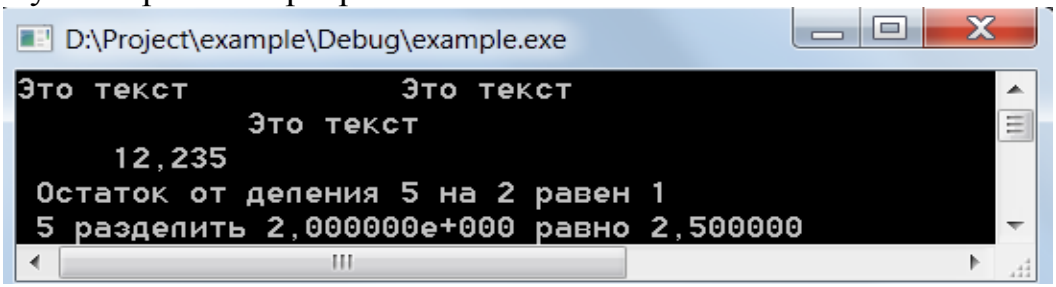
/* выводит строку "Это текст" в 20-символьном поле без выравнивания */

```
printf("%20s\n ", "Это текст");
```

/* выводит вещественное значение с 3 цифрами после запятой в 10-символьном поле. */

```
printf("%10.3f\n", 12.234657);
printf ("Остаток от деления %d на %d равен %d\n ",5, 2, 5%2);
printf ("%gразделить %e равно %f\n ",5., 2., 5./2);
return 0;
}
```

Результат работы программы:



Пример 3. Использование ввода и вывода

```
#include<stdio.h>
void main(void)
{
char a;
int b;
float c;
doubled;
printf("введите символ, целое число, вещественное и очень маленькое число");
scanf(“%c %d %f %lf”, &a, &b, &c, &d);
printf("символ - %c, целое число - %d,\n вещественные числа .3%f и %lf",a, b, c, d);
}
```

2.4 Практические задания

1. Запишите следующий код:

```
int k=-1;
float x=4.23;
```

```
double z=7.12E-3;
```

Выведите их на экран с помощью функции printf().

```
printf("%d%f%lf",k,x,z);
```

2. Поскольку все данные выведены слитно, полученный результат невозможно прочитать. Добавьте в управляющую строку функции printf() после вывода каждой переменной управляющий символ ‘/n’ и добейтесь вывода результатов в отдельные строки.

3. Добавьте поясняющий текст для каждой переменной, например:

```
printf("k=%d/n",k);
```

4. Установите параметры управляющей строки «%6d, %8.1f, %-20.6lf» и выведите значения k, y, z.

5. Выведите значения k, y, zc точностью одна, две и три цифры после запятой соответственно и длиной поля вывода 10 знакомест (используйте спецификацию формата %10.3f).

6. Напишите программу для вычисления значения среднего арифметического **n** вводимых с консоли вещественных чисел (используйте объявление *float x*) с выводом значений с **k** цифрами целой части и **m** – дробной. (Приветствие, приглашение к вводу данных и указание на результат обязательно)

Вариант	***n***	***k***	***m***	***Вариант***	***n***	***k***	***m***
1	2	3	1	***7***	2	2	4
2	3	4	2	***8***	3	3	5
3	4	5	3	***9***	4	2	6
4	2	3	3	***10***	2	3	7
5	3	4	2	***11***	3	2	2
6	4	5	1	***12***	4	3	1

2.5 Контрольные задания

Используя единственную функцию printf(), нарисуйте с помощью какого-нибудь символа (* или +) геометрическую фигуру:

1) прямоугольник;
2) трапеция;
3) треугольник остроугольный;
4) ромб.

При реализации обязательно выведите название фигуры.

ЛАБОРАТОРНАЯ РАБОТА №3
БАЗОВЫЕ ТИПЫ ДАННЫХ

Цель работы: Знакомство с базовыми типами данных и закрепление навыков организации стандартного ввода/вывода.

Программные средства: MICROSOFT VISUAL STUDIO

3.1 Теоретические сведения

Основными элементами языка программирования являются – алфавит, идентификаторы, операторы и ключевые слова.

Алфавит языка Си включает:

– прописные и строчные латинские буквы и знак подчеркивания;

– арабские цифры от 0 до 9;

– специальные знаки, например, {, %, # и т.д.

– управляющие символы: пробел, символы табуляции, символы перехода на новую строку.

Из символов алфавита формируются лексемы языка:

– идентификаторы (имена программных объектов);

– ключевые (зарезервированные) слова;

– знаки операций (символы, определяющие действия над операндами);

– константы (неизменяемые величины);

– разделители (скобки, точка, запятая, пробельные символы).

Операторы программы служат для описания её алгоритма. Конструкция оператора определяет, какие действия должны быть выполнены по обработке данных и\или передаче управления от оператора к оператору. В соответствии с правилами построения различают составные операторы или блоки, структурные операторы и простые операторы.

Простые операторы могут выражать некоторое законченное действие и завершаться знаком**;** (точка с запятой).

Блоки (составные операторы) начинаются с открывающей фигурной скобки **{**и заканчиваются парной закрывающей фигурной скобкой **}**. Они используются в качестве тел функций (например, тело функции main()), могут быть частями структурных операторов или иметь самостоятельное значение.

Обрабатываемые данные хранятся в ячейках оперативной памяти, а переменные и константы представляют их в исходном тексте программы. Отличие переменной от константы состоит в том, что переменным можно присваивать новые значения, а значения констант задают только в исходном тексте программы и нельзя изменять их при её выполнении.

Для объявления переменных используется конструкция вида:

<тип переменной><имя>

Основными типами переменных языка Си являются:

int – целый, длиной 2-4 байта, диапазон значений -32768 ... +32767 (при двухбайтном представлении);

char – символьный длиной 1 байт, его можно рассматривать как целое -128...+127 (иногда 0...255);

float – тип данных с плавающей точкой, длиной 4 байта, вещественное число с диапазоном значений от $\pm 8.4 \cdot 10^{-37}$ до $\pm 3.3 \cdot 10^{38}$ и шестью значащими цифрами;

double – тип данных с плавающей точкой, длиной 8 байт, вещественное

число с диапазоном значений от $\pm 2.2 \cdot 10^{-308}$ до $\pm 1.8 \cdot 10^{308}$ и четырнадцатью значащими цифрами.

В свою очередь, данные целого типа могут быть короткими (**short**) и длинными (**long**), со знаком (**signed**) и беззнаковыми (**unsigned**). Атрибут **long** может использоваться и с типом **double**– длинное вещественное.

С перечисленными типами данных допустимы следующие арифметические операции:

+	Прибавляет величину, находящуюся справа, к величине, стоящей слева
-	Вычитает величину, стоящую справа, из величины, указанной слева
*****	Умножает величину справа на величину, находящуюся слева
/	Делит величину, стоящую справа, на величину – слева. Результат усекается, если оба операнда целые числа

Результат вычисления передается другой переменной с использованием оператора присваивания:

Y = выражение;

Действие выполняется справа налево, т. е. сначала вычисляется ***выражение***, а затем его результат присваивается указанной переменной ***Y.***

3.2 Приемы оформления кода

Хороший код должен быть красиво оформлен.

Правило 1. Написание комментариев

Использовать комментарии нужно обязательно, но не внутри операторов.

Комментарии никак не влияют на создание исполняемой программы и просто пропускаются компилятором. Назначение комментариев состоит в том, чтобы облегчить понимание алгоритма программы при чтении её исходного текста. Оформить текст в виде комментария можно одним из следующих способов:

1) короткий текст – поместив символы // перед текстом строки,

```
//оператор getch(); заставит программу getch();
//ждать нажатия клавиши
```

2) текст, занимающий несколько строк – заключив текст в скобки вида /* и */, например,

```
/*************************************************/
/*       функция оформления текста               */
/*************************************************/
```

Правило 2. Определение переменных и функций

При составлении имен желательно придерживаться какого-либо стиля – CamelCase, under_scope, MixedCase и др.

Распространенным в языке Си является стиль, основанный на использовании символа подчеркивания для разделения слов в именах переменных и функций.

Все объекты программы должны быть названы обоснованными именами, говорящими о назначении того или иного объекта.

Переменные следует писать строчными буквами. Но не следует использовать символ подчеркивания в начале имени, так как это предусмотрено для служебных констант и функций.

Пример.

counter, increment - счетчик

name, lastName - имя, фамилия

home_phone – домашний телефон

Константы рекомендуется писать в верхнем регистре.

Пример.

SIZE, POINT, BOOK

Для именования переменных нужно использовать существительные, а для именования функций – глаголы (либо глагол + существительное), т. к. это действие.

Названия функций должны отражать возвращаемое значение.

Пример.

cos(x) – вычислить косинус

play Song() – воспроизвести музыку

print Array()– вывести значения массива

Шестнадцатеричные константы следует писать заглавными буквами. При этом префикс константы должен быть написан строчной буквой ‘x’.

Пример.

0xFF1234AB

При указании типа данных для целочисленных констант или констант с плавающей точкой суффиксы f, l, u и e следует писать строчными буквами.

Пример.

0.5f

10e10

Правило 3. Форматирование кода

Каждое объявление данных необходимо размещать на отдельной строке. Группировать следует только по логической взаимосвязи.

Пример.

```
double radius;
double sq_round;
intx1,y1;//координат точек
```

Операторы (+, –, *, /, = и т.п.) следует отделять от операндов одним пробелом. Логический оператор отрицания (!) следует писать слитно с операндом.

Длина строк программы не должна превышать ширины экрана (80 символов).

3.3 Пример конструирования программы

Пример 1. Вывод информации о допустимых диапазонах представления переменных целого типа.

Минимальные и максимальные значения переменных заданы в заголовочном файле limits.h в виде переопределения #define, вида:

```
#define SHRT_MIN (-32768) /* minimum (signed) short
#define SHRT_MAX 32767 /* maximum (signed) short
#define USHRT_MAX 0xffff /* maximum unsigned short
```

Поэтому в программе достаточно использовать эти определения.

```
#include < stdio.h >   //Для printf  и puts
#include < limits.h >
#include < locale.h >
int main (void)
{
setlocale(LC_ALL, "RUS");
printf ("Основные типы данных\n");
printf ("Целые числа типа int от %ддо %d.\n",
INT_MIN, INT_MAX);
printf ("Целые числа типа short от %ддо
%d.\n",INT_MIN, INT_MAX);
printf ("Целые числа типа long от %ld до
%ld.\n",LONG_MIN , LONG_MAX);
printf ("Целые числа типа longlong от %lld до
%lld.\n",LLONG_MIN , LLONG_MAX);

system("pause");
   return 0;
}
```

Результаты работы программы представлены ниже на рис.3.1.

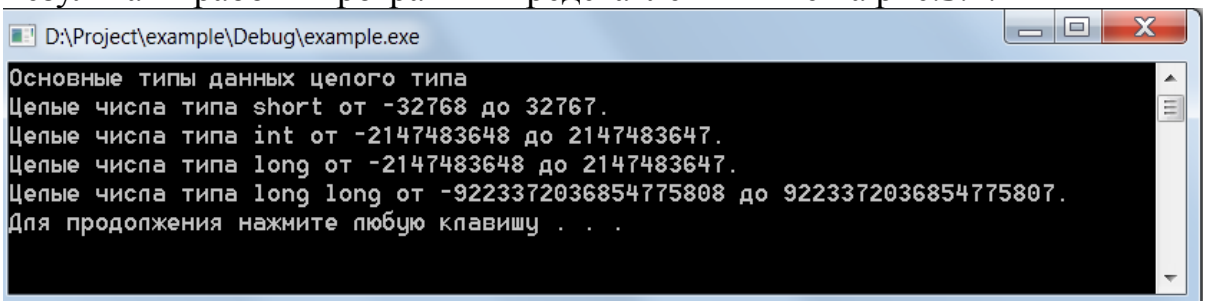


Рис.3.1. Результат работы программы

Обратите внимание на использование специальных спецификаторов формата для вывода переменных типа long.

Пример 2. Ввод и вывод значений различного типа.

```
int main()
{
char c1;
```

```
    short s1;
    int i1;
    long L1;
    float f1;
    double d1;
    printf("input char >"); scanf("%c",   &c1); printf("
%c\n",  c1);
    printf("input short>"); scanf("%hd", &s1); printf("
%hd\n", s1);
    printf("input int  >"); scanf("%d",   &i1); printf("
%d\n",  i1);
    printf("input octal>"); scanf("%o",   &i1); printf("
%o\n",  i1);
    printf("input hexa >"); scanf("%x",   &i1); printf("
%x\n",  i1);
    printf("input long >"); scanf("%ld", &L1); printf("
%ld\n", L1);
    printf("input float>"); scanf("%f",   &f1); printf("
%f\n",  f1);
    printf("input double>"); scanf("%lf", &d1);printf("
%lf\n", d1);
    return 0;
    }
```

3.4 Практические задания

1. Объявите переменную x и присвойте ей значение 1.

Пример: int x=1;

Выведите с помощью функции `printf («Один - %d», x)` результат.

2. Аналогично выполните инициализацию y=2 и z=3. В следующих строках с использованием спецификаторов формата функции `printf` выведите значение «два» и «три» (лесенкой), так чтобы получилось:

```
Один –1.
        Два – 2.
                Три – 3.
```

3. Выведите в обратном порядке значения x, y, z с точностью одна, две и три цифры после запятой соответственно.

4. Реализуйте программу пересчета целых дюймов в сантиметры и метры (английский дюйм равен 2,54 см) по следующим шагам:

А) установите постоянное значение D с помощью директивы препроцессору:

#define D 2.54

Б) объявите переменную для вводимого пользователем значения:

```
int dym;
```

В) с помощью функции print() предложите ввести значение для расчета;

Г) считайте значение с помощью функции scanf(..):

```
scanf ("%d", &dym);
```

Д) используйте функцию printf() и выведите результат как D *dym с точностью до двух цифр после запятой и указанием единиц измерения «см»:

```
printf ("%d дюймов – это %.1f см ", dym, dym*D*1.f);
```

5. Напишите программу вывода текущего времени, для этого

А)организуйте ввод часов (0<N<24);

Б)организуйте ввод значений минут (0<K<60);

В) Выведите результат в виде сообщений:

1	«Сейчас N час K мин»
2	«Местное время N:K»
3	«Идет R минута суток»
4	«R часов M минут до полуночи»
5	«M минут после полудня»
6	«NчKмин – текущее время»

3.5 Контрольные задания

Напишите программу, предназначенную для вычисления:

1) площади боковой поверхности круглого цилиндра по заданному радиусу и высоте;

2) силы тяжести при падении с заданной высоты тела заданной массы;

3) заданного в секундах временного интервала в часах и минутах (<60);

4) возраста по году рождения;

5) числа световых лет из астрономических единиц.

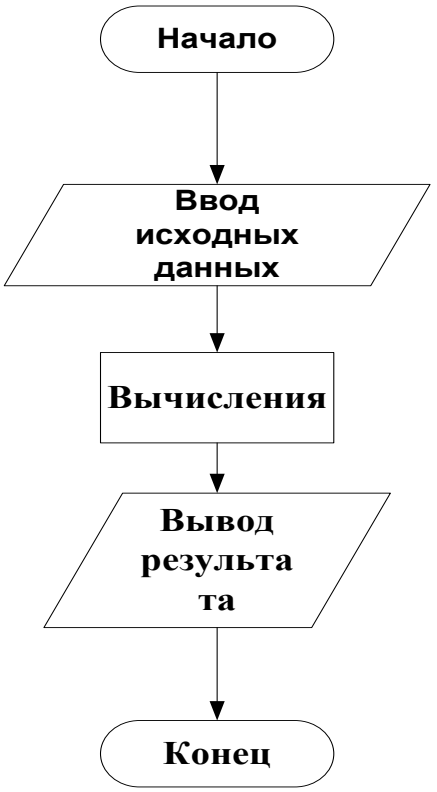
ЛАБОРАТОРНАЯ РАБОТА №4
ЛИНЕЙНЫЙ ВЫЧИСЛИТЕЛЬНЫЙ ПРОЦЕСС

Цель работы: Получение практических навыков построения линейных программ с использованием оператора присваивания, арифметических операций и математических функций.

Программные средства: MICROSOFT VISUAL STUDIO

4.1 Теоретические сведения

Программы с линейной структурой (рис. 4.1) являются простейшими и используются для однократного выполнения заданной последовательности действий при любых значениях исходных данных. В таких программах операторы выполняются последовательно, один за другим, в соответствии с их расположением в программе. Основное место в них занимают операторы присваивания = и выражения, которые содержат операции и операнды.



Формы оператора присваивания

Обычная запись	Сокращенная запись
x=x+y	x+=y
x=x/(y+10)	x/=y+10
x=x%2	x%=2

Различают бинарные операции с двумя операндами и унарные. Бинарные записываются в обычной математической форме, знак унарной предшествует операнду. Правила выполнения операций соответствуют приоритету.

В языке Си существуют операции ***инкремента*** (--) и ***декремента*** (++), т. е. уменьшения или увеличения значения переменной на 1. Операции могут быть ***префиксные*** (++*i* и --*i*) и ***постфиксные*** (*i*++ и *i*--).

При использовании данной операции в выражении в префиксной форме, сначала выполняется сама операция (изменяется значение *i*), и только потом вычисляется выражение.В постфиксной форме – операция применяется после вычисления выражения, например, для значений $b = 7$ и $n = 1$ будут получены следующие результаты:

1) $c = b*++n$; – порядок выполнения: $n = n+1$, $c = b*n$, т.е. $c = 14$;

2) $c = b*n++$; – в этом случае: $c = b*n$, $n = n+1$, т.е. $c = 7$.

!Обратите внимание на приоритет операций и при необходимости расставляйте скобки.

Кроме операций при обработке данных может быть использован вызов библиотечных или пользовательских функций. Вызов представляет собой указание имени вызываемой функции, за которой в круглых скобках следует список аргументов. Например:

```
res=pow(x,3)+5// вызов функции pow
```

Наиболее часто использующиеся ***стандартные математические функции*** описаны в таблице 4.1.

При их использовании обязательно включение заголовочного файла библиотеки математических функций с использованием директивы:

```
#include <math.h>
```

Таблица 4.1

Стандартные математические функции языка Си

Математическая функция	***ID функции***	***Математическая функция***	***ID функции***
$\sqrt{x}$	sqrt(x)	arcsin(x)	asin(x)
|x|	fabs(x)	arctg(x)	atan(x)
e^x	exp(x)	arctg(x/y)	atan2(x,y)
x^y	pow(x, y)	sh*(x)=1/2 (e^x-e^{-x})	sinh(x)
ln(x)	log(x)	ch(x)=1/2 (e^x+e^{-x})	cosh(x)
lg_{10}(x)	log10(x)	tgh(x)	tanh(x)
sin(x)	sin(x)	Остаток от деления x на y	fmod(x,y)
cos(x)	cos(x)	Округление к большему	ceil(x)
tg(x)	tan(x)	Округление к меньшему	floor(x)

4.2 Приемы, используемые при вычислениях

Последовательность выполнения операторов в выражении определяется их приоритетами и порядком их следования (таблица 4.2), а при необходимости изменения порядка вычислений используют круглые скобки.

В выражениях в первую очередь вычисляются обращения к функциям и содержимое круглых скобок, затем – унарные операции изменения знака (-), затем - операции типа умножения (*, /, %) в порядке слева направо, затем – операции типа сложения (+ и -) в порядке слева направо.

Если двуместная операция выполняется над вещественными данными разных типов (float и double), то результат будет иметь тип, представляющий больший диапазон значений (double).На рис. 4.1 представлены арифметические преобразования, гарантирующие сохранение точности и неизменность численного значения.

Таблица 4.2

Приоритет арифметических операций

Приоритет	***Операция***	***Порядок выполнения***
1. первичные	**вызов функции ()**	слева направо
2.унарные	**изменение знака -**	справа налево
	инкремент ++	
	декремент --	
	преобразование типа (тип)	
3.*мульти-пликативные*	**умножение ***	слева направо
	деление /	
	остаток от деления %	

* Синус гиперболический, а в следующей строке – косинус гиперболический.

Приоритет	***Операция***	***Порядок выполнения***
4. *аддитивные*	**сложение +**	слева направо
	вычитание -	
5. *присваивание*	= += -= *и т.д.*	справа налево

long ← ***int*** ← ***short*** ← ***signed char***

long double ← ***double*** ← ***float***

Рис. 4.1. Корректное преобразование типов

Также следует обращать внимание, что в выражениях преобразование операнда целого типа к вещественному выполняется автоматически, если другой операнд вещественный, но если требуется преобразовать вещественное в целое, не используя присваивания, необходимо применить явное преобразование типа: непосредственно перед операндом в круглых скобках записать имя целого типа.

```
res=5+(int) tan(y);
```

Особое внимание следует обращать на тип аргументов и возвращаемое значение стандартных функций. Так функция pow(..) имеет несколько сочетаний типов параметров, но ни одно не допускает использование в качестве первого аргумента целого типа (рис.4.2.).

```
double pow(double _X, double _Y)

double pow(double _X, int _Y)

float pow(float _X, float _Y)

float pow(float _X, int _Y)

long double pow(long double _X, long double _Y)
(+1 overloads)
```

Рис. 4.2 Прототипы pow

Сокращенные формы записи и некоторые другие приемы позволяют значительно сократить количество строк в программе. Так объявление переменной можно совместить с присваиваниями ей численного значения, т.е. инициализировать:

```
int k = 10, m = 3, n;
double c = -1.3, w = -10.23, s;
```

Одним выражением можно присвоить значения сразу нескольким переменным:

```
z=y=x=1;
```

или

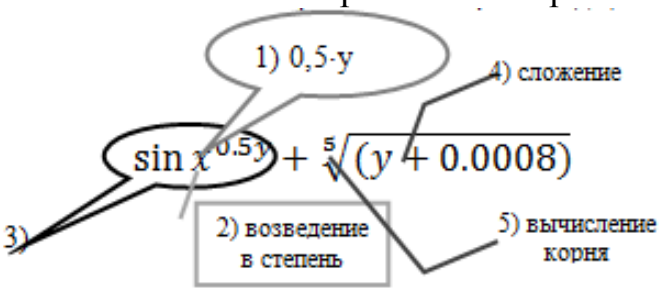
```
z = (x = y) * 5;
```

В этом операторе переменной *x*сначала присваивается значение переменной *y (приоритет скобок)*, далее вычисляется выражение *x**5, и его результат присваивается переменной *z*.

4.3 Примеры выполнения заданий

Пример 1.Вычисление выражения $\sin x^{0.5y} + \sqrt[5]{(y + 0.0008)}$

Решение. Разложим выражение на простые двухместные операции:



Запишем операции последовательно и произведем замену промежуточных переменных путем вложения исходных данных.

Res1=0.5*y

Res2=pow(x, Res1) → Res2=pow(x, 0.5*y)

Res3=sin(Res2) → Res3=sin(pow(x, 0.5*y))

Res4=y+8e-4 →

Res5= pow (Res4, 1/5.f) → Res5= pow (y+8e-4, 1/5.f)

Result= Res3+ Res5→ Result= sin(pow(x, 0.5*y))+pow (y+8e-4, 1/5.f)

В программе можно записать как

```
printf("ответ: %.3f", sin(pow(x, 0.5*y))+pow (y+8e-4, 1/5.f));
```

Пример 2. Требуется вычислить расстояние между двумя заданными точками $M_1(x_1,y_1)$ и $M_2(x_2,y_2)$.

Решение. Расстояние на плоскости между двумя точками $M_1(x_1,y_1)$ и $M_2(x_2,y_2)$ рассчитывается по формуле:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Составим блок-схему алгоритма, а затем уточним выполнение расчета по формуле (рис. 4.3):

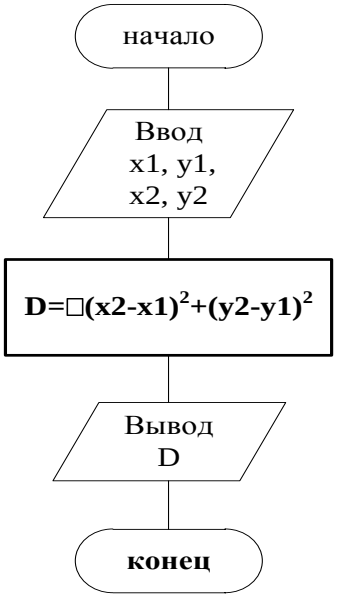


Рис. 4.3. Блок-схема алгоритма

Исходные данные – значение координат целые числа x1, y1 и x2, y2, но не подсчет, поэтому следует выбрать типа float.

Вычисление по формуле состоит из 6 действий: 1) разность, 2) возведение в квадрат, 3) разность; 4) возведение в квадрат; 5) сумма; 6) корень квадратный. Поэтому кроме операций сложения и вычитания потребуется две функции – возведении в степень – pow(x,y) и вычисления корня – sqrt(x), которые имеются в стандартной библиотеке языка Си.

Переводим блок-схему на язык Си.

Шаг 1. Включение заголовков math.h, так как используется математическая функция sqrt(x).

#include<math.h>

Далее реализуется одна единственная функция программы

```
main(){
```

Шаг 2. Объявление переменных для ввода исходных данных и вычисления результата.

float x1, y1, x2, y2, D;

Шаг 3. Организация ввода данных с использованием функции scanf(..).

```
scanf("%f %f %f %f",&x1,&y1, &x2, &y2);
```

Шаг 4. Кодирование формулы с использованием оператора присвоения, знаков арифметических операций и математической функции – вычисления квадратного корня sqrt(x) и возведения в степень pow().

```
D=sqrt(pow(x2-x1,2)-pow( y2-y1,2));
```

Шаг 5. Вывод результата с использованием функции printf(..).

```
printf("Расстояние между точками (%lf, %lf) и (%lf %lf) равно %lf ", x1, y1, x2, y2, D);
```

Шаг 6. Завершение программы.

```
return 0;
}
```

4.4 Практические задания

1. Инициализируйте целую переменную **a** значением 11, и **int b=3**. Объявите переменную **x** типа **int**, **y – float**, а **z** типа **double**. Присвойте им значения равные **a/b** и выведите на консоль полученные значения **x, y** и **z.** Поясните полученные результаты.

2. Напишите программу вывода на экран ряда целых чисел, соответствующих 2^k, где k принимает значение 8, 16, 32, 64, 128. Для вычисления используйте функцию pow(2.f, k). Добейтесь правильного результата работы программы путем корректного выбора типа данных и использования операции преобразования типа.

3. Объявите переменную типа char и реализуйте чтение в нее заданного пользователем символа. Выведите полученный символ a следующим образом:

```
printf("%c %d %x", a, a, a);
```

Прокомментируйте результат.

4. Напишите программу вычисления тригонометрической функции угла,заданного в градусах. Учтите, что некоторые из этих функций принимают аргумент в радианах, поэтому используйте директиву:

```
#define PI 3.14159
```

Обязательно в программе реализуйте приветствие, приглашение к вводу данных и указание на результат.

В комментариях программы, укажите три тестовых значения углов и ожидаемых результатов.

Вариант	***функция***	***Вариант***	***функция***
1	синус	***7***	1-sin(x)
2	косинус	***8***	1-cos(x)
3	тангенс	***9***	1-tg(x)
4	котангенс	***10***	$\sin^2(x)$
5	секанс	***11***	$\cos^2(x)$
6	косеканс	***12***	$tg^2(x)$

4.5 Контрольные задания

Создайте программу вычисления указанной величины. Результат проверьте на заданных исходных данных.

1. $t = \dfrac{2\cos(x-\pi/6)}{0{,}5+\sin^2 y}\left(1+\dfrac{z^2}{3-z^2/5}\right).$	При x = 14.26, y = –1.22, $z = 3.5\times10^{-2}$: **0.564846.**
2. $u = \dfrac{\sqrt[3]{8+\lvert x-y\rvert^2+1}}{x^2+y^2+2} - e^{\lvert x-y\rvert}(tg^2 z+1)^x$.	При x = –4.5, $y = 0.75\times10^{-4}$, $z = 0.845\times10^2$:**–55.6848.**
3. $v = \dfrac{1+\sin^2(x+y)}{\left\lvert x-\dfrac{2y}{1+x^2y^2}\right\rvert} x^{\lvert y\rvert} + \cos^2[arctg(1/z)]$.	При $x = 3.74\times10^{-2}$, y = –0.825, $z = 0.16\times10^2$: **1.0553.**
4. $w = \lvert\cos x-\cos y\rvert^{(1+2\sin^2 y)}\left(1+z+\dfrac{z^2}{2}+\dfrac{z^3}{3}+\dfrac{z^4}{4}\right).$	При $x = 0.4\times10^4$, y = –0.875, $z = -0.475\times10^{-3}$: **1.9873**.
5. $\alpha = \ln(y^{-\sqrt{\lvert x\rvert}})(x-y/2)+\sin^2 arctg(z).$	При x = –15.246, $y = 4.642\times10^{-2}$, $z = 20.001\times10^2$: **–182.036**.

Реализуйте ввод исходных данных с консоли для проведения вычислений по заданной формуле.

Следует обратить внимание на тип входных параметров и возвращаемых результатов используемых математических функций!

ЛАБОРАТОРНАЯ РАБОТА № 5
РЕАЛИЗАЦИЯ РАЗВЕТВЛЯЮЩИХСЯ АЛГОРИТМОВ

Цель работы: Получение практических навыков ветвлений в программах с использованием операторов if-else языка Си и закрепление навыков использования арифметических и логических операций.

Программные средства: MICROSOFT VISUAL STUDIO

5.1. Теоретические сведения

Линейные алгоритмы используются редко, наиболее востребованными являются алгоритмы ветвления и циклические, блок-схемы алгоритмов на рис.

5.1.

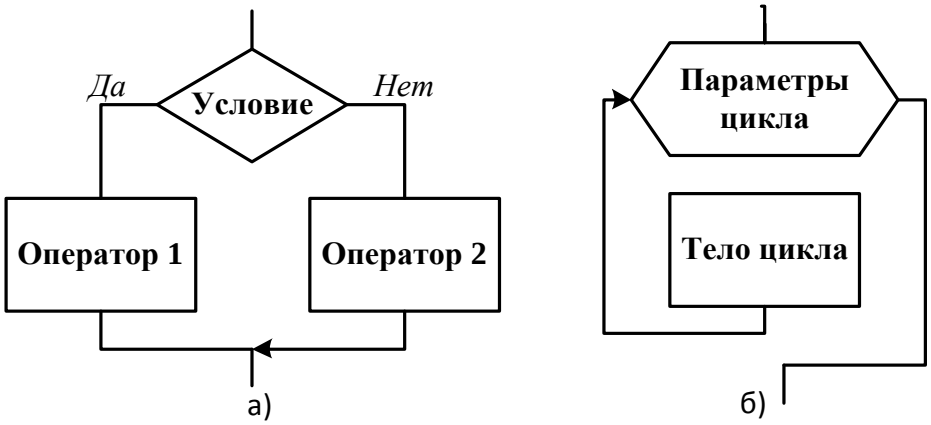


Рис. 5.1. Разветвлённый и циклический алгоритм

Для выбора одной из ветвей вычислений применяется оператор условного перехода:

```
if (выражение) оператор 1;
    else оператор 2;
```

т. е. вычисляется *выражение*, и если его значение не равно 0 (истинно), то выполняется оператор 1, иначе – оператор 2.

Если операторы 1, 2 содержат более одного оператора, то они заключаются в фигурные скобки { }, т.е. применяется блок.

Конструкция **else…** может отсутствовать, и такую форму называют сокращенной, тогда в случае ложности условия, управление передается на следующий за **if** оператор.

Если операторы 1 или 2 в свою очередь являются операторами **if**, то такой оператор называют вложенным, при этом ключевое слово **else** принадлежит ближайшему предшествующему **if**.

Для построения выражений могут использоваться операции отношения и логические операции (таблица 5.1) Операции отношения – бинарные, их общий вид:

```
Операнд 1   знак операции   Операнд 2
```

Операндами операций сравнения могут быть данные любых базовых типов, значения которых перед сравнением преобразуются к одному типу.

Таблица 5.1

Операции отношения и логические операции

Операции отношения		***Логические операции***	
знак операции	***содержание***	***знак операции***	***содержание***
= =	равно	!	отрицание или логическое НЕ
<	меньше		
<=	меньше или равно	&&	конъюнкция или логическое И
>	больше		
>=	больше или равно	||	дизъюнкция или логическое ИЛИ
!=	не равно		

Логические операции используются в качестве условий при составлении более сложных выражений.

Выражения вычисляются слева направо, причем их вычисление прекращается, как только результат становится известен.

Простой выбор варианта вычисления из двух альтернатив может быть организован с использованием тернарной (операцией с тремя операндами)

Тернарная (условная) операция ? имеет форму:

Операнд 1 ?Операнд 2 :Операнд 3

Если значение **операнда 1** истинно (не равно 0), то результатом операции является **операнд 2**, иначе – **операнд 3**.

Пример. Найти наибольшее из двух чисел:

```
max = a>b ? a : b;.
```

5.2 Приемы оформления ветвлений

Для того чтобы код программы был читабелен, следует использовать специальные приемы форматирования.

Чтобы не запутаться в блоках if, в условных операторах используют абзацные отступы. Прием, известный как лестница if-else-if, позволяет упорядочить вложенные блоки if. Смысл в том, чтобы каждый вложенный блок сдвигать несколько в сторону (используется табуляция \t или обычно 4 пробела).

```
if (условие) оператор;
else
if (условие) оператор;
else
    if (условие) оператор;
        .
            .
                .
                else оператор;
```

В лестнице ***условия*** операторов if вычисляются сверху вниз. Если встретилось условие с ненулевым значением (т.е. условие истинно), выполняется оператор этого блока if, а оставшаяся часть лестницы пропускается. Если все условия ложны, то выполняется оператор в последнем блоке else, или не выполняется ни один оператор, если лестница не заканчивается else.

5.3 Примеры решения задач

Пример 1. Составление программы вычисления значения функции

$$Y(X) = \begin{cases} -1/X & при \quad X < -3, \\ \sqrt{-X} & при \quad -3 \le X < 0, \\ X^2 & при \quad 0 \le X < 1, \\ \sqrt{X} & при \quad 1 \le X. \end{cases}$$

При составлении программ нельзя прямо использовать двойные неравенства. Так запись типа -3<=x<0 даст неверный ответ. Пусть x=-1, тогда значение выражения(-3<=-1)будет «правильно» , т. е. 1, значит далее вычисляется значение выражения(1<0), которое дает результат – 0. Хотя с точки зрения математической записи этого двойного неравенства должен получиться результат – правильно, т. е. 1.

Поэтому при программировании следует использовать простые операции отношения соединенные логическими функциями, например: (x>=-3)&&(x<0).

```
void main()
{
double x;
    if(x<-3)printf("Result=%.3f",-1/x);
if((x>=-3)&&(x<0))printf("Result=%.3f",sqrt(-1*x));
if((x>=0)&&(x<1))printf("Result=%.3f",x*x);
if((x>=1) printf("Result=%.3f",sqrt(x));
}
```

Пример 2. Составление программы поиска наибольшего значения

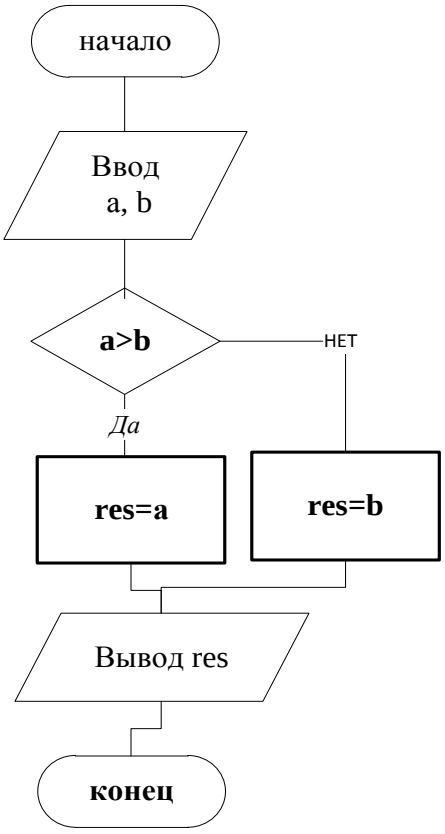


Рис.5.2. Поиск наибольшего значения

Шаг 1. Объявление переменных.

```
Double a, b, res;
```

Шаг 2. Ввод значений

```
scanf ("%lf, %lf", &a, &b);
```

Шаг 3. Реализация ветвления с помощью конструкции if – else

```
if (a>b) res=a
else res=b
```

Шаг 4. Вывод результата

```
printf ("Наибольшее %lf", res);
```

Шаг 3 и 4 можно совместить, выводя результат сразу после сравнения:

```
if (a>b) printf ("Наибольшее %lf", a);
else printf ("Наибольшее %lf", b);
```

Также можно использовать тернарную операцию:

```
a>b? printf ("Наибольшее %lf", a): printf ("Наибольшее %lf", b);
```

Пример 3. Программа разделения трехзначного числа на единицы, десятки и сотни.

```
void main()
{
int num;
puts("введите трехзначное число");
scanf ("%d", &a);
if ((a<0) && (a>999))
{
puts("число введено не верно");return;
}
printf("сотни %d, десятки %d, единицы %d", num/100,
(num/10)%10, num%10);
}
```

5.4 Практические задания

1. Напишите программу, которая проверяет, является ли введенный год високосным (т.е. кратен 4). Реализуйте ее по шагам:
   a. Объявление целой переменной, years;
   b. Приглашение к вводу года;
   c. Ввод с клавиатуры значений;
   d. Вычисление остатка от деления введенного числа на 4;
   e. Применение структуры **if–else** для вывода на экран одного из двух сообщений – «високосный год» и «год не високосный» с использованием функции **printf**.
   f. Вывод на экран – результата в виде: «Год (число) не или високосный».

2. Реализуйте дополнительные проверки високосного года, по следующему правилу: «год является високосным в двух случаях: либо он кратен 4, но при этом не кратен 100, либо кратен 400».

Проверьте правильность работы программы на следующих контрольных значения: *1900 – не високосный, 2000 – високосный, 2100 – не високосный*.

3. Организуйте в программе вывод результата с использованием тернарной операции в функции **printf**:

```
printf("%dгод - %s", years, (yeas%4==0? "високосный": "не
високосный"));
```

4. Напишите программу для вычислений значения функции F(x, y), где x и y – вещественные числа.

Вариант	F(x, y)	Вариант	F(x, y)
1	$\begin{cases} x-y & x>y \\ y-x+1 & иначе \end{cases}$	7	$\begin{cases} x/y & x \geq y \\ 2*y & иначе \end{cases}$
2	min{x,y}+1	8	max {x+y, x*y};

Вариант	F(x, y)	Вариант	F(x, y)
3	$\begin{cases} \sin(x) & x \geq y \\ 1+\cos^2(y) & иначе \end{cases}$	9	$\begin{cases} x^y & x < y \\ y^x & иначе \end{cases}$
4	2*max{x,y}	10	min {x/y, | x-y |};
5	$\begin{cases} \overline{2x+y^2, \quad x > y} \\ 100 \quad x = y \\ \sqrt[y]{x} \quad x < y \end{cases}$	11	$\begin{cases} \overline{x^y, \quad x > y} \\ -1 \quad x = y \\ y^x \quad x < y \end{cases}$
6	$\min\{x+\frac{y}{2}, \frac{x+y}{2}\}$	12	$\max\{x-\frac{y}{2}, \frac{x-y}{2}\}$

5.5 Контрольные задания

1. Напишите программу, которая выводит пример на умножение двух однозначных чисел, запрашивает ответ пользователя, проверяет его и выводит сообщение "Правильно!" или "Вы ошиблись" и правильный результат.

2. Напишите программу, которая по трем введенным с клавиатуры значениям сторон треугольника определяет, является ли он прямоугольным.

3. Напишите программу, которая по введенной пользователем координате точки A(x,y) определяет, к какой четверти она относиться.

4. Напишите программу, которая сравнивает два числа и выводит ответ в текстовой форме (например, «5 меньше 3»).

5. Напишите программу, которая запрашивает у пользователя номер дня недели и выводит одно из сообщений: "Рабочий день" или "Выходной". В случае, если пользователь введет недопустимое число, программа должна вывести сообщение "Ошибка ввода данных".

6. Напишите программу, которая по запросу пользователя округляет введенное пользователем вещественное число в большую или меньшую сторону.

7. Напишите программу, определяющую состояние воды (твердое, жидкое и газообразное) по известной температуре.

При написании программ необходимо сопровождать ввод данных поясняющим текстом, выводить название программы и оформлять вывод результатов с использованием элементов форматирования функции printf ().

ЛАБОРАТОРНАЯ РАБОТА №6
МНОГОАЛЬТЕРНАТИВНАЯ ОБРАБОТКА ДАННЫХ

Цель работы: Формирование навыков использования операторов выбора, прерывания и продолжения последовательности действий для управления ходом программы

6.1 Теоретическая справка

Помимо конструкций if-else, позволяющих осуществлять ветвление, в языке Си предусмотрена возможность многоальтернативного выбора последовательности действий с помощью оператора switch. Он позволяет передать управление в зависимости от заданного значения выражения. В общем случае использование конструкции switch имеет вид:

```
switch (выражение)
{
case константа_1: операторы
case константа_2: операторы
...
default: операторы
}
```

Сначала вычисляется значение выражения. Затем это значение сравнивается с константами. Управление передается на метку, для которой это сравнение дает значение «истина». Если значение выражения не совпадает ни с одной из констант, то управление передается инструкции с меткой **default**. А если этой метки нет, то происходит выход из блока switch. Для принудительного выхода из блока **switch** используется инструкция **break.**

При использовании оператора switch необходимо помнить, что выражение и константы в инструкции switch должны иметь целочисленный тип и никакие две константы не могут иметь одинаковое значение.

Часто при программировании управляющих инструкций необходимо, чтобы при выполнении некоторого условия выполнялись сразу несколько операторов. Для этой цели инструкции объединяются в блок. Начало и конец блока отмечаются соответственно – { и }. Фигурные скобки следует размещать непосредственно под ключевым словом и по горизонтали на одном уровне с ним. Важно запомнить, что после окончания блока ставить точку с запятой не нужно.

Для принудительного выхода из блока используется оператор break, который осуществляет переход к следующему после ближайшей фигурной скобки оператору.

При множественном выборе (оператор switch) следует указывать поведение по умолчанию (default), что облегчает поиск ошибок при отладке программы (рис.6.1).

Оператор switch предпочтительнее if в тех случаях, когда требуется больше двух ветвей программы и выражение, по значению которого производится переход, является целочисленным.

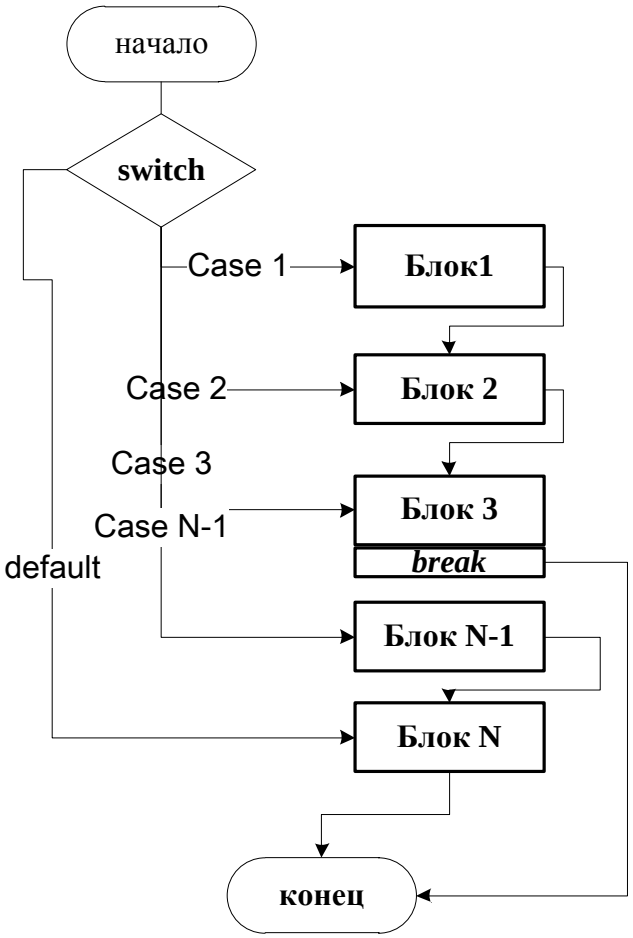


Рис. 6.1. Управление ветвлением с операторами switch и break

6.2 Примеры использования управляющих конструкций

Пример 1. Использование оператора switch

```
#include<stdio.h>
int main()
{
char c;
printf("Введите символы 'a' or 'b': ");
scanf("%c", &c);//считывание с консоли
switch (c)
{
case 'a':
    printf("Введено 'a'.\n");
    break;
case 'b':
    printf("Введено 'b'.\n");
    break;
default:
    printf("Неизвестный символ\n");
}
```

```
return 0;
}
```

Пример 2. Пример использования вложенных if-else для организации меню в консольной программе

```
#include <math.h>
#include <stdio.h>
void main ( void )
{
int choice;     /* Текущийвыборпунктаменю*/
double fun, x;  /* Значения функции и аргумента */
printf( "\nВведите аргумент x=" );
scanf( "%lf", &x );
printf ( "\n   Введите номер функции:\n" );
printf ( "1. sin(x)\n2. cos(x)\n" );
printf ( "3. tan(x)\n4. Конец работы\n" );
scanf  ( "%d", &choice );
if      (choice==1) fun=sin(x);
    else if (choice==2) fun=cos(x);
        else if (choice==3) fun=tan(x);
            else if (choice==4) { loop=0; break; }
                    else {printf("Неверный выбор\n"
                    ); continue; }
  printf( "Значение функции %lf\n", fun );
 }
```

Пример 3. Программа, определяющая какая из курсорных клавиш была нажата

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include <locale.h>
void main()
{
intkey, scan=0;
setlocale(LC_ALL, "RUS"); // для переключения русской
кодировки
printf("Нажмите одну из курсорных клавиш\n ");
key=_getch(); //scan=getch();
if (key>=0xe0)  scan=_getch();
switch (scan)
{
case 77:
printf("стрелка вправо\n");
break;
case 75:
```

```
printf("стрелка влево\n");
break;
case 72:
printf("стрелка вверх\n");
break;
case 80:
printf("стрелка вниз\n");
break;

default:
printf("Не стрелка");
}

system("pause");
}
```

В программе использована функция getch(), которая возвращает код нажатой пользователем клавиши и не отображает символ на экране. В случае нажатия функциональных или курсорных клавиш, эта функция возвращает 0 или 0xE0 в зависимости от компилятора, и ее повторный вызов позволяет получить расширенный код клавиши.

6.3 Практические задания

1. Напишите программу «Калькулятор», для этого:

А) Оформите приглашение к вводу данных, так чтобы вводимая пользователем информация могла быть считана следующей функцией:

```
scanf("%f%c%f", &x, &c, &y);
```

где x и y типа float, а c – char.

Б) С использованием оператора switch интерпретируйте введенный символ для определения вида операции.

В) Организуйте вычисление и вывод результата непосредственно в функции printf:

```
printf("=%f", a+b);
```

2. Оформите внешний вид программы и добавьте бесконечный циклwhile() для повторения вычислений, пока пользователь не введет в качестве первого операнда 0.

```
while(1)
{
scanf("%f%c%f", &x, &c, &y);
if(x==0.) break;
// основной текст программы
}
```

3. Для разработанной программы придумайте дополнительный

функционал и реализуйте дополнительный функционал – возведение в степень, логическое сложение и умножение, или возможность ввода операции умножения различными символами (‘*’или ‘x’), или различные проверки корректности ввода данных.

6.4 Контрольные задания

1. Написать программу, которая в зависимости от порядкового номера дня недели выводит на экран его название. Предусмотреть возможность нумерации, как с понедельника, так и с воскресенья.

2. Составить программу, которая в зависимости от порядкового номера месяца выводит количество дней в этом месяце. Предусмотреть возможность выбора года (високосный или невисокосный).

3. Написать программу, которая проверяет, является ли последняя цифра заданного двухзначного числа – простым числом. Конструкцию if-else в программе использовать нельзя.

4. Составить программу, которая в зависимости от порядкового номера месяца выводит на экран время года. Использовать конструкцию if-else недопустимо.

5. Составить программу, которая для заданного числа от 1 до 12 выводит его отображение римскими числами.

6. Для натурального числа k (k<20) напечатать фразу «в программе найдено k ошибок», согласовав окончание слова «ошибка» с числом k. Использовать конструкцию if-else недопустимо.

7. Написать программу, которая интерпретирует степень числа 2 в соответствующие приставки (10 – K, 20 – M, G, T, P, E).

ЛАБОРАТОРНАЯ РАБОТА №7
ЦИКЛИЧЕСКИЙ ВЫЧИСЛИТЕЛЬНЫЙ ПРОЦЕСС

Цель работы: Получение навыков использования операторов циклов для организации повторений вычислительного процесса

Программные средства: MICROSOFT VISUAL STUDIO

7.1. Циклический вычислительный процесс

Циклический вычислительный процесс характеризуется повторением одних и тех же вычислений над некоторым набором данных. Различают циклы с переменным и фиксированным числом повторений. В первом случае число повторений определяется динамически, в зависимости от условия. Во втором – числом повторений цикла управляет специальная переменная, называемая счетчиком. На счетчик накладывается условие, определяющее, до каких пор следует выполнять цикл.

Повторяемый блок вычислений называют телом цикла (рис. 7.1). В теле цикла должно быть обеспечено изменение значения счетчика либо переменной, участвующей в условии, чтобы он мог завершиться.

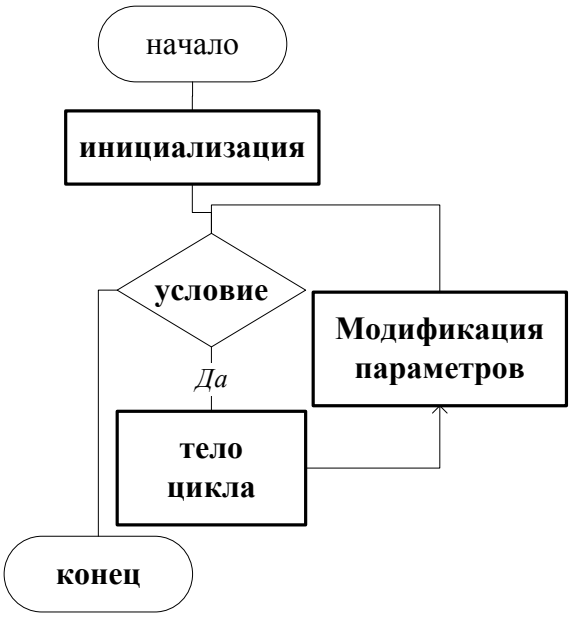


Рис.7.1. Цикл

Возможны всего два вида циклов:

– с предусловием, когда проверка условия предшествует выполнению тела цикла;

– с постусловием, когда проверка условия происходит после выполнения тела цикла.

В языке Си имеются три управляющие конструкции для организации цикла:

while – циклы с предусловием;

for – для фиксированного числа повторений;

do-while– циклы с постусловием.

Цикл **while** имеет следующий формат (синтаксис) записи:

while (условие)

оператор цикла;

Если в результате проверки условия получается истинный результат, то выполняется **оператор**, следующий непосредственно за закрывающей круглой скобкой. Затем снова оценивается условие, и если результатом расчета окажется **TRUE**, то вновь будут выполнены операторы цикла. Цикл повторяется до тех пор, пока выполняется условие, которое является признаком окончания цикла, после чего выполнение программы продолжается со следующего за циклом оператора.

Оператор цикла **for** имеет следующий формат записи:

for (инициализация; условие; модификация параметров)

оператор цикла;

Первый параметр **инициализация** используется для задания начального значения цикла.

Второй компонент определяет **условие или условия**, в соответствии с которыми будет осуществляться выход из цикла. Повторение будет происходить до тех пор, пока это условие (или условия) выполняется. Если условие не выполняется, то цикл немедленно заканчивается.

Третий параметр выполняется каждый раз, когда заканчивается обработка тела цикла и, как правило, необходим для **модификации** некоторых

переменных, позволяющих достигнуть условия выхода из цикла.

Любую из трех частей можно опустить, но точки с запятыми должны остаться на своих местах. Если же опустить проверку условия, то по умолчанию считается, что условие продолжения цикла всегда истинно, и тогда цикл станет бесконечным (произойдет зацикливание).

Конструкция цикла, реализованная оператором **for**, может быть выполнена также оператором **while**:

```
инициализация;
while (условие)
{
оператор цикла;
модификация параметров;
}
```

В случае необходимости производить проверку условия выполнения цикла после исполнения тела цикла (т. е. тело цикла гарантировано выполнится хотя бы один раз), прибегают к циклу **do – while**.

Оператор цикла **do – while** имеет следующий формат записи:

```
do
оператор цикла;
while (условие);
```

Повторение цикла будет продолжаться до тех пор, пока условие истинно, в противном случае выполнение цикла прекратится и произойдет переход к оператору программы, непосредственно следующему за циклом.

Когда требуется выполнение нескольких последовательных действий, тело цикла строится из последовательности операторов, заключенных в фигурные скобки.

7.2. Приемы эффективного построения циклов

Язык Си обеспечивает широкие возможности по организации циклов, поэтому при написании программ трудно определить, какой оператор следует выбрать. Все, что можно сделать с помощь while можно сделать и с for. Применение цикла for предпочтительно, когда явно используется инициализация и модификация переменной, а while удобнее, когда этого не требуется.

Если обязательно следует выполнить цикл хотя бы раз – лучше использовать do-while.

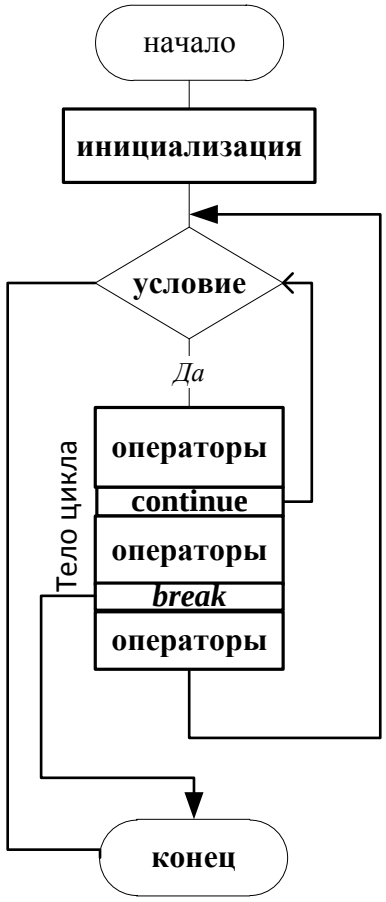


Рис. 7.2. Управление циклом с операторами break и continue

Оператор **break** следует использовать, если необходимо выйти из цикла в произвольной точке тела цикла.

Оператор **continue** – если следует продолжить повторение, но в некоторой итерации требуется пропустить часть операторов цикла.

В некоторых случаях для управления выходом из цикла очень удобны флаги. Флагом принято называть переменную, принимающую значение 1, если выполнено некоторое условие или произошло определенное событие.

7.3. Примеры построения циклов

Пример 1. Программа вычисления 100 чисел Фибоначчи

Числа Фибоначчи — элементы числовой последовательности, в которой каждое последующее число равно сумме двух предыдущих чисел:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, …

Такую программу легко реализовать в цикле. Две переменные используются для хранения двух предшествующих элементов, третья – в качестве счетчика элементов последовательности. В теле цикла каждый следующий элемент вычисляется как сумму предшествующих и осуществляется замена предшествующего значения вычисленным.

```
int main (void)
{
int f,g,h;
g=0; h=1;//инициализированы первые два числа
printf("%d \t%d \t ",g,h);//вывод первых элементов

for (int i=2;i<100;i++)
```

```
{
f=g+h;//вычисление суммы предшествующих чисел
printf("%d \t",f);//вывод числа
h=g;//сдвиг последовательности
g=f;//текущее значение становится предшествующими
}
return 0;
}
```

Пример 2. Программа-счетчик повторений

```
#include<stdio.h>   //Для printf и puts
#include <process.h>  //Для system

int main (void)
{
int MyTik=0; //Счетчик остановок работы
puts ("Остановка работы на 5 тактов");
//Цикл на пять тактов
while (MyTik<5)
   {
//Вывод оставшегося времени до возобновления работы
printf ("Осталось %d такт.\n",5-MyTik);
//Приостановка работы до нажатия клавиши
system("pause");
//Увеличение на 1 счетчика приостановок
MyTik++;
   }
//Вывод сообщения о возобновлении работы
puts ("Работа возобновлена");
return 0;
}
```

Эта же задача с использованием цикла for:

```
intmain (void)
{
puts ("Остановка работы на 5 тактов");
for(int MyTik=0; MyTik<5; MyTik++;)
{
printf ("Осталось %d такт.\n",5-MyTik);
system("pause");
   }
puts ("Работа возобновлена");
return 0;
}
```

Пример 3. Программа, иллюстрирующая универсальность цикла for.

```
#include <stdio.h>
main(void) {
int num=0;
for(printf("Запоминайте введенные числа!\n"); num != 
5; scanf(" %d", &num));//!пустой оператор в теле цикла
printf("Это как раз то, что я хочу!\n");
return 1;
}
```

Пример 4. Программа, определяющая, есть ли во введённом пятнадцатизначном числе цифра 5.

```
main()
{
    long long int i;
    puts("Введите число\n");
    scanf("%lld", &i);
    while (i!=0)
    {
        if (i % 10 == 5)
        {
            puts("Да, цифра 5 имеется в числе\n");
            system("pause");
            return;
        }
        i = (int)(i / 10);
    }
    printf("Нет, цифры 5 в числе нет\n");
    system("pause");
}
```

7.4. Практические задания

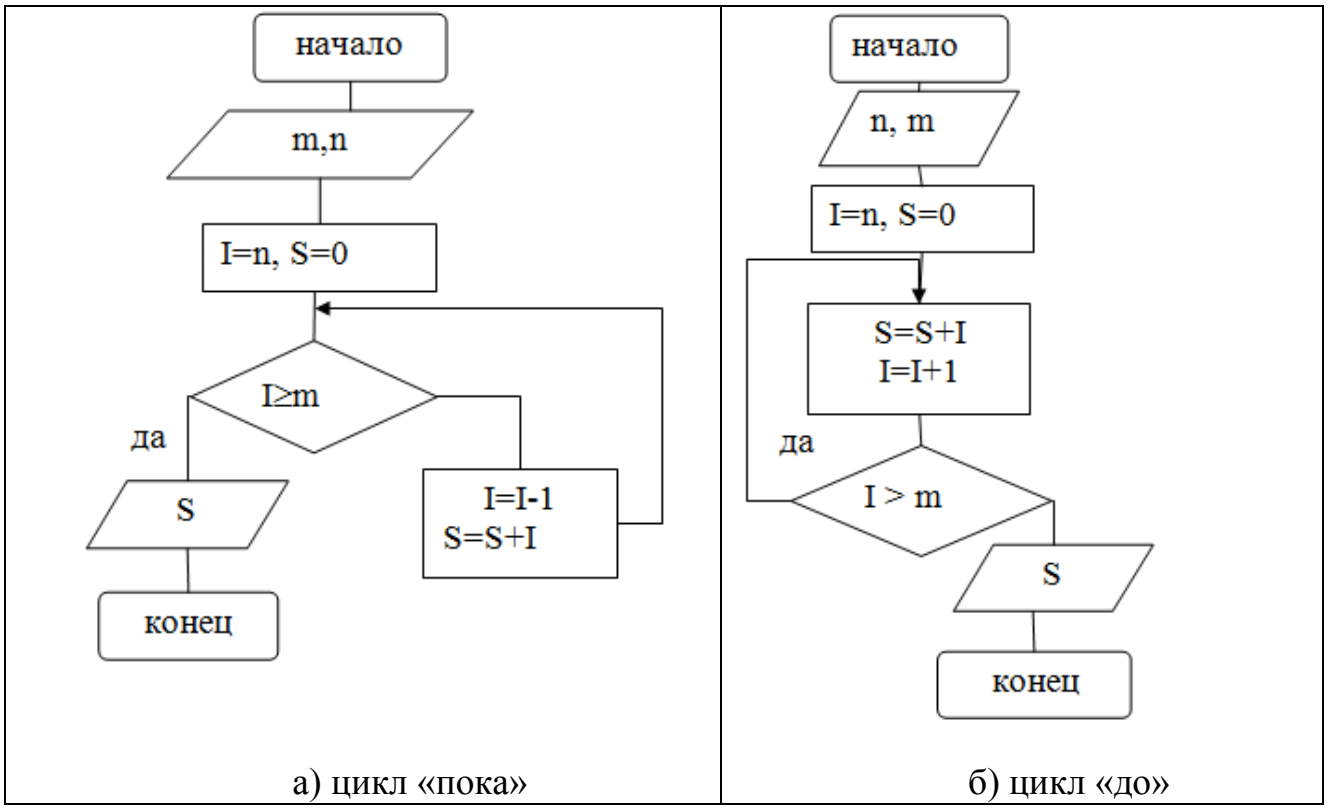
1. Разработайте программу вычисления суммы целых чисел от m до n (включительно)

Комментарий.

Для составления программы вычисления необходимо реализовать цикл.

В качестве счетчика цикла выбрана переменная I, изменяющая значение от m до n. Переменная S используется для накопления результата. В теле цикла использована одна операция сложения результирующей переменной S с текущим значением переменой I. Данная схема может быть реализована как циклом «пока» (рис. 7.3.а), так и «до» (рис. 7.3.б). Цикл «пока» реализуется увеличением значений переменной I от m до n. Цикл «до» реализуется уменьшением значений переменной I от n до m. Использование цикла «до»

предпочтительнее ввиду уменьшения количества переходов. Выход из цикла будет осуществлен на основе проверки условия, сведенного к операции арифметического сравнения.



а) цикл «пока» б) цикл «до»

Рис. 7.3. Блок-схема алгоритма вычисления суммы последовательных чисел

Реализуем цикл «пока» с использованием оператора for.

Шаг 1. Все переменные необходимо объявить.

```
int n, m;
```

Шаг 2. Необходимо организовать ввод значений n и m с клавиатуры.

```
scanf("%d", &n);
```

Шаг 3. Инициализация переменных может быть включена в конструкцию цикла

```
for (int i=n, s=0; i>=m; s+=i, i--);
```

Шаг 4. По завершению цикла необходимо вывести результат.

```
printf("результат %d", s);
```

Обратите внимание, объявлять переменную s внутри цикла нельзя.

2. Измените порядок следования выражений третьего параметра цикла for.

3. Организуйте тело цикла вне скобок.

4. Реализуйте эту задачу с использованием конструкции while (по блок-схеме а).

5. Реализуйте цикл «пока», через конструкцию do-while.

7.5 Контрольные задания

Написать программу табуляции заданной функции в определенном интервале и с произвольным шагом. Обязательно использовать конструкцию for. Вначале программы вывести табулируемую функцию. Результаты оформить в виде таблицы. Организовать возможность ввода значения шага табуляции и задания произвольного интервала по шаблону "%c%f %f%c", z1,a,b,z2, где z1,z2 могут быть [ ] или (), a и b - числовые значения границ интервала.

Варианты функций:

1) $y = x^2 + \sin 5x$, x$\in$[0.1; 2.1]

2) $y = 2^x - 2x^2 - 1$, x$\in$[2.0; 4.0]

3) $y = \left|e^x - 2\right| - x^2$, x$\in$[1.0; 3.0]

4) $y = \sqrt{x} - x\pi\cos\pi x$, x$\in$[0.2; 2.5]

5) $y = \ln x^2 - x + 4$, x$\in$[1.2; 28.5]

ЛАБОРАТОРНАЯ РАБОТА №8
ИСПОЛЬЗОВАНИЕ ВЛОЖЕННЫХ ЦИКЛОВ

Цель работы: Закрепление навыков использования операторов управления языка Си

Программные средства: MICROSOFT VISUAL STUDIO

8.1. Практические аспекты использования циклов

В теле цикла разрешены любые исполнимые операторы, в том числе и циклы, т.е. вложенные циклы. Такие конструкции очень распространены при вычислении числовых рядов и построении таблиц.

Если один цикл находится внутри другого цикла, то первый цикл называют внутренним, а второй – внешним. Внутри вложенного цикла, в свою очередь, может быть вложен еще один цикл, образуя следующий уровень вложенности и так далее. Количество уровней вложенности, как правило, не ограничивается (рис.8.1).

Одна из проблем, связанных с вложенными циклами – организация досрочного выхода из них. Оператор break позволяет выйти из цикла до его завершения, а continue начать новую итерацию. Вызов break из вложенного цикла приведёт к завершению только этого внутреннего цикла, внешний же цикл продолжит выполняться. Если нужно завершить всю программу, то можно использовать оператор return или функцию exit(). В противном случае придется использовать штатные средства завершения циклов, устанавливая специальные флаги, требующие немедленного завершения внешнего цикла обработки.

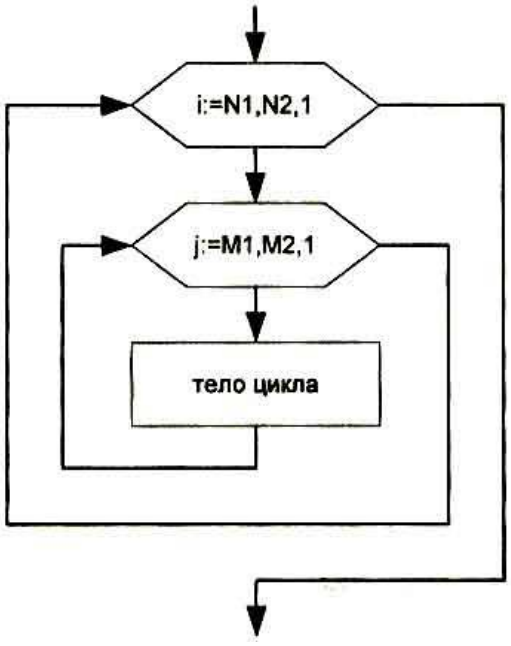


Рисунок 8.1. Блок-схема алгоритма с внутренним циклом

Вложенные циклы параметризацией:

```
for ( i=0; i<n; i++)
{// цикл 1
//операторы цикла 1;
for( j=0; j<n; j++)
{  // цикл 2
    // операторы
    // в теле цикла 2
}
//операторы цикла 1;
}
```

8.2. Методы оптимизации циклов

Неудачное построение цикла приводит к значительному снижению быстродействия программы, поэтому широко используется оптимизация циклов. Самый простой прием оптимизации – вынесение из цикла тех переменных, которые в нем не модифицируются.

Например,

```
sum = 0;
for (i = 1; i<1000; i++)
    sum += a * (x+i);
```

Медленнее

```
sum = 0;
for (i = 1; i<1000; i++)
    sum += (x+i);
sum*=a;
```

Быстрее

Методы оптимизации циклов:

- развертка циклов (loop unrolling),
- объединение циклов (loop fusion),
- разрезание циклов (loop distribution),
- выравнивание циклов (loop alignment),
- перестановка циклов (loop interchange),
- разделение на блоки (loop blocking).

Сопоставление

```
for ( int j = 0; j< 100000; j++)
        (for int k = 0; k< 1000; k++) a=1;
```

Медленнее, так как внутренний

```
for ( int k = 0; j< 1000; k++)
        (for int j = 0; k< 100000; j++) a=1;
```

Быстрее, так как внутренний счетчик

счетчик инициализируются 10001 раз инициализируются 1001 раз

При программировании вложенных циклов по возможности следует делать цикл с наибольшим числом повторений самым внутренним, а цикл с наименьшим числом повторений — самым внешним.

Также следует отметить, что время выполнения цикла со счетчиком и цикла с постусловием при всех прочих равных условиях совпадает, цикл с предусловием выполняется дольше.

8.3. Примеры использования вложенных циклов

Пример 1. Программа составления таблицы умножения

```
void main()
{
int row, col;
puts ("\t Таблица Пифагора\n");
for (row=1;row<=10;row++)
{
    for (col=1;col<=10;col++)
        printf("%5d", row*col);
printf("\n");}}
```

Пример 2. Программа рисования прямоугольного треугольника, заполненного заданным пользователем символом

```
void main()
{
char paint;
puts ("\t Введите символ\n");
scanf("%c",&paint);
for (int i=0;i<5;i++)
{
    for (int k=0;k<=i;k++)
        putchar(paint);
putchar('\n');
}
getchar();
}
```

Пример 3. Модификация программы поиска заданной цифры во введенном числе для повторного выполнения по желанию пользователя

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

int main(void)
{
```

```
    long long int i;
    char a;
    while(1)
        {
        setlocale(0, "Russian");
        puts("введите число");
        scanf("%lld", &i);
        getchar();
            while (i!=0)
            {
                if (i % 10 == 5)
                {
                    printf("Да, цифра 5 имеется в
числе\n");
                    break;
                }
                i = (int)(i / 10);
            }
        if  (!i)  printf("Нет, цифры 5 не имеется в
числе\n");
        printf("Продолжить?(Да - y, нет -n)");
    if ((a=getchar())=='n') break;
    }
    system("pause");
    }
```

8.4. Практические задания

1. Напишите программу вычисления функции sin с заданной точностью eps.

Математическая модель решения – разложение функции синус в ряд Тейлора имеет вид:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + ...(-1)^{(k-1)} \frac{x^{2k-1}}{(2k-1)!}$$

При решении важно понимать, что чем больше членов ряда будет просуммировано, тем точнее будет вычислен синус. Так если приемлемая точность равна eps, то достаточно суммировать члены ряда до тех пор, пока очередной член ряда не окажется меньше eps.

Для решения этой задачи следует использовать метод накопления, когда формирование следующего члена ряда осуществляется на основе предшествующего. В данном случае для получениek-го члена ряда нужно (k-1)-й член умножить на x*x, разделить на k*(k-1). Это позволяет исключить использование функции возведения в степень и явное вычисление факториала в каждой итерации.

Следует также обратить внимание, что ряд – знакопеременный, т.е. в одной итерации значение прибавляется, а в следующей вычитается. Для такого случая применим прием, известный как «мерцающий счетчик», когда выделяется переменная и ее значение в каждой итерации умножается на (-1), таким образом чередуются отрицательные и положительные значения.

А) Оформите приглашение к вводу значения eps и инициализацию значения:

```
doubl eeps=1e-5;
```

Б) Организуйте цикл вычислений (ak – член ряда, sum – текущее значение ряда, k – счетчик итераций)

Инициализация:

```
ak=x; sum=ak; k=1;
```

Проверка условия:

```
fabs(ak)<eps;
```

Модификация:

```
k+=2; ak*=(-1)*x*x/(k*(k-1));
```

Тело цикла:

```
sum+=ak;
```

Для данной задачи можно выбирать любые конструкции построения цикла вычислений, но при выборе do-while будет иная инициализация переменных.

2. Дополните программу бесконечным циклом while() для повторения вычислений, пока пользователь задаст точность 0.

```
if(eps==0.) break;
```

3. Доработайте программу, чтобы она работала в трех режимах вычислений, включая

А) число итераций N задается пользователем;

Б)число итераций определяется из условия $\left|\frac{ak}{sum}\right| < eps$.

4. Сопоставьте полученные результаты с библиотечной функцией sin(x).

Для этого проведите численный эксперимент, вводя различные значения точности eps,и вычислите погрешность, как разницу между значениями ряда и вычисленным с помощью библиотечной функции. Сделайте выводы.

5. Самостоятельно выполните приближенное вычисление ряда по заданным формулам:

№ вар.	Ряд	функция
1	$1 - x^2/ 2! + x^4/4! + \ldots + (-1)^n x^{2n}/(2n)!$	cos(x)
2	$z=((x-1)/(x+1))$ $f(x)= (2/1)z + (2/3) z^3 +...+ (2/2n-1)z^{2n-1}$	ln(x)
3	$1+ x \ln3 + (x \ln3)^2/2! +...+(x \ln 3)^n/n!$	3^x
4	$(x-1)/x+(x-1)^2/2x^2+(x-1)^3/3x^3+\ldots+(x-1)^n/nx^n$	ln(x)

№ вар.	Ряд	функция
5	$x + x^3/3! + x^5/5! + \ldots + x^{2n-1}/(2n-1)!$	sh(x)
6	$1 + x^2/2! + x^4/4! + \ldots + x^{2n}/(2n)!$	ch(x)
7	$x - x^3/3 + x^5/5 + \ldots + (-1)^{n+1}x^{2n-1}/(2n-1)$	arctg(x)
8	$\pi/2 - 1/x + 1/3x^3 - 1/5x^5 + (-1)^n(2n-1)x^{2n-1}$	arctg(x)
9	$x + x^3*1/(2*3) + x^5*(1*3)/(2*4*5) + \ldots$ $+x^{2n+1}(1*3*\ldots*2n-1)/(2*4*\ldots 2n(2n+1))$	arcsin(x)
10	$1+x(1/2)+x^2(1*1)/(2*4)-x^3(1*1*3)/(2*4*6)+\ldots$ $(-1)^n x^n(1*1*3*\ldots*2n-3)/(2*4*6*\ldots*2n)$	$(1+x)^{0.5}$
11	$1-x(1/2)+x^2(1*3)/(2*4)-x^3(1*3*5)/(2*4*6)+\ldots$ $(-1)^n x^n(1*3*\ldots*2n-1)/(2*4*\ldots*2n)$	$1/(1+x)^{0.5}$
12	$1+x/1!+x^2/2!+x^3/3! \ldots x^n/n!$	e^x

8.5. Контрольные задания

1. Написать программу, которая выводит количество простых чисел в заданном интервале значений. Простыми называются числа, которые делятся только на 1 и сами на себя.

2. Написать программу, которая находит все трехзначные числа, сумма цифр которых равна N (N<27). Операции деления, целочисленного деления и определения остатка не использовать.

3. Написать программу, которая выводит все числа с четными десятками в заданном интервале.

4. Написать программу, которая выводит в порядке возрастания все «счастливые» номера (первая и последняя цифра трехзначного числа совпадают).

5. Написать программу генерирования двоичной последовательности (каждое следующее равно удвоенному предыдущему) заданной длины

6. Написать программу нахождения числа, сумма цифр которого равна заданному числу.

7. Написать программу вывода трехзначных чисел, у которых все цифры различны.

8. Написать программу вычисления суммы первых m цифр пятнадцатиразрядного числа n.

ЛАБОРАТОРНАЯ РАБОТА №9
СТРУКТУРИРОВАНИЕ ПРОГРАММЫ С ИСПОЛЬЗОВАНИЕМ ФУНКЦИЙ

Цель работы: Закрепление понятия области видимости и получение навыков написания собственных функций.

Программные средства: MICROSOFT VISUAL STUDIO.

9.1. Теоретические сведения

Для структурирования программы следует использовать подпрограммы, которые в языке Си называют функции (рис.9.1).

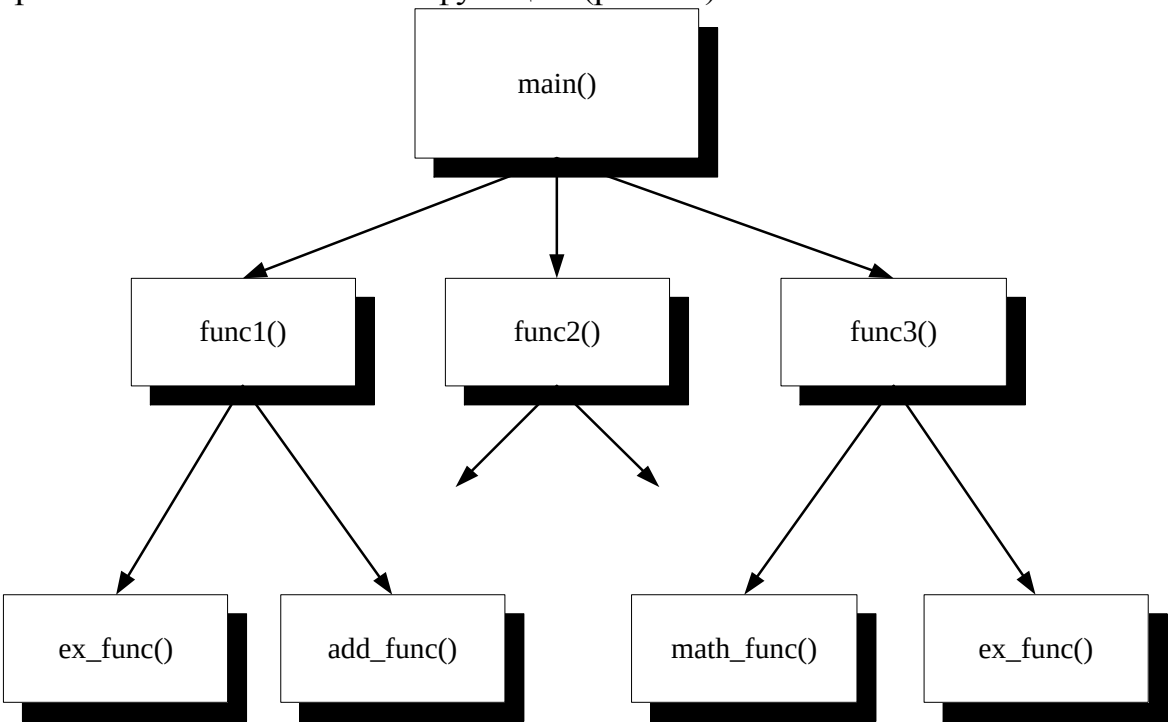


Рис. 9.1. Иерархия функций в программе на Си

Под функцией понимают часть программы, которая реализует вполне законченный алгоритм и может быть вызвана (запущена) из любого места программы. После окончания работы функции программа продолжается с места ее вызова.

Функция обладает именем, которое идентифицирует ее и служит для вызова на выполнение в программе. После имени функции всегда располагаются круглые скобки, в которых описываются или передаются параметры.

С использованием функций в языке Си связаны три понятия: определение функции; объявление (прототип) и вызов функции.

Определение функции задает ее имя, типы и число ее формальных параметров и операторы, которые определяют действие функции.

Описание функции начинается заголовком:

<тип возвращаемого значения> имя_функции (<тип > имя параметра, …)

За заголовком функции должно следовать тело функции, т.е. операторы, реализующие функцию. При этом они оформляются подобно операторам основной программы, т.е. тело функции начинается с { и заканчивается } (рис.9.2).

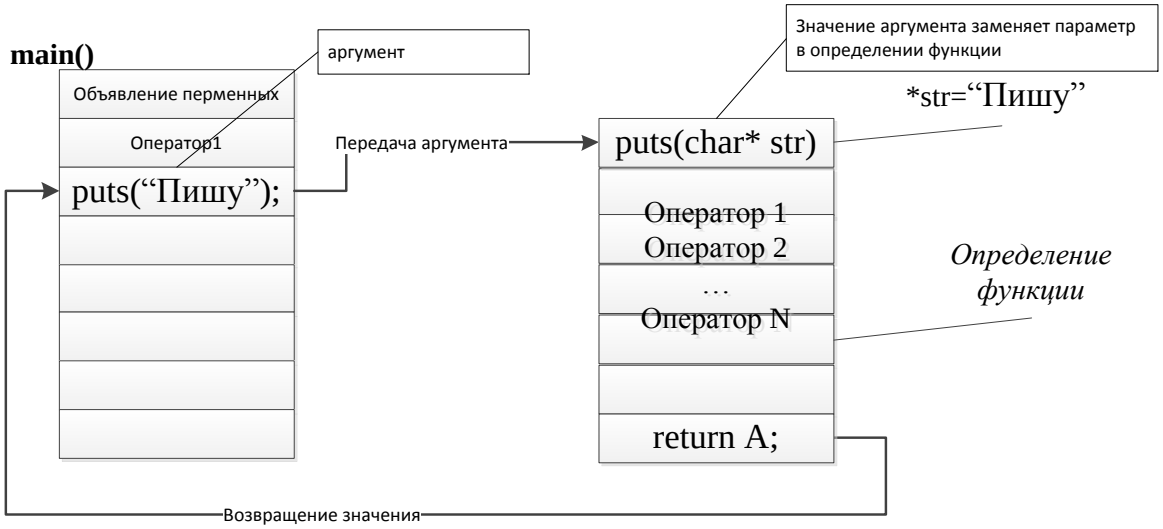
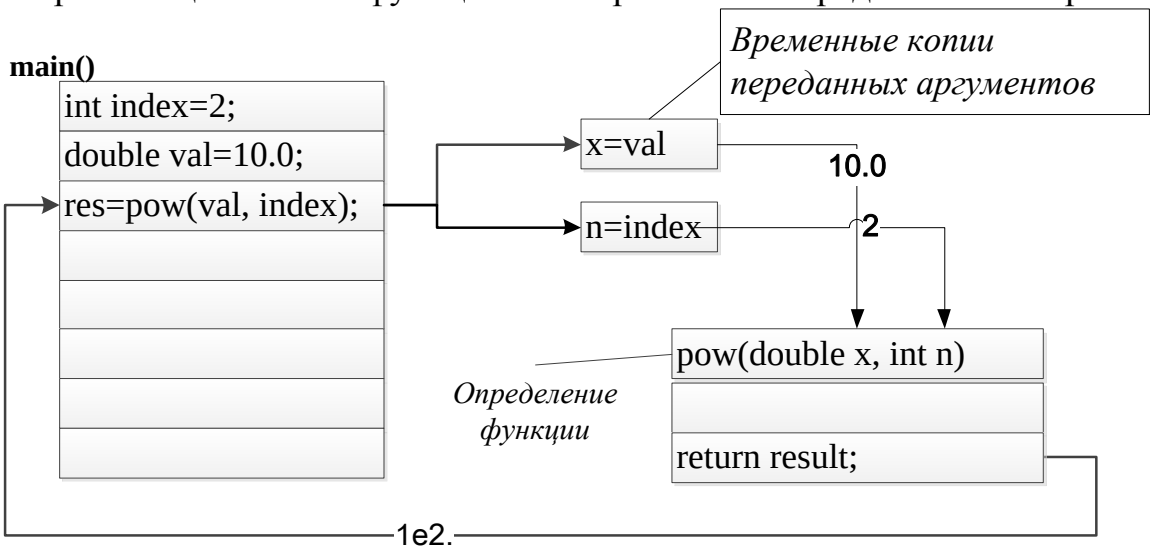


Рис.9.2. Организация программы с использованием функций

Для возврата в вызывающую функцию и для передачи ей вычисленного значения используется оператор return.

`return [выражение];`

Возвращаемое значение выражения или переменная, или константа должны быть того же типа, что и тип функции, указанный в ее описании. Это значение будет передано вызывающей программе в качестве значения функции. Схема организации вызова функции и возврата из нее представлена на рис. 9.3.



Рису.9.3. Передача параметров по значению

В отличие от возвращаемого значения количество параметров может быть любым. В правильно организованной функции использование списка аргументов (параметров) и возвращаемого значения является единственным способом связи этой функции с остальными.

При вызове функции в стеке создаются локальные переменные, соответствующие по типу формальным параметрам функции. Затем они инициализируются значениями фактических параметров, задаваемых при

вызове функции в строгом соответствии с последовательностью формальных параметров. При необходимости производится допустимое преобразование типов. Соответствие фактических и формальных параметров устанавливается не по именам параметров, а по их местоположению в списках параметров.

В языке Си нет требования, чтобы определение функции обязательно предшествовало вызову функции. Однако, чтобы компилятор мог выполнить проверку соответствия типов передаваемых аргументов типам формальных параметров до вызова функции необходимо поместить ее объявление (прототип).

Объявление (прототип) функции имеет тот же формат, что и определение но не имеет тела и заканчивается точкой с запятой.

Прототипы библиотечных функций находятся в заголовочных файлах и включаются в программу с помощью директив препроцессора #include.

9.2. Правила написания функций

Правило 1. Многократно используемый код должен быть помещен в функцию. Если в программе имеется почти идентичный код, появляющийся более чем в одном месте, то этот код должен быть выделен в подпрограмму, которая вызывается из нескольких мест. Выгода состоит в меньшем размере программы, ее лучшей сопровождаемости, необходимости исправления ошибки только в одном месте.

Имя функции также дает хорошую абстракцию. Вызовы функции с хорошо выбранным именем являются обычно самодокументирующимися, так как устраняется необходимость в лишних комментариях.

Правило 2. Функции должны иметь небольшие размеры (не более 100 операторов языка программирования высокого уровня) и, желательно, целиком помещаться на экране.

При росте размера функции следует разбивать ее на более мелкие функции — составные части.

Правило 3. Функция должна решать только одну задачу. Иначе говоря, все функции должны иметь четко выраженное функциональное назначение. Если нельзя кратко описать, что делает функция, следует пересмотреть структуру программы.

Функции, по возможности, не должны использовать общие блоки данных (глобальные переменные), то есть должны получать все необходимые для работы входные данные в форме параметров вызова и возвращать выходные данные вызывающей функции в явном виде.

Иногда в функцию передают всевозможные флаги (управляющие признаки), в зависимости от которых она выполняет весьма разнящиеся операции. Этого следует избегать. Чем меньше число получаемых функцией управляющих признаков, которые в ней анализируются, тем она более функционально проста.

Правило 4. Функция должна быть оформлена. С одной стороны это сделает код читабельным, с другой стороны еще раз заставит оценить функционал.

9.3. Примеры написания и использования функций

Пример 1. Функция вычисления котангенса
(реализация до функции main())

double cotan(double x)	Объявление функции *cotan* с типом возвращаемого значения *double* и формальным параметром *x* типа *double*
{	Начало функции
double ctg;	Объявление текущей переменной для результата
ctg = 1.0 / tan(x);	Вычисление тангенса с использованием вызова стандартной функции *tan(x)* и присваивания результата переменной *ctg*
return ctg;	Возращение вычисленного значения
}	Конец функции

Вариант кода без использования дополнительных переменных

```
double cotan( double x )
{
return 1.0 / tan(x);
}
```

Пример 2. Вычисление биномиальных коэффициентов $C_n^m = \frac{n!}{m!(n-m)!}$

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

Описание функции:

```
int fact(int k) //вычисление факториала
{
int i, j;
for(i=1, j=1; i<=k; i++) j*=i;
return j;
}// конец определения функции
/* Вычисление биномиального коэффициента
void main()
{
int n, m, nmc, nm;
while(1)
 {
printf ("Введите n: ");
scanf("%d", &n);
printf ("Введите m:");
scanf("%d", &m);
```

```
    if (m>=n && n>=m && n<10) break;
        printf ("Должно быть 0<=m<=n");
    }
    nm=n-m;
    nmc=fact(n)/fact(m)/fact(nm);//вызов функции
    printf ("\n Биномиальный коэффициент c (%d из %d)
=%d", m, n, nmc);
    }// конец программы
```

Вариант реализации факториала через рекурсивную функцию

```
long fact(int n)
{
if(n<0) return 0;
return ( (n==0) ? 1 : n*fact(n-1)  );
}
```

9.4. Практические задания

1. Создайте собственную функцию рисования заданной геометрической фигуры. Эта функция будет выполнять только вывод заданного символа и не будет иметь результирующего значения, поэтому ее тип следует объявить как void. В качестве аргумента в функцию можно передавать символ, которым рисуется фигура типа char.

Вариант	Фигура	Заполнение контура
1	квадрат;	да
2	прямоугольник	нет
3	прямоугольный треугольник;	да
4	равнобедренный треугольник;	нет
5	равнобедренная трапеция;	да
6	ромб	нет
7	остроугольный треугольник;	да
8	тупоугольный треугольник;	нет
9	выпуклый четырехугольник;	да
10	самопересекающийся четырехугольник;	да

Прототип функции имеет вид:

```
void paint (char c);
```

Его следует объявить после директив препроцессора.

После функции main следует описать саму функцию:

```
void paint (char c)
{
int i;
for (i=0; i<5;i++) putchar (c);
}
```

Аналогично объявите функцию вычисления площади фигуры:

```
float calc(int a, int b);
```
Реализуйте саму функцию:
```
float calc(int a, int b)
{
float sq;
    sq=1.*a*b;
    return sq;
}
```
Обратите внимание, что данная функция возвращает значение типа float в вызывающую ее функцию.

2. Проверьте их работу, разместив в функции main() их вызов с фиксированными значениями параметров и выводом результатов на консоль. Обязательно выведите название фигуры.
```
main()
{
short a;
printf ("Фигура\n выберите 1- рисовать, 2 – считать,
3- выход");
switch(a)
{
    case 1: paint('*'); break;
    case 2: calc(5,4); break;
    default: return;
    }
return;
}
```
3. Напишите программу, которая по требованию пользователя рисует фигуру заданное число раз и вычисляет площадь с произвольно заданными параметрами.

9.5. Контрольные задания

Написать программу вычислительного эксперимента для оценки погрешности приближенного вычисления функции через конечные ряды f(x)=|*Y*(*x*)–*S*(*x*)|. S(x) реализовать в виде отдельной функции с парамтерами x и N, где N – число членов ряда.

Результат оформить в виде таблицы:

| N | x | f(x) | f(x)/ *Y*(*x*)*100 |

где x изменяется в интервале от 0,1 до 1 с фиксированным шагом 0,1, а N изменяется кратно 10 от начального значения, при котором значения *Y*(*x*) и *S*(*x*) различаются во втором знаке после запятой.

Варианты функций и ряда для приближения.

1. $S(x)=\sum_{k=0}^{n}(-1)^k\frac{x^{2k+1}}{(2k+1)!}$, $Y(x)=\sin(x)$.

2. $S(x)=\sum_{k=1}^{n}(-1)^{k+1}\frac{x^{2k}}{2k(2k-1)}$, $Y(x)=x\cdot arctg(x)-\ln\sqrt{1+x^2}$.

3. $S(x)=\sum_{k=0}^{n}\frac{\cos(k\pi/4)}{k!}x^k$, $Y(x)=e^{x\cos\frac{\pi}{4}}\cos(x\sin(\pi/4))$.

4. $S(x)=\sum_{k=0}^{n}(-1)^k\frac{x^{2k}}{(2k)!}$, $Y(x)=\cos(x)$.

5. $S(x)=\sum_{k=0}^{n}\frac{\cos(kx)}{k!}$, $Y(x)=e^{\cos x}\cos(\sin(x))$.

ЛАБОРАТОРНАЯ РАБОТА №10
СОЗДАНИЕ ОДНОМЕРНЫХ МАССИВОВ

Цель работы: Получение практических навыков работы с одномерными массивами

Программные средства: MICROSOFT VISUAL STUDIO

10. 1 Теоретические сведения

Наиболее распространенным производным типом данных является массив. Массив – структура данных, представляющая совокупность элементов одного и того же типа, проиндексированных порядковыми номерами.

Одномерный массив представляет собой последовательность однотипных значений, объединенных общим именем. Его можно представить линейной таблицей, где каждому значению соответствует его порядковый номер.

Порядковый номер	***0***	***1***	***2***	***3***	***…***	***i***	***…***	***n-1***
Значение	-5	19	80	-127	…	127	…	-124
	Массив из n элементов							

Порядковые номера элементов называются индексами, и обращение к каждому конкретному элементу происходит по его индексу.

Описание массива производится аналогично другим переменным – тип данных, имя, но при этом за именем массива в квадратных скобках должна быть записана целая положительная константа или константное выражение, равное размеру этого массива, то есть максимально возможному числу элементов.

Объявление массива:

```
int a[100], ab[2*40];
```

```
double c[200], speed[NVAL];
char name[20];
```

Описание массива определяет его размещение в памяти и имя массива без квадратных скобок за ним равно адресу первого элемента этого массива.

Обращение к элементам массива осуществляется путем индексации, т.е. использования индексированных переменных, состоящих из имени массива и порядкового номера элемента в массиве (индекса).

```
Обращение к элементу массива:
ab[i]=10+i*100;//задание значения i-му элементу
a=5/ab[0];//использование значения нулевого элемента
```

Следует помнить, что в языке Си ***нумерация элементов массива начинается с нуля.***

Индексированные переменные могут использоваться в любых выражениях в тех местах, где допускается применение переменных соответствующих типов.

```
Использование элементов массива:
B=c[5]+1;
y=2*sin(speed[19]/180*M_PI);
```

При работе с индексированными переменными необходимо внимательно следить за тем, чтобы индексы не вышли из допустимого диапазона, определяемого описаниями массивов.

Для инициализации массива за его именем располагают знак присваивания и список инициализации, который представляет собой заключенные в фигурные скобки и разделенные запятыми инициализирующие значения. Константа в списке инициализации должно быть не больше, чем объявленный размер массива. Если их меньше, то элементы, для которых нет констант, обнуляются. Для инициализируемого массива допускается вообще не указывать размер. В этом случае размер массива определяется по количеству констант в описании.

```
Инициализация массива:
int a[4]     =    { 15, 21, 1, 304 };
int c[] = { 1, 15, 18, 11, 20 };
```

Индекс	***0***	***1***	***2***	***3***	***4***
Значение	1	25	18	11	20

Массив c[]

Для ввода и вывода элементов массива в языке Си не предусмотрено специальных средств, поэтому для однотипной обработки удобно использовать циклыспараметрами for.При инициализации задается начальный индекс, в условии проверяется, не выходит ли значение индекса за границы массива. В теле цикла на каждой итерации изменяется индекс и выполняется операция с индексируемым элементом массива.

```
Вывод массива:
```

```
for (int i=0;i<4;i++)
     printf("\n a[%d]= %d", i, a[i]);
```

10.2 Приемы инициализации и заполнения массивов случайными числами

При отладке программ возникает необходимость заполнить массив некоторыми значениями, если массив небольшой, то это можно сделать явно:

```
double a[]={-12.6, 2e-2, 123, 2.1e14, 2.5};
```

Или заполнить константами в цикле:

```
for(i=0; i<n; i++)a[i]=MYCONST;
```

Но лучше заполнить случайными числами. Случайные числа могут быть сгенерированы функцией стандартной библиотеки Си rand(), которая генерирует числа в диапазоне от 0 до RAND_MAX и возвращает число типа unsigned long. Функция rand() не имеет аргументов. Константа RAND_MAXи прототип функции rand() объявлены в <**stdlib.h**>.

Так как возможна генерация числа только по определенному алгоритму, то для превращения функции rand в генератор действительно случайных чисел, нужно его каждый раз инициировать неким случайным числом. Для этого следует использовать функцию **srand (unsigned seed)**, которая в качестве аргумента **seed** получает некоторое число, которое определяет последовательность генерации случайных чисел функцией rand.

```
srand (12);
```

Если этим числом будет системное время, тогда при каждом запуске программы будет происходить инициализация генератора «новым значением», и, как следствие, создаваться неповторяемаяпоследовательность чисел.

```
/* объявление необходимых переменных */
int stime;
longl time;
/* получает текущее календарное время */
  ltime = time(NULL);
/* преобразование его к типу int */
stime = (unsigned) ltime/2;
/* инициализация генератора */
srand(stime);
```

За одно обращение к функции rand можно получить одно случайное число в диапазоне от 0 до RAND_MAX. Для получения случайного целого числа **m** в заданном диапазоне от **A** до **B**, необходимо выполнить преобразование:

```
m = A+ rand() % (B -A);
```

Если необходимо заполнить массив случайными числами вещественного типа, то вещественное число в интервале от 0 до 1 может быть получено:

```
x=1.* rand()/RAND_MAX
```

Заполнение массива случайными вещественными числами в интервале от

XMIN до XMAX может быть организовано

```
for(i=0; i<N; i++)
a[i]=XMIN+1.f*(XMAX-XMIN)*rand()/RAND_MAX;
```

Константы диапазонов генерации чисел удобно задавать с использованием директив препроцессора:

```
#define XMAX 112.4
```

Подобное управление размерами и параметрами массива на этапе препроцессорной обработки позволяет легко менять их размер и другие связанные с ними параметры.

```
#define NMAX 10 //определение размера массива
main()
{
int value[NMAX];
for(i=0;i<NMAX; i++) value[i]=100-i*10;
}
```

10.3 Примеры работы с массивами

Пример 1. Ввод элементов массива с клавиатуры

```
#include<stdio.h>
void main(void)
{
double a[100], c;
intn, i;
printf("Введите количество чисел n = ");
scanf("%d", &n);
if( n>(sizeof a)/sizeof(double) )
{ printf("Слишком много элементов\n"); return; }
for(i=0; i<n; i++)
{
printf("a[%d] = ", i);
scanf("%lf", &c);
a[i]=c;
}
  /* Операторы, обрабатывающие массив */
     }
```

Во многих случаях удобно возложить на программу подсчет числа элементов, вводимого массива, при этом ввод завершается при появлении во входном потоке признака конца данных. Таким признаком в следующей программе служит число большее 1.0e300

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

```
double a[100], temp; int n, end;
for(end=n=0; n<(sizeof a)/sizeof(double); n++)
{
printf("a[%d] = ", n); scanf("%lf", &temp);
if( temp>=1.0e300 ) { end=1; break; }
a[n] = temp;
 }
if( end )
{
/* Операторы, обрабатывающие массив */
}
else
printf("Переполнение массива\n");
}
```

Пример 2. Инициализация массива случайными числами

```
#include<stdlib.h>
#include <stdio.h>
#define XMIN 5
#define XMAX 15

void main(void)
{
int N, i;
float a[100];
printf("Введите количество чисел n = ");
scanf("%d", &N);
if(N>100)
{ printf("Слишком много элементов\n"); return; }
for(i=0; i<N; i++)
    a[i]=XMIN+1.f*(XMAX-XMIN)*rand()/RAND_MAX;

// Обработка массива
}
```

Пример 3. Печать массива, заполненного случайными числами

```
main()
{
int s[10];
printf("Массив заполнится числами в заданном
диапазоне от -10 до 20: \n");
srand(time(NULL)|clock());
for (int i=0;i<10;i++)
    s[i]=-10+rand()%31;
printf("получен массив с целочисленными
```

```
элементами\n");
    for (int i=0;i<10;i++)
         printf("\n s[%d]= %d", i, s[i]);
    }
```

Комментарий к примеру

Функция srand осуществляет инициализацию генератора случайных чисел. Функция time возвращает текущее календарное время системы. Эта функция вызывается с нулевым указателем. Функция clock возвращает приблизительное процессорное время, потраченное на работу с программой. Таким образом, в качестве аргумента функции srand всегда будет новое неповторяющееся значение. Объявления этих функций в заголовочных файлах ctime.hи stdlib.h.

Пример 4. Формирование нового массива со значениями элементов, равными корню квадратному из значений исходного массива и вывод их на печать в виде таблицы

```
main()
{
int A[10],B[10], i;
// Инициализация массива A
printf("Индекс | Новый | Старый");
for(i=0; i<10; i++)
{
 B[i]=sqrt(A[i]);
printf("\n%-7d|%7.2f|%7.2f", i, B[i], A[i]);
}
return;
}
```

10.4. Практические задания

1. Объявите массив вещественных чисел из 10 элементов:

```
float A[10];
```

2. Напишите программу заполнения массива с консоли.

```
for(int i=0; i<10; i++)
{
printf("a[%d] = ", i+1);
scanf("%lf", &A[i]);
}
```

3. Выполните указанное преобразование над элементами массива:

1.	удвоить значение каждого элемента массива;
2.	увеличить на 1 значение каждого второго элемента массива;
3.	обнулить отрицательные элементы массива;
4.	возвести в квадрат значение каждого элемента массива;
5.	уменьшить в 10 раз значение каждого элемента массива;

6.	изменить знак каждого элемента массива;
7.	уменьшить на константу значение каждого элемента массива;
8.	отбросить дробную часть у всех элементов массива;
9.	установить в 1 значения всех четных элементов массива;
10.	округлить в большую сторону все элементы массива;
11.	к каждому элементу массива прибавить значение последнего элемента;
12.	заменить нулевые значения элементов массива их порядковым номером.

4. Выведите результаты на экран в виде таблицы, в первом столбце которой указан номер элемента, во втором – исходный элемент, в третьем – преобразованный.

5. Сгенерируйте массив из 100 случайных вещественных чисел в интервале от -1 до 1 и выполните над ними указанное выше преобразование.

6. Выведите результаты в виде строки, содержащей имя массива, индекс и исходное значение и преобразованное.

10.5. Контрольные задания

1. В одномерном массиве, состоящем из *100* случайных чисел в диапазоне от -5 до 5 найти сумму положительных элементов.

2. Заполнить массив из Nэлементов по правилу N-i, где i – порядковый номер элемента при счете по порядку от 1. Вывести на печать индекс, значение элемента и накопленную сумму элементов массива.

3. В одномерном массиве, состоящем из *N*введенных с клавиатуры элементов найти сумму значений, расположенных между первым и последним нулевыми элементами.

4. В одномерном массиве, состоящем из *20* случайных целых чисел от 0 до 100, найти и напечатать порядковые номера элементов, значения которых больше заданного числа A.

5. Задан одномерный массив. Сформировать новый массив значений, в котором каждое значение равно сумме двух соседних элементов исходного массива, деленной на 2.

ЛАБОРАТОРНАЯ РАБОТА №11
АДРЕСНАЯ АРИФМЕТИКА

Цель работы: Формирование понимание сущности указателя в языке Си и особенностей их использования при разработке программ

Программные средства: MICROSOFT VISUAL STUDIO

11.1 Теоретические сведения

Указатель– переменная, значением которой является адрес памяти, по которому хранится объект определенного типа. Так как указатель – это адрес некоторого объекта, то он позволяет обращаться к данным, которые по этому адресу расположены. Указатель может указывать на обычную переменную, массив, строку, структуру, другой указатель или функцию. Таким образом, указатели представляют прямой быстрый доступ к памяти и являются альтернативным способом обращения к переменным, массивам, структурам данным и функциям программы.

Для идентификации указателей используют *. При объявлении указателей всегда указывается тип объекта, который будет храниться по данному адресу.

Объявление указателя:

<тип переменной> * name;

где name – переменная, объявляемая как указатель.

По этому адресу (указателю) хранится значение указанного типа.

Основные операции с указателями:

&– операция взятия адреса;

* – операция получения значения по заданному адресу (операция разадресации).

Так, если объявлена переменная **floata;** то оператор**adr_a=&a;** записывает в переменную adr_a адрес переменной a, переменная adr_a должна быть указателем на тип float и объявлена следующим образом: **float *adr_a;**

Операция ***** выполняет действие обратное операции *&*. Она возвращает значение переменной, хранящееся по заданному адресу. Поэтому оператор **a=*adr_a;** записывает в переменную a вещественное значение, хранящееся по адресу adr_a.

В общем случае выражения с указателями подчиняются тем же правилам, что и обычные выражения, допустимы:

– присваивание;

– преобразование типов;

– сложение и вычитание.

Если указатели одного типа, то можно присвоить один другому с помощью обычной операции присваивания.

Присваивание значения указателю:

```
int x = 99; //объявление переменной
int *p1, *p2;//объявление указателя
  p1 = &x;// получение адреса переменной и присваивание его указателю
   p2 = p1;//присваивание значений
```

Если указатели ссылаются на различные типы, то при присваивании значения одного указателя другому, необходимо использовать преобразование типов.

```
double x = 100.1;
int  *p;
/* указателю на целое p присваивается значение, ссылающееся на double */
p = (int *) &x;
```

При подобных присвоениях необходимо следить за размерами переменных. Если переменная с более длинного типа будет преобразовываться на тип с меньшими размерами, то значение переменной будет урезаться. *Например, предыдущий код будет неверен, так как истинное значение переменной теряется (double = 4 байтам, а int = 2 байта).*

Нельзя создавать переменную типа void, но можно создавать указатель на тип void. Указателю на void можно присвоить указатель любого другого типа. Однако, при обратном присваивании необходимо использовать явное преобразование указателя на void.

```
void *pv;
float f, *pf;
pf = &f;
pv = pf;
pf = (float*)pv;
```

Результат прибавления к указателю или вычитания из него целочисленной величины является указателем того же типа, значение которого отличается от значения исходного указателя на число байт, определяемое произведением целочисленной величины на размер данного, которое адресует указатель.

```
<тип переменной> * ptr;
ptr++=содержимое ptr+sizeof(тип переменной);
ptr--=содержимое ptr-sizeof(тип переменной);
ptr+i=содержимое ptr+i*sizeof(тип переменной);
ptr-i=содержимое ptr-i*sizeof(тип переменной);
```

Так значения *(pa+i) и *pa+i не равны, если pa – это адрес переменной типа int, например 0x10, и i=2, то операция *(pa+i) соответствует обращению по адресу (0x100+2=0x102), а *pa+i вызовет переход через два адреса (2*размер целого, равного 2 байта, т. е. по адресу 0x104.

К указателем можно применять только операции сложения и вычитания с целым числом, остальные операции недопустимы.

Указатели допускается использовать в операциях сравнения. При этом всегда возможно сравнение указателя с нулем и сравнение двух однотипных указателей.

Для проверки корректности инициализации указателя перед его использованием, рекомендуется сравнить значение указателя с нулем.

```
Сравнение указателей:
int *p; . . .
if (p == NULL) exit(-1);.
```

Значение NULL обычно используется в программе для указания на

специальное условие.

11.2 Способы инициализации указателей

Инициализации указателей корректными значениями адресов памяти необходимо уделять повышенное внимание. Дело в том, что указатели можно инициализировать точно так же, как и обычные переменные, например:

```
int *p1 = (int *)1242, *p2 = (int *)1246; // преобразования целого значения к указателю на целое значение (к адресу)
int *page=0xB800;// указание на известный адрес начала видеобуфера цветного дисплея
```

Такая инициализация возможно вызовет предупреждающее сообщение компилятора о том, что нарушается мобильность программы, но программа будет скомпилирована, отредактирована и выполнена. Такое возможно, если точно известно, что в этих адресах имеются нужные данные целого типа и возможно получение доступа к ним через указатели. В противном случае результат не предсказуем. Более корректными являются следующие способы инициализации указателей.

Первый способ инициализации: описать переменные программы. Поскольку компилятор отводит память под переменную в момент ее описания, то присваивание указателю адреса переменной гарантирует, что нужная память отведена и там будут находиться значения переменных, например:

```
float r, b;
float *ukr = &r, *ukb = &b;
```

Второй способ: присвоить указателю значение другого указателя, к этому моменту уже правильно инициализированного, например:

```
char *uk_simv_rus, *uk_simv_lat;
char simvsmoll, simvlarge;
 . . . . . .
uk_simv_lat = &simvsmoll;
uk_simv_rus = uk_simv_lat;
```

Разные указатели содержат адреса одной и той же переменной. Значит, к переменной simvsmoll можно обратиться как через указатель uk_simv_rus так и через указатель uk_simv_lat, что зачастую предоставляет программисту дополнительные удобства.

Третий способ: использовать одну из стандартных функции распределения памяти, например, malloc.

Функция malloc(size) возвращает указатель на первый байт области памяти размером size, которая была выделена из динамически распределяемой области памяти.

```
int *x = (int *)malloc(sizeof(int));
```

11.3 Примеры работы с указателями

Пример 1. Реализация операции присваивания указателей

```
intx = 1, y, z = 3;
int *p, *q;
p = &x;
printf("%d\n", *p); // на печать 1
y = *p;
printf("%d\n", y); // на печать 1
*p = 0;
printf("%d %d\n", x, y); // на печать 0 1
q = &z;
printf("%d\n", *q); // на печать3
p = q;
*p = 4;
printf("%d\n", z); // на печать4
printf("%p %p\n", p, q); // на печать 4 4
```

Пример 2. Реализация операции преобразования типов

Если указатели ссылаются на различные типы, то при присваивании значения одного указателя другому, необходимо использовать преобразование типов. Без преобразования можно присваивать любому указателю указатель void *.

```
int main()
{
float PI=3.14159,*p1;
double *p2;
//В переменную p1 записываем адрес PI
p1=&PI;
//указателю на double присваиваем значение, которое ссылается на тип float.
p2=(double *)p1;
printf("По адресу p1=%p хранится *p1=%g\n",p1,*p1);
printf("По адресу p2=%p хранится *p2=%e\n",p2,*p2);
}
```

По адресу p1=0012FF7C хранится *p1=3.14159

По адресу p2=0012FF7C хранится *p2=2.642140e-308

В указателях p1 и p2 хранится один и тот же адрес, но значения, на которые они ссылаются, оказываются разными. Это связано с тем, что указатель типа *float адресует 4 байта, а указатель*double – 8 байт. После присваивания ***p2=(double *)p1***; при обращении к *p2 происходит следующее: к переменной, хранящейся по адресу p1, дописывается еще 4 байта из памяти и в результате значение *p2 не совпадает со значением *p1.

Преобразование типов в сторону уменьшения размера отводимой памяти

также приведет к некорректным результатам. Так приведенная ниже программа выделит младший байт шестнадцатеричного числа 0x8e41.

```
int main()
{
    int i, *ptr;
    i = 0x8e41;
    ptr = &i;
    printf("%x\n", *ptr); // на печать 0x8e41
    printf("%x\n", *((char *)ptr)); //на печать 0x41
}
```

Пример 3. Сравнение указателей

В общем случае сравнение двух указателей одной из операций отношения имеет смысл, если оба адресуют общий для них объект.

```
int main()
{
inta, b;
int *p1, *p2, *p3;
 p1=&a; p2=&b;p3=p1;
if(p1!=p2) printf ("Указатели p1:%p и p2:%p неравны",
p1, p2);
if(p1==p3) printf ("Указатели равны");
}
```

11.4 Практические задания

1. Определите объем памяти, выделяемой под указатель на переменные типа char, int, double.

Под сам указатель (там, где хранится адрес) также должна быть выделена память. Объем этой памяти можно узнать с помощью функции sizeof():

```
int *pi;
float *pf;
printf("%lu\n", sizeof(pi));
printf("%lu\n", sizeof(pf));
```

Под указатели всех типов выделяется одинаковый объем памяти, т. к. размер адреса не зависит от типа, а зависит от вычислительной системы. Однако по типу данных определяется, сколько ячеек памяти занимает значение, на которое ссылается указатель, и через сколько ячеек начнется следующее значение.

2. Проанализируйте приведенный ниже код. Объясните результат.

```
int main()
{
float PI=3.14159,*p1,*p2;
p1=p2=&PI;
printf("Поадресур1=%pхранится *p1=%g\n",p1,*p1);
```

```
printf("По адресу p2=%p хранится *p2=%g\n",p2,*p2);
}
```

! Обратите внимание на спецификатор формата **%p**. Он используется для вывода значений указателей в функции printf. Но поскольку указатель – это адрес, т.е. целое положительное число, то в зависимости от ситуации можно использовать любые спецификаторы целочисленных переменных. Для наглядности результатов используйте **%x**– для вывода шестнадцатеричном коде.

3. Выполните приведенный ниже код и определите, на сколько увеличится значение, на которое указывает **pa**. Полученные результаты поясните.

```
int *pa, x;
pa = &x;
pa++;
```

4. Выполните приведенный ниже код, выведя на печать все задействованные переменные до и после выполнения арифметических операций. Поясните, что получается в результате операции инкрементирования и декрементирования.

```
double *p1;
float *p2;
char *i;
p1++;
p2--;
i++;
```

Перед выполнением операций обязательно выполните инициализацию указателей через описание переменных (см. первый способ инициализации).

11.5 Контрольные задания

Напишите программу побайтного вывода заданного числа, начиная с младшего или старшего байта по желанию пользователя. Для наглядности используйте шестнадцатеричный вывод. Реализуйте варианты с использованием чисел разного типа:

Тип	Типичный размер в битах	Тип	Типичный размер в битах
int	16 или 32	unsigned long int	32
unsigned int	16 или 32	unsigned long long int	64
short int	16	**float**	32
unsigned short int	16	**double**	64
long int	32	long double	80
long long int	64		

ЛАБОРАТОРНАЯ РАБОТА №12
РАБОТА СО СТРОКАМИ ЧЕРЕЗ УКАЗАТЕЛИ

Цель работы: Формирование навыков работы с массивами символов (строками) посредством указателей.

Программные средства: MICROSOFT VISUAL STUDIO

12. 1. Теоретические сведения

В языке Си строка – это одномерный массив символов, заканчивающийся нулевым байтом, каждый бит которого равен нулю, при этом для нулевого байта определена константа ´\0´ (признак окончания строки или нуль-терминатор).

Строки можно инициализировать с точным указанием размера строки, равного количеству символов в строке плюс константа ´\0´:

```
char S1[10] = ”123456789”;
```

посимвольная инициализация:

```
char S2 [10]={'y','e','s','\0'};
```

или неявно, когда размер строки устанавливается по фактическому количеству символов:

```
char S3[ ] = ”12345”;
```

Для доступа к элементам массива существуют два способа:

1) с использованием индексных выражений – **`S[i]`**;

2) с использованием указателя на массив –**`*(S+i)`**.

Поскольку начальный адрес элемента массива – это его имя, т. е. указатель на нулевой элемент, то можно обращаться к элементам массива, используя указатели.

Указатели	*S	*(S+1)	*(S+2)	*(S+3)	*(S+4)	*(S+5)	
Адреса	S	S+1	S+2	S+3	S+4	S+5	
Значения	***‘s’***	***‘t’***	***‘r’***	***‘i’***	***‘n’***	***‘g’***	***‘\0’***
Индексы	0	1	2	3	4	5	

Выражения *(S+i) иS[i] ссылаются на одно и то же значение и эти обе формы доступа к массиву можно сочетать в одном операторе.

Ввиду отсутствия средств языка Си для работы с массивами в целом, стандартная библиотека обладает богатым и разнообразным набором функций для обработки строк и символов. Строковые функции работают с массивами символов (строками), заканчивающимися символом конца строки. Поскольку указатель обеспечивает эффективную передачу массива для обработки вызываемой функции, аргументами функций для работы со строками являются указатели.

Для ввода строк с консоли можно использовать функцию:

```
char* gets(char * str)
```

где str – это массив символов. Функция gets() возвращает указатель на str.

Функция gets() читает последовательность символов, введенных с клавиатуры, и помещает их по адресу, указанному в аргументе. Можно набирать символы, пока не будет нажат ввод. Символ, соответствующий клавише ввод – ‘\n’, не станет частью строки. Вместо этого в конце строки появится нулевой символ, и gets() закончит работу.

Используя gets(), можно перейти границы массива. Это возможно, поскольку не существует способа указать gets(), где находится граница массива. Например, если вызвать gets() с массивом длиной в 40 байт, а затем ввести 40 или более символов, то произойдет выход за пределы массива. Это, естественно, вызывает проблемы и часто может привести к краху системы. В качестве альтернативы можно использовать функцию fgets().

Для обработки строк можно использовать различные функции описанные в string.h, наиболее часто используемые функции представлены ниже в таблице 12.1.

Таблица12.1

Наиболее часто используемые функции работы со строками

Прототип	Возвра-щаемое значение	Описание
strcpy(char * S1, const char * S2)	char *	копирует содержимое строки S2 в строку S1
strcat(char * S1, const char * S2)	char *	присоединяет строку S2 к строке S1 и помещает ее в массив, где находилась строка S1
strcmp(const char * S1, const char * S2);	int	сравнивает строки S1 и S2 и возвращает 0, если строки равны
strlen(const char *S);	size_t	возвращает длину строки, т.е. количество символов, начиная с первого и до нуль-терминатора, который не учитывается
strstr(const char *S1, const char *S2);	char *	указывает первое появление подстроки S2 в строке S1

Если требуется работа с отдельными символами, то также следует пользоваться средствами стандартной библиотеки. Так функция ***tolower(int ch)*** изменяет символ-букву ch на строчную, ***toupper(int ch)*** – на прописную, а функция ***isdigit(int ch)*** возвращает ненулевое значение, если ее аргумент ch является цифрой. Эти функции объявлены в ctype.h.

12. 2. Примеры работы со строками

Пример 1. Программа, которая выводит шестнадцатеричный и десятичный код введенного с клавиатуры символа в виде эхо.

```
#include <stdio.h>
#include <process.h>
#include <conio.h>

void main()
{
unsigned char ch;
printf("Введитесимвол.\n");
printf("Для завершения введите TAB.\n");
do
{
ch = getche();// считывание символа
printf("Символ: %c Код: %d Шестнадцатеричный код: %x
\n", ch, ch, ch);//вывод
}
while(ch!='\n');
system ("pause");
}
```

Обратите внимание, функция getchar() буферизирует ввод до нажатия ENTER, поэтому при вводе данных после первого введенного символа для начала работы программы потребуется нажатие клавиши ENTER. Причем, если последовательно нажать несколько символов, программа получит тот, который предшествовал нажатию ENTER. Эта особенность работы с потоками подсистемы ввода-вывода операционной системы, поэтому в примере использована функция getche(), которая в отличие от getch() отображает введенный символ.

Пример 2. Программа, вычисляющая длину введенной с клавиатуры строки.

```
#include <stdio.h>
void main()
{
char Str [80]; // введенная строка
int i = 0; // длина строки
puts("Введите строку и нажмите <Enter>");
printf("->");
gets (Str);
while(Str[i++]);
printf("Длина введенной строки: %i\n", i);
}
```

Пример 3. Программа, приветствующая пользователя

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

```
int main(void) {
    //объявление строк
charName[20];
charSnew []="Привет, ";
puts("Привет!  Как  тебя  зовут?\n");//приглашение  к
вводу
scanf ("%s\n", Name);//считывание строки с клавиатуры
printf("%s%s !\n", Snew, Name);//вывод двух строк
return 1;
}
```

Пример 4. Программа деления строки на слова (в качестве разделителя используется запятая)

С помощью функции strtok() можно разбить строку на отдельные части (лексемы). Объявление этой функции выглядит так char *strtok (char * str, const char *deliment). При первом вызове функции в качестве первого параметра указывается строка str, которую требуется разбить, далее строка-разделитель – deliment, т.е. лексема. При последующих вызовах функции для этой же строки первым параметром должен быть NULL, т.к. функция уже работает со стеком параметров. Функция возвращает указатель на это слово, который должен быть присвоен объявленной заранее переменной типа указатель на char.

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

int main(void)
{
char str[] = "one, two, three, four, five";
char *sp;
sp = strtok(str, ", ");
while (sp)
{
puts(sp);
sp = strtok(NULL, ", ");
}
return 1;
}
```

Пример 5. Две альтернативные реализации функции копирования строк с использованием массивов и с использованием указателей

```
char *strcpy(char s1[], char s2[])
{
int i;
for(i=0; s1[i]=s2[i];i++);
return (s1);
```

```
}
char *strcpy(char *s1, char *s2)
{
char *str=s1;
while(*s1++=*s2++);
return (str);
}
```

12.3 Способы преобразования чисел в строки

Для преобразования чисел в строки можно использовать:

1) собственно разработанные программы;

2) функцию форматированного вывода sprintf();

3) функции itoa, ltoa, которые преобразуют целое int и длинное целое long в строку, но они не входят в стандарт Си, и могут быть недоступны на некоторых платформах.

Наиболее универсальный, хотя и не типо безопасный вариант - sprintf(). Функция sprintf() идентична функции printf() за исключением того, что поток вывода записывается в массив, адресуемый указателем buf, а не в стандартный поток. По окончанию работы функции этот массив будет завершаться символом конца строки (нуль-символом). Это позволяет формировать строки из данных различного типа, не прибегая к другим функциям преобразования.

Прототип функции имеет вид:

```
int sprintf(char *buf, const char *format, ...);
```

где buf – указатель на строковый массив;

format – управляющая строка, подробно описана в 2.2;

... – список аргументов, на которые указывают спецификаторы в управляющей строке.

Например, оператор с функцией sprintf:

```
sprintf (buffer, "%d plus %d is %d", a, b, a+b);
```

формирует новую строкуbuffer, которая может быть использована в других функциях с одним аргументом типа строки.

Вариант реализации itoa для представления чисел в различных системах счисления в виде собственной функции, возвращающей указатель на строку[1]:

```
char* itoa(int val, int base){
static char buf[32] = {0};
int i = 30;
for(; val && i ; --i, val /= base)
buf[i] = "0123456789abcdef"[val % base];
return &buf[i+1];

}
```

12. 4 Практические задания

1. Напишите программу, которая запрашивает имя пользователя и здоровается с ним, используя введённое имя.

Для начала определим последовательность шагов выполнения задания.

Шаг 1. Вывод на экран строки с вопросом.

Шаг 2. Считывание введенной пользователем строки.

Шаг 3. Формирование новой строки, содержащей «приветствие» и введенную пользователем строку.

Шаг 4. Печать новой строки.

Затем для каждого шага выберем языковую конструкцию или функцию.

Шаг 1. – функция printf(“Приглашение к вводу имени»);

Шаг 2. – функция scanf(“%s”, Name);

Шаг 3. – функция strcat(Snew, Name);

Шаг 4. – функция printf (“%s!”, Snew);

В программе необходимы две строки:Snew, содержащая приветствие и Name, содержащая введенное имя.

Начинать написание программы следует с включения заголовочных файлов, использованных функций <stdio.h> и <string.h>, функции main, содержащей объявления строк как массивов известной размерности и шагов 1-4.

2. Запустите и отладьте программу.

3. Измените строку вывода так, чтобы все приветствие было написано заглавными буквами.

4. Добавьте к строке приветствия множество восклицательных знаков так, чтобы суммарная длина строки составляла 40 символов.

Для этого понадобится вычислять длину введенной строки, что можно сделать:

А) использовав стандартную функцию strlen(Str), которая возвращает количество символов строки без нулевого;

Б) использовав функцию sizeof для определения размера массива, которым является введенная пользователем строка в виде sizeof(S)/sizeof(char);

В) напрямую подсчитав количество ненулевых элементов в строке переменной i с использованием оператора цикла while(Str[i++]); (перебираем все элементы массива-строка, пока не встретится нулевой символ, увеличивая счетчик на 1).

5. Измените программу так, чтобы она воспринимала ответ пользователя, состоящий из имени и фамилии.

Обратите внимание, функция scanf вводит только часть строки до первого пробела, поэтому перепишите программу, используя функцию gets(S).

6. Измените первоначальный вариант программы, выводящий строку типа «Привет, Вася!» так, чтобы

1. удваивалась последняя буква введенного имени
2. первая буква имени была строчной, а остальные выводились заглавными (прописными)
3. в имени две первые буквы менялись местами (без изменения регистра ввода)
4. перед каждой последней буквой вставлялся символ подчеркивания
5. имя выводилось наоборот (Оля-Ялю)
6. в имени исчезала вторая буква
7. половина букв имени выводилась прописными, а остальные заглавными
8. после первых трех букв имени появлялись три точки, и остальная часть имени не выводилась
9. после каждой буквы имени добавлялся пробел
10 вместо буквы ‘а’ вставлялся символ @
11 каждая предпоследняя буква имени заменялась тире
12 в имени утраивалась заглавная буква

12.5 Контрольные задания

1. Составить программу, которая запрашивает название футбольной команды и повторяет его на экране со словами "— это чемпион!" случайное число раз.

2. Составить программу, которая запрашивает названия трех городов и выводит на экран самое длинное название.

3. Составить программу, которая запрашивает слово, состоящее из четного числа букв и выводит на экран его первую половину, не используя оператор цикла.

4. Составить программу, которая запрашивает название футбольного клуба и печатает его на экране "столбиком".

5. Составить программу, которая запрашивает предложение и определяет число слов в нем. Слова разделены только пробелом, без знаков препинания.

6. Составить программу, которая запрашивает предложение и заменяет в нем все пробелы на символ подчеркивания.

7. Составить программу, которая удаляет из строки программы на языке Си комментарии, оформленные как //.

ЛАБОРАТОРНАЯ РАБОТА №13
ИСПОЛЬЗОВАНИЕ ИНТЕГРИРОВАННЫХ ТИПОВ ДАННЫХ ДЛЯ РАЗРАБОТКИ ПРОГРАММ И СОЗДАНИЯ БИБЛИОТЕК

Цель работы: Формирование навыков работы со структурами данных и создания собственных типов данных

Программные средства: MICROSOFT VISUAL STUDIO

13.1 Теоретические сведения

Под структурой данных в общем случае понимают множество элементов данных и множество связей между ними. Различаются простые (базовые, примитивные) структуры (типы) данных и интегрированные (структурированные, композитные, сложные). Простыми называются такие структуры данных, которые не могут быть расчленены на составные части, большие, чем биты. Интегрированные структуры данных конструируются программистом с использованием средств интеграции данных, предоставляемых языками программирования.

Интегрированный тип в языке Си – структура. Структуры применяют для логического объединения, связанных между собой данных различного типа. Структура имеет общее имя, которые связывает данные воедино.

товар на складе:
название
цена
количество

Для объявления структуры используется ключевое слово **struct**, за которым следует идентификатор, являющийся именем типа структуры, и список полей или элементов, заключенных в фигурные скобки.

```
struct ИМЯ_ТИПА_СТРУКТУРЫ{
                };
```

Поля структуры объявляются внутри фигурных скобок как обычные переменные, массивы или другие структуры.

Объявление структуры (переменной структурного типа) может осуществляться различными способами:

1) создание переменной структурного типа без явного указания типа

```
struct { charname[21];
float price;
int number;} goods;
```

2) создание структуры особого типа с последующим задание переменной

```
struct PRODUCT{ //особый типPRODUCT
char name[21];
float price;
int number;};
/*объявление переменных типа PRODUCT */
struct PRODUCT prod1, prod2;
/*объявление типа и переменных этого типа в одной конструкции */
struct PRODUCT { char name[21];
float price;
int number;} prod;
```

3)создание собственного типа данных с использованием оператора

typedef.

В языке Си имеется возможность создания собственных типов данных с помощью оператора **typedef** в следующей форме**:**

```
typedef  переименовываемый_тип имя_нового_типа;
/*    создание собственного типа данных      */
typedef struct { char name[21];
float price;
int number;} PRODUCTS;
```

В дальнейшем использование ключевого слова struct для задания переменной для структур этого типа не потребуется, достаточно объявить:

```
PRODUCTS product1, product2;
```

Для доступа к полю структуры используется операция "."

```
имя_структурной_переменной.имя_поля
```

Или каждый элемент структуры имеет составное имя, состоящее из имени структуры и имени поля, разделенных точкой.

С составным именем можно выполнять любые действия, разрешенные для переменных того же типа, что и элемент структуры.

```
product1.price = 20*product2.price;
scanf("%s", product1.name);
printf("цена%.2f", product1.price);
product2.name[10]='1';product2.name[11]='c';
```

Для структур одного и того же типа допускается операция присваивания:

```
product1 = product2;
```

Ввод/вывод структур, как обработка выполняется поэлементно.

```
printf("%s цена%.2f", product1.name,product1.price);
```

Структуры можно инициализировать перечислением значений элементов.

```
PRODUCTS product1={"Pen", 12.1, 200};
```

На практике эффективно использование массива структур, что позволяет организовывать базы данных.

13.2 Примеры программ с использованием структур

Пример 1. Программа, выполняющая сложение и умножение комплексных чисел.

```
#include <stdio.h>
typedef struct {
double x, y;
}complex;
main() {
complex a, b, sum, mult;
printf("Число A\n");
printf("   действ. часть: "); scanf("%g",&a.x);
printf("   мнимая часть: "); scanf("%g",&a.y);
```

```
    printf("Число B\n");
    printf("   действ. часть: "); scanf("%g",&b.x);
    printf("   мнимая часть: "); scanf("%g",&b.y);

        sum.x = a.x + b.x;
        sum.y = a.y + b.y;
        mult.x = a.x * b.x - a.y * b.y;
        mult.y = a.y * b.x + a.x * b.y;
    printf("Сумма:   %.2f+%.2fj\n", sum.x,sum.y);
    printf("Произв.: %.2f+%.2fj\n", mult.x,mult.y);
    }
```

Пример 2. Программа, иллюстрирующая работу функций со структурой.

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

    struct circle { int x, y; float dia; char color[10];
};
    struct circle new_circle();
    void cross (struct circle);

    main () {
        struct circle a;

        a = new_circle();
        cross(a);
    }

    struct circle new_circle() {
        struct circle new;

        printf("Координаты:  "); scanf("%d%d",  &new.x,
&new.y);
        printf("Диаметр: "); scanf("%f", &new.dia);
        printf("Цвет:           "); scanf("%s",
new.color);//gets(new.color);

        return new;
    }

    void cross (struct circle c) {
        double hyp;

        hyp = sqrt((double) c.x * c.x + (double) c.y *
c.y);
```

```
        printf("Расстояние от центра круга до начала
координат: %.2lf\n", hyp);
        if (hyp<= c.dia / 2) puts("Круг пересекает
начало координат");
        elseputs("Круг не содержит точки начала
координат");
    }
```

Функция new_circle() возвращает структуру, a void cross (struct circle c) — принимает ее в качестве параметра.

13.3 Практические задания

1. Напишите программу, которая выводит на экран текущее время.

Для решения этой задачи следует использовать функцию localtime(), которая возвращает указатель на структуру типа tm, описанную в time.h, как:

```
    struct tm {
    int tm_sec;     /* секунды - [0,59] */
    int tm_min;     /* минуты - [0,59] */
    int tm_hour;    /* часы - [0,23] */
    int tm_mday;    /* день - [1,31] */
    int tm_mon;     /* месяц - [0,11] */
    int tm_year;    /* год от 1900 */
    int tm_wday;     /* день недели с воскресенья - [0,6]
*/
    int tm_yday;    /* номер дня с 1 января - [0,365] */
    int   tm_isdst;         /*   флаг   летнего   времени
устанавливается >0, если <=0, то информация недоступна*/
            };
```

Поскольку typedef отсутствует, то необходимо полное объявление.Так указатель на структуру tm объявляется как:

```
    struct tm *mytime;
```

В качестве аргумента функции localtime() передается указатель на переменную специального типа time_t, который определен в time.h как long или int в зависимости от системы.

```
    time_t t;
```

Значение переменной t может быть получено функцией time(), которая возвращает текущее календарное время системы. Функцию time() можно вызывать либо с нулевым указателем:

```
    t = time(NULL);
```

либо с указателем на переменную типа time_t:

```
    time(&t);
```

В таком случае по адресу переменной, указатель на которую передан в функцию будет размещено календарное время значением типа time_t.

Поэтому последовательность шагов получения текущего времени следующая:

Шаг 1. Объявление служебных переменных:

```
struct tm *mytime;// указатель на структуру tm
time_t t;/*переменная типа time_t для значения системного времени*/
```

Шаг 2. Получение системного времени:

```
t = time(NULL);
```

Шаг 3. Трансформация его в структуру tm:

```
mytime= localtime(&t);
```

Шаг 4. Использование полученных значений для печати на экран:

```
printf("Московское время %02d:%02d:%02d \n", mytime ->tm_hour, mytime ->tm_min, mytime ->tm_sec);
```

2. Добавьте вывод следующей информации:

№ вар	Информация
1	День недели (словами)
2	День, месяц и год через разделитель
3	День (цифра) и месяц (слово)
4	«сегодня N-й день k года»
5	«до воскресенья k дней и m часов»
6	«с первой пары прошло k часов m минут»
7	«до Нового года осталось m месяцев и k дней»
8	«до конца пары m минут и s секунд»
9	«до конца месяца d дней, до конца года m месяцев»
10	Месяц (римская цифра) и год (число)
11	Время года и две последние цифры года
12	«идет k день зимы/осени»

Для формирования строки используйте функцию sprintf().

3. Организуйте вывод на экран так, чтобы сообщения размещались точно в центре строки экрана (80 символов). Для подготовки строки используйте функцию sprintf(). Свободное пространство заполните символами ‘*’ и стилизуйте вывод в виде прямоугольного окна (10 символов).

13.4 Практические аспекты создания библиотек на языке Си

Библиотеки позволяют использовать разработанный ранее программный код в различных программах. Таким образом, программист может не разрабатывать часть кода для своей программы, а воспользоваться тем, что входит в состав библиотек. Обычно код библиотек отличается качеством, позволяет писать более ясный код, понятный большинству программистов.

В языке программирования Си код библиотек представляет собой функции, размещенные в файлах, которые скомпилированы в объектные файлы, а те, в свою очередь, объединены в библиотеки. В одной библиотеке

объединяются функции, решающие определенный тип задач.

У каждой библиотеки должен быть свой заголовочный файл, в котором должны быть описаны прототипы (объявления) всех функций, содержащихся в этой библиотеке.

При компиляции программы библиотеки подключаются компоновщиком (linker). Если программе требуются только стандартные библиотеки, то дополнительных параметров передавать не надо, во всех остальных случаях при компиляции программы требуется указать имя библиотеки и ее местоположение. Это может настраиваться средой разработки при установке соответствующих параметров или указываться в аргументах командной строки при компиляции.

Библиотеки бывают двух видов — статические и динамические. Код первых при компиляции полностью входит в состав исполняемого файла, что делает программу легко переносимой. Код динамических библиотек не входит в исполняемый файл, последний содержит лишь ссылку на библиотеку. Если динамическая библиотека будет удалена или перемещена в другое место, то программа работать не будет. С другой стороны, использование динамических библиотек позволяет сократить размер исполняемого файла. Также если в памяти находится две программы, использующие одну и ту же динамическую библиотеку, то последняя будет загружена в память лишь единожды.

Последовательность шагов создания статической библиотеки следующая.

1) Создать каталог проекта и разметить в нем все файлы кодов функций разрабатываемой библиотеки.

2) Собрать объявления всех функций в один заголовочный файл.

3) Выполнить настройку среды или задать параметры командной строки для построения библиотеки.

4) Построить проект.

При использовании разработанных библиотечных функций в новом проекте необходимо подключать заголовочный файл библиотеки: через относительный адрес:

```
#include "../library/mylib.h"
```

Две точки обозначают переход в каталог на уровень выше, т. е. родительский по отношению к project, после чего путь продолжается во вложенный в родительский каталог library. Можно указывать и абсолютный путь.

13.5 Контррольные задания для совместной разработки библиотеки

1. Преобразовать функции, разработанные в работе 9, так чтобы в качестве исходных данных функция paint() получала структуру с координатами окна для вывода символов, а для Calc() с его актуальными размерами.

Так структура описывающая размеры прямоугольника имеет вид:

```
struct rectangle {
```

```
    int w, h; // размеры прямоугольника
    };
```

А структура, предназначенная для его отображения:

```
struct paint_rectangle {
    int x0, y0; // левая верхняя точка прорисовки
    struct rectangle rect;
    char pen; // выводимый символ заполнения
    };
```

2. Подготовить заголовочный файл с именем mylib*.h (вместо звездочки указывается номер варианта) содержащий описание структур через typedef и прототипы всех разработанных функций.

3. Собрать проект и откомпилировать библиотеку.

4. Написать программу, которая предлагает пользователю:

А) выбор фигуры;

Б) вычисление ее площади или периметра;

В) рисование контура фигуры или заполнение ее произвольным символом;

Г) задание пользователем символа для рисования контура или заполнения контура фигуры.

ЛАБОРАТОРНАЯ РАБОТА №14
ЗАПИСЬ И ЧТЕНИЕ ФАЙЛОВ

Цель работы: Получение навыков организации файлового ввод и вывода
Программные средства: MICROSOFT VISUAL STUDIO

14.1 Теоретические сведения

В широком смысле файл – это набор данных, размещенный на внешнем носителе, и рассматриваемый в процессе обработки как единое целое.

Структура FILE содержит всю внутреннюю информацию о состоянии относительно соединения со связанным файлом, включая индикатор позиции файла и информацию буферизации. Прежде чем работать с файлом, его нужно открыть для доступа, т. е. создать и инициализировать область данных, которая содержит информацию о файле: имя, путь и т.д.

В языке Си это выполняет функция fopen(), которая связывает физический файл на носителе с логическим именем в программе. Логическое имя – это указатель на файл, т.е. на область памяти, где хранится информация о файле. Указатели на файлы необходимо объявлять:

```
FILE  *указатель на файл;
```

Функция открытия файла:

```
FILE *fopen(char *fname, char *mode);
```

fname– указатель на имя файла в файловой системе в обычной форме: полный путь к файлу или сокращенное имя (если файл в текущем каталоге).

mode–параметр, задающий тип операций, допустимых с файлом, записываемый следующими символьными константами (таблица 14.1).

Таблица 14.1

Пояснения символьных констант параметра *mode*

mode	Цель	Пояснение
“w”	для записи	если файла с заданным именем нет, то он будет создан, если же такой файл существует, то перед открытием прежняя информация уничтожается
“r”	для чтения	если файл уже существует, его длина обнуляется, иначе создается новый файл
“a”	для добавления в конец файла	если файл уже существует, начальное содержимое не изменяется, и вывод потока добавляется в конец файла, иначе, создается новый пустой файл
r+"	Изменение начальной позиции	открывает текстовый файл для чтения и записи
w+"		создает текстовый файл для чтения и записи
a+"		открывает текстовый файл для чтения и записи

Функция ***fopen*** возвращает указатель на поток, описываемойFILE и NULL, если при открытии файла произошла ошибка.

По умолчанию файл открывается в текстовом режиме (t), указание b в строке mode позволит открыть файл в двоичном режиме.

После работы доступ к файлу необходимо закрыть с помощью функции:

```
int fclose(указатель файла);
```

Работа с файлом обязательно включает следующие шаги:

1. Объявление указателя на используемый файл.	`FILE *f_my;`

2. Открытие файла и обработка возможных ошибок	`if(!(f_my = fopen(“myfile.txt”, “r+t” ) ) ) {`
	`  puts(“\n Ошибка открытия файла!”);`
	`  return;`

3. Работа с файлом	// последовательность операторов
4. Закрытие файла	`fclose(f_my);`

Прототипы большинства функций по обработке файлов описаны в библиотеках stdio.h и io.h.

Для работы с текстовыми файлами в консольном приложении удобнее

всего пользоваться функциями fprintf() и fscanf(), параметры и выполняемые действия аналогичны функциям printf() и scanf(), только первым параметром добавлен указатель файла, к которому применяется данная функция:

int fprintf (FILE *stream, const char *template, ...).
int fscanf (FILE *stream, const char *template, ...)

14.2 Примеры программ работы с файлами

Пример 1. Программа, считывает установленное пользователем количество данных из файла, выполняет над ними преобразование – вычисление среднего соседних чисел и записывает полученные значения в новый файл.

```
#include <stdio.h>
#include <locale.h>
int main()
{
    int u, o, p;
    double SrAref;
    FILE *f, *f1;
    setlocale(0, "Russian");
    if ((f = fopen("input.txt", "rt")) != 0) {
        printf("Открытие источника прошло
успешно\n");
    }
    if ((f1 = fopen("output.txt", "wt")) != 0) {
        printf("Открытие приемника прошло
успешно\n");
    }
    printf("Сколько бы чисел вы бы хотели
считать?\n");
    scanf("%d", &u);
    for (int i = 0; i < u; i++)
    {
        fscanf(f, "%d", &o);
        fscanf(f, "%d", &p);
        SrAref = double((o + p) / 2);
        fprintf(f1, "%.2lf ", SrAref);
    }
if (fclose(f1) == 0) printf("Закрытие приемника
прошло успешно\n");
else return -1;
if (fclose(f) == 0) printf("Закрытие источника прошло
успешно\n");
else return -1;
```

```
return 1;
}
```

Пример 2. Программа очистки кода программы на Си от комментариев

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

int main()
{
    FILE *in, *out;
    char finp[256];
    char fout[256];
    char c, p,a;
    int sig_comm;
    setlocale(LC_ALL, "RUS");
    printf("Приступайте к вводу имени файла\nИмейте ввиду если исходный файл находится в корневой папке программы,\nто вам понадобится лишь ввести имя файла и его расширение,\nиначе вам придется написать полный путь к исходному файлу: ");
    scanf("%s", finp);
    printf("Введите желаемое имя выходного файла:");
    scanf("%s", fout);
    if ((in = fopen(finp, "r")) == NULL)
    {
        printf("Файл %s невозможно открыть.\n", finp);
        return 0;
    }
    if ((out = fopen(fout, "w")) == NULL)
    {
        printf("Файл %s невозможно создать.\n", finp);
        return 0;
    }
printf("Файлы успешно открыты, запускается алгоритм отчищения от комментариев.\n");
p = fgetc(in);
sig_comm = 0;
while (!feof(in))
{
    c = fgetc(in);
    if ((sig_comm == 0) && (c == '"') && (p=='('))
        sig_comm = 3;
    if ((sig_comm == 0) && (c == '*') && (p == '/'))
```

```
        sig_comm = 2;
        if ((sig_comm == 0) && (c == '/') && (p == '/'))
            sig_comm = 1;
        if (sig_comm == 0) fputc(p, out);
        if ((sig_comm == 1) && (c == '\n'))
        {
            c = fgetc(in);
            sig_comm = 0;
            }
        if ((sig_comm == 2) && (c == '/') && (p == '*'))
            {
                c = fgetc(in);
                sig_comm = 0;
            }
        if (sig_comm == 3) fputc(p, out);
        if ((sig_comm == 3) && (c == ')') && (p=='"'))
            {
                fputc(c, out);
                c = fgetc(in);
                sig_comm = 0;
            }
            p = c;
        }
        printf("Код      отчищен      от      комментариев,
производится закрытие файлов.\n");
        fclose(in);
        fclose(out);
        printf("Файлы      успешно      закрыты.\nМожете
просмотреть полученный код..\n");
        system("pause");
    }
```

14.3 Практические задания по записи файла

1. Напишите программу, которая создает новый файл number.txt в текущей папке и записывает в него произвольную цифру и закрывает файл.

Запишите стандартную последовательность работы с файлом:

```
int main(void)
{
char fname[20] = "number.txt";// имяфайла
FILE *out; // файл чисел
puts("Создание файла");
if ((out = fopen(fname, "wt")) == NULL)
{
```

```
printf("Ошибка открытия файла для записи");
return 0;
}
/********начало работы с файлом**********/
fprintf(out,"%d",5);// ввод числа 5 в файл
/********конец работы с файлом **********/
fclose(out); // закрытьфайл
return 1;
}
```

А) Убедитесь, что она работает, открыв файл *number.txt* из текущейworkspace.

Б) Измените код в функции ***fopen***так, чтобы следующий раз число 12,56 добавлялось в файл.

В) Добейтесь, чтобы каждое новое число записывалось в файл с новой строки.

2. Измените программу так, чтобы она записывала в файл 50 случайных рациональных чисел от -1 до 1.

3. Организуйте форматированную запись в виде:

| порядковый номер | число |

4. Установите параметр mode при открытии файла как «a+» и запустите программу еще раз.

Проверьте два режима работы – с открытием нового файла и последовательной записью в ранее созданный файл.

14.4 Практические задания на чтение файла

1. Напишите программу, которая вычисляет среднее арифметическое целых чисел, содержащихся в файле.

Любая работа с потоками одинакова, поэтому открываем поток для чтения, последовательность шагов выполнения следующая:

Шаг 1. – считать число из открытого потока;

Шаг 2. – добавить его к накопленной сумме;

Шаг 3. – повторить шаг 1, если не достигнут конец файла;

Для каждого шага выберем языковую конструкцию или функцию.

Шаг 1. – функция fscanf(in,"%d", &a);

Потребует объявления переменной для приема числа из потока.

Шаг 2. – sum += a; n++;

Потребует объявления переменной для накопления суммы и счетчика чисел в потоке.

На шаге 3 потребуется проверка – достигнут ли конец файла, это можно сделать функцией feof(in), реализовав возврат на шаг 1 с помощью оператора цикла:

```
while (!feof(in))
```

```
{
fscanf(in,"%i", &a);
sum += a;
n++;
}
```

Начинаем собирать программу с включения заголовочных файлов, использованных функций <stdio.h> и создаем функцию main с объявлением необходимых переменных, инициализации структуры FILE, далее стандартная процедура открытия, обработки с циклом while и закрытия потока.

Запускаем программу на компиляцию и проверяем правильность выполнения.

! Обратите внимание, файл исходных данных должен быть в текущем каталоге.

2.Добавьте в программу поясняющий текст – сколько значений было введено, из какого файла и какой ответ получился.

3. Организуйте в программе вывод среднего геометрического значения (получается от перемножения всех значений и извлечения из этого произведения корня, показатель которого равен числу этих значений).

14.5 Контрольные задания

Написать программу, которая открывает указанный пользователем файл, содержащий выборку рациональных чисел, каждое их которых записано с новой строки.

Выполнить преобразование над ними согласно варианту.

Записать результат в указанный пользователем файл.

Варианты преобразований:

1.Увеличить все четные числа на 1.
2. Вычислить остаток от деления каждого значения на 2.
3. Заменить каждое второе число на 0.
4. Уменьшить все нечетные числа в 4 раза.
5. Все значения умножить на 0,3 и округлить до ближайшего целого.
6. Удалить все значения меньше 5.
7. Записать среднее арифметическое двух соседних чисел.
8. К каждому значению добавить его порядковый номер.

ЛАБОРАТОРНАЯ РАБОТА №15
ПОИСК В СТАТИЧЕСКОМ ОДНОМЕРНОМ МАССИВЕ

Цель работы: Закрепление навыков работы с одномерными массивами
Программные средства: MICROSOFT VISUAL STUDIO

15.1 Теоретические сведения

Массив – это структура данных, которая обладает следующими свойствами:

– все элементы массива имеют один и тот же тип;

– массив имеет одно имя для всех элементов;

– доступ к конкретному элементу массива осуществляется по индексу (индексам).

Различают массивы статические и динамические. Размер статических массивов фиксирован, а в динамических массивах число элементов может изменяться в ходе программы.

В языке Си обработка массивов осуществляется по элементам и, как правило, реализуется в цикле.

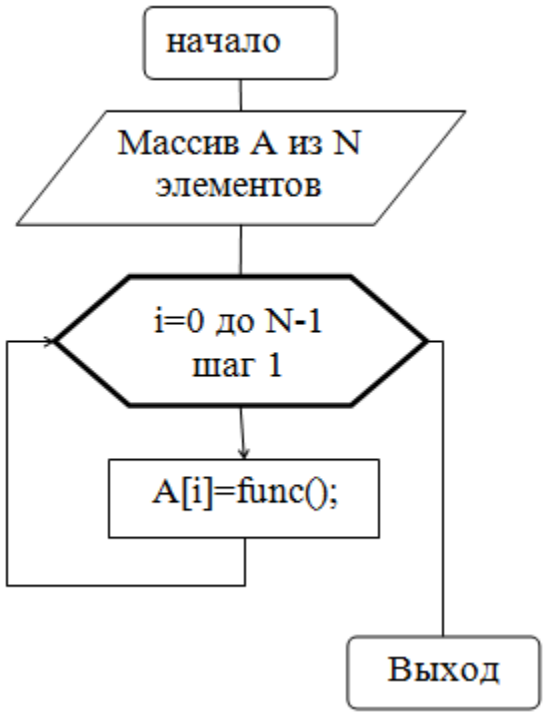


Рис.15.1. Алгоритм поиска по заданному шаблону

Распространенными методами работы с массивами являются:

– поиск;

– сортировка:

– вставка;

– удаление.

Для этих методов предложены различные по эффективности алгоритмы реализации. В большинстве случаев нет необходимости «изобретать велосипед», достаточно воспользоваться известными алгоритмами [5]. Выбранные алгоритмы могут различаться по скорости работы и используемым вычислительным ресурсам.

Самым простым вариантом поиска можно считать поиск в неупорядоченном одномерном массиве.

Если массив является неупорядоченным, то единственный алгоритм поиска, применимый в этом случае, является последовательный поиск, при котором последовательно перебираются все элементы массива. Если на каком-то шаге цикла обнаруживается, что массив закончился или обнаруживается искомый элемент, то цикл заканчивается.

Классическим алгоритмом поиска в отсортированном массиве является двоичный (бинарный) поиск (также известен как метод деления пополам и дихотомия).

Алгоритм линейного поиска состоит в следующем – поочередно просматриваются элементы, каждый из которых сравнивается с искомым, если

они тождественны, то поиск заканчивается, в лучшем случае предстоит произвести всего одно сравнение, а в худшем – n, где n— количество элементов массива. Аналогично ищется максимальный элемент.

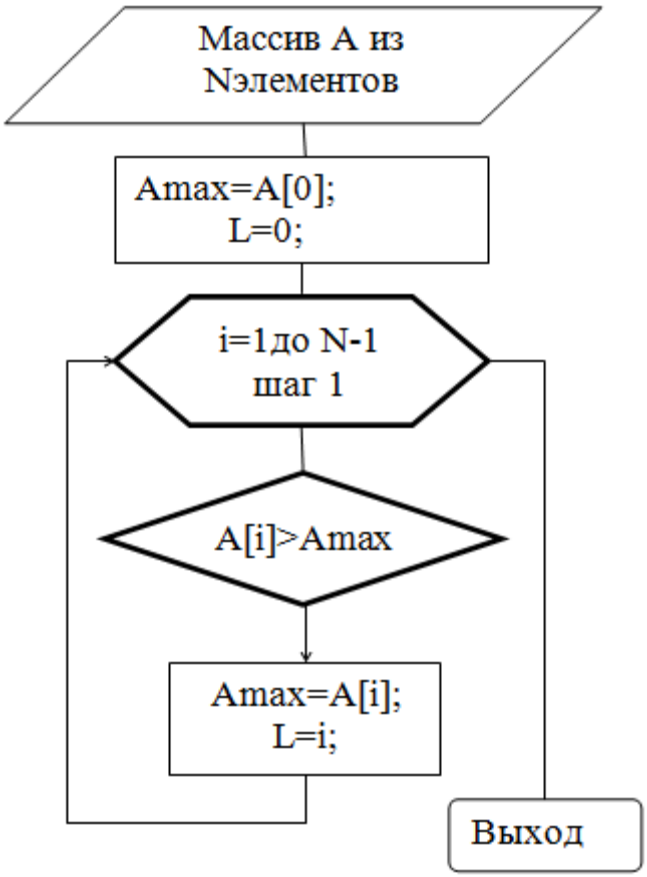


Рис. 15.2. Алгоритм линейного поиска

Описание алгоритма.

Исходные данные: A[N] – массив из N элементов.

Результаты:

Amax – максимальное значение

L– индекс искомого элемента.

Вначале предполагается, что искомый элемент нулевой и текущее значение Amax устанавливается равным ему. Затем каждый элемент массива последовательно сравнивается с Amax. Если текущий элемент больше Amax, то его значение сохраняется в качестве текущего результата. При достижении границы массива текущее значение Amax и соответствующего индекса L становиться результатом.

Следует отметить, что для поиска минимума достаточно поменять операцию отношения в условии на «меньше».

Бинарный поиск осуществляется путем неоднократного деления массива на две части таким образом, чтобы искомый элемент попадал в одну из этих частей. Поиск заканчивается при совпадении искомого элемента с элементом, который является границей между частями множества или при отсутствии искомого элемента.

Алгоритм бинарного поиска:

Шаг 1. Определить номер среднего элемента массива.

Шаг 2. Если значение среднего элемента массива равно искомому, то возвращаем значение, равное номеру искомого элемента, и алгоритм завершает работу.

Шаг 3. Если искомое значение больше значения среднего элемента, то возьмем в качестве массива все элементы справа от среднего, иначе возьмем в качестве массива все элементы слева от среднего (в зависимости от характера упорядоченности).

15.2 Приемы реализации линейного поиска

Поиск элемента неупорядоченного массива равного заданному значению – классическая задача поиска, общий алгоритм представлен на рис.15.2.

Линейный поиск предусматривает последовательный просмотр всех элементов массива в порядке их расположения, пока не найдется элемент равный заданному. Если достоверно неизвестно, что такой элемент имеется в массиве, то необходимо следить за тем, чтобы поиск не вышел за границы массива.

```
print_linear_seach(int* k, int N)
{
i=0;
while(k[i]!=v && i<N) i++;//перебор элементов
if (k[i]==v) printf("%d %d",v,i);
else printf("%d не найден",v);
}
```

Чтобы не проверять каждый раз в цикле условие достижения границы массива можно установить барьер (стоппер) для ограничения изменение индекса. В качестве барьера можно установить искомое значение, которое следует добавить в конец массива. Достоинством такого решения является уменьшение одной проверки в цикле. Выход из цикла, в котором теперь остается только одно условие поиска, может произойти либо на найденном элементе, либо на барьере.

```
print_linear_seach(int* k, int N)
{
intk[N+1],v,i;
...
k[N]=v; /* установка  стоппера    */
i=0;
while(k[i]!=v) i++;
if (i<N) printf ("%d %d",v,i);
else printf ("%d не найден",v);        }
```

Приведенные выше функции позволяют найти первое вхождение искомого значения в массиве, если необходимо найти последнее, то следует построить цикл так:

```
int j=-1;
for(i=0; i<N; i++)if(a[i]==K) j=i;
if (j==-1) puts("искомый элемент не найден");
else printf("Найден элемент %d/n", j);
```

Для поиска всех возможных вхождений потребуется организация хранения индексов. Непосредственный вывод в цикле найденных элементов можно организовать так:

```
print_linear_seach(int* k, int N)
```

```
{
inti;
// Задание значений K и чтение массива
for(i=0; i<N; i++)
if(a[i]==K) printf("Найден элемент %d/n", i);
if (i==N) puts("искомый элемент не найден");
return;
}
```

15.3 Примеры реализации алгоритмов поиска

Пример 1.Поиск расстояния между минимальным и максимальным элементом массива.

```
int dist (double *a, int n)
{
imin=0;// для хранения индекса миним. элемента
imax=0;//для хранения индекса макс. элемента
for(i=0; i<n; i++)
 {
    if(a[i]<a[imin] imin=i;
    if(a[i]>a[imax] imax=i;
    }
return abs(imin-imax);
}
```

Поиск не самого значения минимума или максимума, а индекса минимального (максимального) элемента эффективнее, так как уменьшает число присваиваний, при естественном хранении искомого элемента в исходном массиве.

Пример 2. Программа вычисления суммы элементов массива, начиная с первого отрицательного элемента.

```
#include <stdio.h>
#include <locale.h>
int main()
{
int Arr[100];
int i, n, Sum;
setlocale(0,"Russian");
printf("Введите количество элементов в массиве - ");
scanf("%d", &n);
if(n>100) {puts(“Много элементов”);return -1;}
/*чтение элементов*/
for (i = 0; i < n; i++)
{
    printf("№%d - ",i+1);
```

```
        scanf("%d", (Arr+i));
        }
    for (i = 0; i < n; i++)
        if (Arr[i] <0) break;
    if(i==n)    {    puts("Отрицательный    элемент    не
найден");return -1;}
    for (Sum=0, i < n; Sum+ = Arr[i++]);
    printf("Сумма - %d\n", Sum);
    system("pause");
    return 0;
    }
```

Пример3. Поиск минимального элемента упорядоченного массива равного заданному значению

```
    main()
    {
    inta[10]; // массив
    int min; // номер минимального элемента
    int i; // индекс массива
    printf("Введите элементы массива \n");
    for (i = 0; i < 10; i++) scanf("%i",&a[i]);
    min =0; // предположим, что первый эл-т минимальный
//   сравним оставшиеся эл-ты массива с минимальным
    for (i = 1; i < 10; i++)
    if (a[i] < a[min]) min = i;
    printf("Минимальный элемент массива: ");
    printf("a[%i]=%i ", min+1, a[min]);
    printf("\n Для завершения нажмите <Enter>");
    getchar();
    }
```

15.4 Практические задания

1. Объявите массив вещественных чисел из 20 элементов:
float a[20];
2. Заполните его случайными числами в диапазоне от -100 до 100.
3. Выполните заданную операцию над элементами неупорядоченного массива.

1	Суммирование элементов массива, расположенных между максимальным и минимальным элементами.
2	Замена максимального элемента на минимальный
3	Поиск номера первой и последней пары положительных элементов
4	Поиск номера максимального и минимального элементов массива, попавших в интервал (A,B)

5	Суммирование элементов массива, расположенных до последнего положительного элемента
6	Произведение элементов массива, расположенных между максимальным и минимальным элементами
7	Суммирование элементов массива, расположенных между первым и последним нулевыми элементами
8	Суммирование положительных элементов массива, расположенных до максимального элемента
9	Вывод порядковых номеров элементов, значения которых четные
10	Суммирование модулей элементов массива, расположенных после первого элемента равного нулю
11	Суммирование элементов массива, расположенных после минимального элемента
12	Вычисление среднего значения всех положительных элементов массива

4. Считайте данные в массив из входного файла «input.txt». Каждый элемент данных представляет вещественное число и записан с новой строки, количество элементов задано в файле первым целым значением.

5. Выполните над элементами считанного упорядоченного массива заданную операцию.

1	Определение номера максимального и номер минимального элементов массива, попавших в интервал (A,B)
2	Подсчет количества одинаковых элементов.
3	Вывод порядковых номеров элементов значения, которых больше заданного числа A.
4	Вычисление среднего значения всех положительных элементов массива больше заданного числа A
5	Суммирование элементов массива, расположенных до последнего положительного элемента
6	Вывод значений и порядковых номеров элементов массива, значения которых не равны нулю
7	Вычисление среднего арифметического значений элементов массива с p_1 по p_2 (значения p_1 и p_2 вводятся с клавиатуры; $p_2 > p_1$
8	Вычисление произведения всех отрицательных элементов массива большее (по модулю) заданного числа A
9	Вывод порядковых номеров элементов, значения которых четные
10	Суммирование элементов, кратных заданному пользователем значению K.
11	Вывод порядковых номеров элементов значения, которых меньше заданного числа A.
12	Суммирование элементов массива с $к_1$ по $к_2$ (значения $к_1$ и $к_2$ вводятся с клавиатуры, $к_2 > к_1$).

15.5 Контрольные задания

1. Составить программу нахождения элемента данных наиболее близкого к среднему значению.
2. Вычислить суммы отрицательных и положительных элементов массива.
3. Преобразовать пять последних элементов массива, умножив их на номер максимального элемента.
4. Найти среднее значение элементов массива, расположенных между максимальным и минимальным значением в массиве.
5. Найти два наименьших значения массива, имеющих четные индексы.
6. Вычислить разницу между максимальным отрицательным и минимальным положительным элементами массива.

ЛАБОРАТОРНАЯ РАБОТА №16
СТАТИЧЕСКИЙ МНОГОМЕРНЫЙ МАССИВ

Цель работы: Формирование навыков работы со статическим многомерным массивом

Программные средства: MICROSOFT VISUAL STUDIO

16.1 Теоретические сведения

Многомерным называется такой массив, который отличается двумя или более измерениями, причем доступ к каждому элементу такого массива осуществляется с помощью определенной комбинации двух или более индексов.

Каждый многомерный массив (матрица) характеризуется:

– именем;
– размерностью (числом измерений).

В памяти многомерный массив располагается в последовательных ячейках по слоям (строкам). Элементы с меньшими значениями индекса хранятся в более низких адресах памяти, т. е. самый правый индекс возрастает первым.

Двумерный массив можно представить в виде таблицы (матрицы), при этом первое измерение определяет строку, второе – столбец. Каждый элемент двумерного массива имеет два индекса. Первый задает номер строки, второй – номер столбца элемента в таблице.

Примером двумерного массива в математике является матрица:

$$a = \|a_{ij}\| = \begin{pmatrix} a_{00} \_ a_{01} \_ a_{02} \\ a_{10} \_ a_{11} \_ a_{12} \\ a_{20} \_ a_{21} \_ a_{22} \end{pmatrix}$$

В программе каждый массив должен быть объявлен. Объявление массивов производится в начале программы (функции) аналогично простым переменным:

```
<тип>   имя [размер1][размер2];
```

где тип – ключевое слово, определяющее тип элементов массива;

имя – идентификатор массива (формируется аналогично имени переменной);

размер 1 – целая константа, определяющая количество строк массива;

размер 2 – целая константа, определяющая количество столбцов массива.

Пример объявления двухмерного массива целочисленных элементов с именем d размером 16 строк и 20 столбцов:

```
int d[16][20];
```

При обращении к элементам двухмерного массива нужно указать имя массива, номер строки и номер столбца, на пересечении которых расположен этот элемент.

Пример обращения к элементу в 4-ой строке 3-го столбца:

```
mas[4][3]=0;   /* обнуление*/
```

При статической (определяемой на этапе компиляции) инициализации значения перечисляются в порядке указания размеров (индексов) в определении массива. Каждый уровень (индекс), кроме самого младшего, многомерного массива заключается в свою пару фигурных скобок. Значения самого младшего индекса указываются через запятую:

```
#define ROW 3
#define COL 5
int ary[ROW][COL] = {
{ 1, 2, 3, 4, 5 },
    { 2, 4, 6, 8, 10 },
    { 3, 6, 9, 12, 15 }
};
```

Самый левый размер массива может быть опущен, в этом случае компилятор сам определит этот размер, исходя из списка инициализации.

```
int a[][3] = {
  {  18,  21,   5 },
  {   6,   7,  11 },
  {  30,  52,  34 },
  {  24,   4,  67 }
              };
```

Массив a	Столбец 0	Столбец 1	Столбец 2
Строка 0	18	21	5
Строка 1	6	7	11
Строка 2	30	52	34
Строка 3	24	4	67

Ввод, вывод и обработка двухмерных и многомерных массивов также

осуществляются поэлементно.

16.2 Приемы работы со статическим многомерным массивом

Матрицы, у которых число строк равно числу столбцов называются квадратными – *a(n,n)* – квадратная матрица.

Квадратная матрица имеет главную и побочную диагонали. Например:

$a = \begin{pmatrix} 1\_2\_3 \\ 4\_5\_6 \\ 7\_8\_9 \end{pmatrix}$ на главной диагонали лежат элементы 1, 5, 9, на побочной – 3, 5, 7.

Для элемента a[i,j]:

Если i=j – элементы расположены на главной диагонали;

Если i>j – элементы расположены ниже главной диагонали;

Если i<j – элементы расположены выше главной диагонали;

Если i+j=n-1 – элементы расположены на побочной диагонали;

Если i+j<n-1 - элементы расположены над побочной диагональю;

Если i+j>n-1 – элементы расположены под побочной диагональю.

В двумерном массиве два индекса (один для строки и один для столбца), то есть для того, чтобы пройтись по всему массиву потребуется два вложенных цикла for.

Дан двумерный массив int A[n][n]. Необходимо элементам, находящимся на главной диагонали, проходящей из левого верхнего угла в правый нижний угол (то есть тем элементам A[i][j], для которых i==j) присвоить значение 1, элементам, находящимся выше главной диагонали – значение 0, элементам, находящимся ниже главной диагонали – значение 2. То есть получить такой массив (пример для n==4):

1 0 0 0
2 1 0 0
2 2 1 0
2 2 2 1

Элементы, которые лежат выше главной диагонали – это элементы A[i][j], для которых i<j, а для элементов ниже главной диагонали i>j. Таким образом, можно сравнивать значения i и j и по ним определять значение A[i][j]. Получается следующий алгоритм:

```
for (i = 0; i < n; ++i)
{
    for (j = 0; j < n; ++j)
    {
    if (i < j)
    {
            A[i][j] = 0;
    }
```

```
        else if (i > j)
            {
            A[i][j] = 2;
        }
        else
        {
                A[i][j] = 1;
            }
        }
    }
```

Данный алгоритм плох, поскольку выполняет одну или две инструкции if для обработки каждого элемента. Если усложнить алгоритм, то можно обойтись вообще без условных инструкций.

Сначала заполняется главная диагональ, для чего необходим один цикл:

```
    for (i = 0; i < n; ++i)
    {
        A[i][i] = 1;
    }
```

Затем заполняются значением 0 все элементы выше главной диагонали, для чего необходимо в каждой из строк с номером i присвоить значение элементам **A[i][j] для j=i+1, ..., n-1**. Здесь надо применить вложенные циклы:

```
    for (i = 0; i < n; ++i)
    {
        for (j = i + 1; j < n; ++j)
        {
            A[i][j] = 0;
        }
    }
```

Аналогично присваиваивается значение 2 элементам **A[i][j]** для **j=0, ..., i-1:**

```
    for (i = 0; i < n; ++i)
    {
        for (j = 0; j < i; ++j)
        {
            A[i][j] = 2;
        }
    }
```

Можно также внешние циклы объединить в один и получить еще одно, более компактное решение:

```
    for (i = 0; i < n; ++i)
    {   // Заполняем строку с номером i
        for (j = 0; j < i; ++j)
```

```
        {
        A[i][j] = 2;   // Сначала пишем 2 ниже диагонали
        }
        A[i][j] = 1; // После завершения предыдущего
цикла i==j, пишем 1
        for (++j; j < n; ++j)    // Цикл начинаем с
увеличения j на 1
        {
            A[i][j] = 0;  // Записываем 0 выше диагонали
        }}
```

16.3 Примеры работы со статическим многомерным массивом

Пример 1. Ввод и вывод элементов двухмерного массива

```
int a [50][50];
int i, j, n, m;
printf("Введите количество строк: ");
scanf("%d", &n);
printf("Введите количество столбцов: ");
scanf("%d", &m);
if(!(n<50) || !(m<50)) return;
// Ввод элементов массива
for(i=0; i<n; i++)  // цикл по строкам
{
for(j=0; j<m; j++)  {// цикл по столбцам
printf(" a[%d][%d] = ",i,j);
scanf("\n%d", &a[i][j]);
}
}
    // Вывод элементов массива
  for(int i=0; i<n; i++)  // цикл по строкам
  {
    for(int j=0; j<m; j++)  // цикл по столбцам
{
printf("a[%d][%d] = %d", i,j,a[i][j]); //
    }
printf("\n");
    }
```

Пример 2. Заполнение элементов двухмерного массива случайными числами и вывод на экран

```
  int a [50][50];
  int i, j, n, m;
  srand(time(NULL)|clock());
  printf("Введите количество строк: ");
```

```
  scanf("%d", &n);
  printf("Введите количество столбцов: ");
  scanf("%d", &m);
  // Ввод элементов массива
  for(i=0; i<n; i++)  // цикл по строкам
   {
    for(j=0; j<m; j++)  {// цикл по столбцам
  a[i][j]=-30+rand()%101;
  } }
// Вывод элементов массива
  for(int i=0; i<n; i++)  // цикл по строкам
  {
    for(int j=0; j<m; j++)  // цикл по столбцам
      {
       printf("a[%d][%d] = %d", i,j,a[i][j]); //
    }
    printf("\n");
    }
```

Пример 3. Функция заполнение элементов двухмерного массива в стиле «шахматная доска»

```
void printchessboard(int size)
/* size – размер доски*/
{
int chessboard[80][80];
int x, y;
for(x = 0;x<size;x++){
    for(y = 0;y<size;y++){
        chessboard[x][y] = abs((x-y)%2);
        printf("%c",chessboard[x][y]?'#':'O');
        }
    printf("\n");
    }
}
```

Пример 4. Функция поиска минимального элемента в трехмерном массиве.

Трехмерный массив int arr[L][M][N] состоит из L двухмерных массивов размерностью M на N, располагающихся по порядку:

Первый массив M на N	Второй массив M на N	Третий массив M на N		
arr	arr+(M*N)	arr+2*(M*N)		
Начальный адрес				

Каждый двухмерный массив M на N располагается по M строкам, содержащим одномерный массив из N элементов. Поэтому обращение к k-

элементу j-й строки i-го массива M на N вычисляется как:

```
arr+i*(M*N)+j*N+k
```

В функцию передается адрес начала массива и размеры массива. Возвращается указатель на структуру, содержащую индекс минимального элемента.

```
struct INDEX {
             int i, j, k;} index
struct INDEX * find(int *prt, int L, int M, int N)
{
int min;
int i, j, k, ind;
min=*ptr;
index.i=index.j=index.k=0;
for(i=0;i<L; i++)
    for(j=0;j<M; j++)
        for(k=0;k<N; k++)
        {
        ind=i*(M*N)+j*N+k;
        if(min>*(ptr+ind))
        {
        min=*(ptr=ind);
         index.i=i;
         index.j=j;
         index.k=k;
        }
        }
return &index;
}
```

16.4 Практические задания

1. Составьте программу, которая создает двумерный массив (исходный массив инициализировать как с консоли, так и случайным образом), каждый элемент которого формируется по следующему правилу:

1) равен сумме его индексов;

2) числу элементов, расположенных в массиве после него;

3) произведению порядкового номера элемента в строке на число предшествующих элементов в столбце;

4) латинской букве алфавита, порядковый номер которой соответствует порядковому номеру элемента в строке;

При написании программы учесть, что размерность массива (количество строк и столбцов) задается пользователем.

2. Вывести на экран результат в виде таблицы, столбцы и строки которой

обозначены соответствующими индексами массива.

3. Составьте программу, которая формирует двумерный массив целых чисел от 1 до 100 (размерностью 7 на 7) и транспонирует его. Организуйте вывод на экран исходного и транспонированного массива.

4. Найдите максимальный элемент в созданном двумерном массиве и заменить его значение суммой его индексов.

5. Полученный массив запишите в файл.

16.5 Контрольные задания

1. Из файла считать вещественные числа, сформировать двухмерный массив Arr[][10], для каждого одномерного массива вычислить сумму всех отрицательных элементов и произведение всех положительных.

2. Из файла считать вещественные числа, сформировать двухмерный массив Arr[][9], в каждом одномерном массиве из 9 элементов заменить значение последнего элемента на минимальное значение, найденное в соответствующем одномерном массиве.

3. Из файла считать целые числа, сформировать двухмерный массив Arr[][5],для каждого столбца которого вычислить сумму всех нечетных элементов и среднее значение.

4. Из файла считать вещественные числа, сформировать два двухмерных массива Arr[][7] так, чтобы первый содержал целую часть числа, а второй – дробную.

ЛАБОРАТОРНАЯ РАБОТА № 17
ДИНАМИЧЕСКИЕ МАССИВЫ

Цель работы: Получение практических навыков работы с динамическими массивами и пользовательскими функциями

Программные средства: MICROSOFT VISUAL STUDIO

17.1 Теоретические сведения

Динамический массив – это массив, размер которого заранее не фиксирован и может меняться во время исполнения программы. Работа с динамическими массивами строится на использовании указателей и специальные функции. Динамические массивы обеспечивать гибкость программы.

Последовательность использования динамических массивов в программе следующая:

1) объявление массива;

2) выделение памяти;

3) обработка элементов массива;

4) освобождение памяти.

Под объявлением одномерного динамического массива понимают объявление указателя на переменную заданного типа для того, чтобы данную переменную можно было использовать как динамический массив.

```
<тип> * имя_массива;
```

где Имя_Массива – идентификатор массива, то есть имя указателя для выделяемого блока памяти.

тип – тип элементов объявляемого динамического массива.

Выделение памяти осуществляется с помощью библиотечных функцийдинамического распределения памяти malloc(size)и calloc(num, size_t), которая служит для выделения динамической памяти.

```
имя_массива = (тип *) malloc(N*sizeof(тип));
имя_массива = (тип *) calloc(N, sizeof(тип));
```

Эти функции возвращают указатель на первый байт выделенной области памяти. Если для удовлетворения запроса нет достаточного объема памяти, возвращается нулевой указатель. Перед попыткой использовать распределенную память важно проверить, что возвращаемое значение не равно нулю.

Обработка элементов динамического массива происходит аналогично статическим, как через индексирование, так и по указателю. К i-му элементу динамического массива p можно обратиться одним из двух способов:

```
*(имя_массива+i) или имя_массива[i].
```

Изменением размеров массива после его создания можно управлять, используя функцию void* realloc(void* ptr, size_tsize), которая изменяет размер блока ранее выделенной памяти, адресуемой указателем ptr в соответствии с заданным размером size. Значение параметра size может быть больше или меньше, чем перераспределяемая область. Функция realloc() возвращает указатель на блок памяти, поскольку не исключена необходимость перемещения этого блока (например, при увеличении размера блока памяти). В этом случае содержимое старого блока (до size байтов) копируется в новый блок, поэтому функция возвращает указатель на перераспределенную память.

Когда использование массива завершено, необходимо освободить с помощью функции free().

```
free(имя_массива);
```

Прототипы функций динамического распределения памяти находятся в заголовочном файле ***stdlib.h.***

17.2 Примеры работы с динамическими массивами

Пример 1. Функция для формирования одномерного динамического массива

```
int * make_mas(int n)
(
```

```
int *mas;
mas=(int*)malloc(n*sizeof(int));
for(int i=0;i<n;i++)
mas[i]=random(10);
return mas;
}
```

Пример 2. Программа перераспределения памяти.

Эта программа сначала выделяет блок памяти для размещения строки "Пусть всегда будет мир", а затем использует realloc() для увеличения размера блока, чтобы разместить в конце восклицательный знак.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
int main(void)
{
char *p;
int size;
size= sizeof("Пусть всегда будет мир");
p = malloc(size);
if(!p) {
printf("Ошибка при распределении памяти\n");
    exit(-1);
  }
  strcpy(p, "Пусть всегда будет мир");
  p = realloc(p, size+1);
if(!p) {
    printf("Ошибка при распределении памяти\n");
exit(-1);
  }
strcat(p, "!");
printf(p);
free(p);
return 0;
}
```

Пример 3 Программа, вычисляющая сумму элементов массива

```
float summa (float* mas, int n); //прототпи функции
void input_arr_random (float *mas, int n);// прототип функции
float main(float)
{
    setlocale(0,"Russian");
    float* mas; //указатель на массив
    int n;
printf("введите размерность массива \n");
```

```
    scanf_s("%d",&n);
mas=(float *)malloc(n*sizeof(float));//выделение памяти
    input_arr_random (mas,n);// вызов функции
    printf("\n %f", summa (mas,n));
    free(mas); //высвобождение памяти
    system("pause");
    return 0;}

функция ввода элементов массива
void input_arr_random (float *mas, int n)
{ printf("Массив заполнится числами в заданном диапазоне
от -10 до 20: \n");
    srand(time(NULL)|clock());
    for (int i=0;i<n;i++)
            mas[i]=-10+rand()%31;
        printf("получен массив\n");
            for (int i=0;i<n;i++)
        printf("\n a[%d]= %f", i, mas[i]);
    return;
}
функция вычисляющая сумму элементов массива
float summa (float* mas, int n)
{
float s=0;
int i;
for(s=0,i=0;i<n;i++)
s+=*(mas+i);
     return s;}
```

17.3 Особенности работы с двумерными динамическими массивами

Недостатком языка Си является невозможность по имени массива определять его размерность и размеры по каждой из размерностей. Это затрудняет передачу массивов в качестве параметров функций.

В функцию можно передавать двумерный массив в качестве параметра, если размер этого массива фиксирован и объявлен в описании функции. То есть если заранее известен размер массива, то можно определить функцию, получающую в качестве параметра двумерный массив такого размера:

```
    void f (int A[10][10])
    {……
    }

    int main()
```

```
{
int B[10][10];
    f(B);
}
```

Проблема заключается в том, что в этом случае нельзя использовать массивы произвольного размера. Чтобы использовать массивы произвольного размера, необходимо использовать указатели.

Так, объявленный одномерный массив int A[n] может быть доступен по указателю на переменную типа int: int * A.

Тогда двумерный массив – это массив, каждый из элементов которого является одномерным массивом, т.е. указателем на какой-то адрес целого числа в памяти. Или двумерный массив – это массив элементов типа int * или же это указатель на переменную типа int *, т. е. это переменная типа int **.

Итак, двойной указатель можно объявить так:

```
int ** A;
```

Память под многомерные динамические массивы выделяется по строкам, начиная с первого индекса. Это делается для того, чтобы обеспечить применение операции индексирования ‘[]’ столько раз, какова размерность многомерного массива. В этом случае тип динамического массива объявляется как указатель, который содержит оператор обращения по адресу ‘*’ столько раз, какова размерность массива. Например, указатели на двухмерный и трехмерный целочисленные массивы могут быть объявлены следующим образом:

```
int **a; /* указатель на двумерный массив */
int ***b; /* указатель на трехмерный массив */
```

Пусть требуется разместить в динамической памяти матрицу, содержащую n строк и m столбцов. Двумерная матрица будет располагаться в оперативной памяти в форме ленты, состоящей из элементов строк. При этом индекс любого элемента двумерной матрицы можно получить по формуле:

```
index = i*m+j;
```

где i - номер текущей строки; j - номер текущего столбца.

Объем памяти, требуемый для размещения двумерного массива, определится как **n·m·(размер элемента)**

Однако, поскольку при таком объявлении компилятору явно не указывается количество элементов в строке и столбце двумерного массива, традиционное обращение к элементу путем указания индекса строки и индекса столбца a[i][j] является некорректным.

Правильное обращение к элементу с использованием указателя будет строится следующим образом:

```
*(prt+i*m+j) или *(*(prt+i)+j),
```

где prt - указатель на массив, m - количество столбцов, i - индекс строки, j - индекс столбца.

Динамические двумерные массивы представляются как массив

указателей, например:

```
int **A;
```

Первый массив A[0]	Второй массив A[1]	Третий массив A[2]		
*(A)	*(A+1)	*(A+2)		
Массив указателей **A				

Выделение памяти необходимо осуществлять с использованием циклов, сначала выделяется память для указателей на одномерный массив, а потом – элементов каждого одномерного массива.

```
int **A = (int **)malloc(N*sizeof(int *));
for(int i = 0; i < N; i++) {
    A[i] = (int *)malloc(M*sizeof(int));
}
```

Важно правильно освободить выделенную память. Сначала освобождение памяти занимаемой строками, затем освобождение памяти, занимаемой массивами указателей.

```
for(int i = 0; i < N; i++) {
free(A[i]);
}
free(A);
```

Пример. Выделение и высвобождение памяти для многомерного массива

```
int main() {
   // указатель на массив
  int  **a ,i, j, n, m;
  setlocale(0,"Russian");
  printf("Введите количество строк: ");
  scanf("%d", &n);
  printf("Введите количество столбцов: ");
  scanf("%d", &m);
  // Выделение памяти под массив указателей
 a= (int**) malloc(n*sizeof(int*));
  //  for(i=0; i<n; i++)  // цикл по строкам
    {
        a[i] = (int*)malloc(m*sizeof(int));
    for(j=0; j<m; j++)  // цикл по столбцам
     {
     *(*(a+i)+j) = -10+rand()%21;
     }
   }
// Вывод элементов массива
  for(int i=0; i<n; i++)  // цикл по строкам
  {
    for(int j=0; j<m; j++)  // цикл по столбцам
```

```
      {
       printf("%5d ",*(*(a+i)+j) ); // 5 знакомест под
элемент массива
      }
    printf("\n");
  }
    for(int i=0; i<n; i++) // высвобождение памяти
занимаемой строками
    free(a[i]);
  free(a);
  system("pause");
  return 0;
}
```

17.4 Практические задания

1. Объявите указатель на одномерный динамический массив (выбор типа зависит от заданного преобразования).

2. Организуйте ввод размера массива с клавиатуры.

2.Сформируйте массив с помощью функций malloc (calloc).

3.Заполните массив случайными числами.

4.Выполнить над элементами массива указанное преобразование и сформируйте новый массив(ы)-результат(ы).

1.	Удаление элемента с заданным номером, добавление элемента с заданным номером.
2.	Удаление из него элемента с заданным ключом, добавление элемента с заданным ключом.
3.	Удалить из него К элементов, начиная с заданного номера, добавление одного элемента с заданным ключом.
4.	Удаление элемента с заданным номером, добавление К элементов, начиная с заданного номера.
5.	Удаление строки с заданным номером.
6.	Добавление строки с заданным номером
7.	Удаление элементов значение которых превышает К.
8.	Удаление максимального и минимального элемента.
9.	Добавление в начало массива значения, равного сумме всех элементов
10.	Добавление в конец массива двух значений, равных среднему значению элементов массива
11.	Удаление все элементов, значения которых четные.
12.	Удаление всех строк, в которых встречаются цифры.

5.Напечатать массив(ы)-результат(ы).

6.Удалить динамические массивы с помощью функции free().

17.5 Контрольные задания

Задача 1 Дано число n. Создайте массив размером n×n и заполните его по следующему правилу:

Числа на диагонали, идущей из правого верхнего в левый нижний угол равны 1.

Числа, стоящие выше этой диагонали, равны 0.

Числа, стоящие ниже этой диагонали, равны 2.

Полученный массив выведите на экран. Числа в строке разделяйте одним пробелом.

Задача 2 Дан двумерный массив. Поменяйте в нем первую и последнюю строку. Полученный массив выведите на экран.

Программа получает на вход два числа: количество строк n в массиве и количество столбцов m. Далее идет n строк, каждая из которых содержит m чисел - элементы массива.

Выведите массив на экран разделяя числа в строке одним пробелом.

Задача 3 Дан двумерный массив и два числа: i и j. Поменяйте в массиве столбцы с номерами i и j и выведите результат.

Задача 4 Дано число n и массив размером n×n. Проверьте, является ли этот массив симметричным относительно главной диагонали. Выведите слово “YES”, если массив симметричный, и слово “NO” в противном случае.

Задача 5 Дан двумерный массив размером n×m. Симметричный ему относительно главной диагонали массив называется транспонированным к данному. Он имеет размеры m×n: строки исходного массива становятся столбцами транспонированного, столбцы исходного массива становятся строками транспонированного. Для данного массива постройте транспонированный массив и выведите его на экран.

Задача 6 Дан двумерный массив размером n×n. Транспонируйте его и результат запишите в этот же массив. Вспомогательный массив использовать нельзя.

Задача 7 Дан квадратный массив. Поменяйте местами элементы, стоящие на главной и побочной диагонали, при этом каждый элемент должен остаться в том же столбце (то есть в каждом столбце нужно поменять местами элемент на главной диагонали и на побочной диагонали).

ЛАБОРАТОРНАЯ РАБОТА № 18
ПЕРЕДАЧА ПАРАМЕТРОВ В ФУНКЦИЮ

Цель работы: Закрепление навыков передачи параметров в функцию по значению и по адресу

Программные средства: MICROSOFT VISUAL STUDIO

18.1 Теоретические сведения

Обмен информацией между вызываемой и вызывающей функциями осуществляется с помощью механизма передачи параметров. Передача параметров в функцию может осуществляться:

– по значению;

– по адресу.

При передаче данных по значению функция работает с копиями фактических параметров, и доступа к исходным значениям аргументов у нее нет. При передаче по адресу в функцию передается не переменная, а ее адрес, и, следовательно, функция имеет доступ к ячейкам памяти, в которых хранятся аргументов.

по значению	по адресу
void change (int x, int y) **{** **int k=x;** **x=y;** **y=k;** **}**	**void change (int *x, int *y)** **{** **int k=*x;** ***x=*y;** ***y=k;** **}**
при вызове передаются значения	При вызове передаются адреса:
change (a, b); **//a=5, b=7**	**change (&a, &b);** **//&a – вычисление адреса переменной a**

Таким образом, данные, переданные по значению, функция изменить не может, в отличие от данных, переданных по адресу.

Наибольший интерес представляет использование указателей как параметров и возвращаемых функцией значений, это обеспечивает возможность передачи массивов, структур и даже функций.

Передача массива в функцию осуществляется только через указатель (по адресу):

```
func (* arr);
```

Если в функцию передается указатель на одномерный массив, то в самой функции его можно объявить одним из трех вариантов:

как указатель,	как массив определенного размера	как массив без определенного размера.
void func(int *x) **{** **/* ... */** **}**	**void func(int x[7])** **{** **/* ... */** **}**	**void func(int x[])** **{** **/* ... */** **}**

При передаче функции члена структуры передается только одно значение, поэтому оно может быть передано как по значению, так и по адресу.

Вся структура целиком может быть передана также и по значению, и по указателю. При передаче структуры тип аргумента должен совпадать с типом параметра, обязательно должны совпадать даже имена их типов. Поэтому **объявление типа структуры должно быть глобальным**, чтобы структурный тип можно было использовать во всех функциях программы.

В функцию в языке Си можно передавать указатель на другую функцию. Указателем на функцию является адрес, по которому расположен код функции. Именно этот адрес используется при вызове функции. Так как указатель хранит адрес функции, то она может быть вызвана с помощью этого указателя. Он позволяет также передавать ее другим функциям в качестве аргумента и получать в качестве возвращаемых функциями значений. Это очень мощное средство!

Указатель на функцию объявляется как:

```
<тип> (*pfunc)(<тип>, <тип>, ..);
```

где pfunc – имя указателя, за которым располагается список аргументов (<тип>, <тип>, ..) .

Описание указателя на функцию должно соответствовать описанию самой функции: число и типы аргументов указателя должны совпадать с числом и типами аргументов функции.

```
Объявление указателя на функцию:
double (*pmyfunnc) (double, int);
Инициализация указателя именем функции:
pmyfunnc=pow;
Обязательно должен существовать такой прототип:
pow(double, int);
```

В ряде случае изменения массива или структуры функцией допустить нельзя, или наоборот необходимо иметь постоянный адрес. В связи с этим различают:

указатель на константный объект	const char * ptr	Объект не может быть изменен
константный указатель на объект	char * const ptr	Адрес не может быть изменен
константный указатель на константный объект	const char * const ptr	Ни адрес, ин объект не может быть изменен

18.2 Передача параметров функции main

Функция main, с которой начинается выполнение программы, может быть определена с параметрами, которые передаются из системного окружения.

Заголовок функции main имеет вид:

```
int main (int argc, char *argv[], char *argp[])
```

Все данные системного окружения представляются в виде строк символов. Для передачи этих строк в функцию main используются два

параметра:

argc имеет тип int и служит для передачи числа передаваемых строк,

argv – это массив указателей на строки, каждая из которых содержит одно слово из командной строки.

Значение параметра argc формируется из анализа командной строки и равно количеству слов в командной строке, включая и имя вызываемой программы (под словом понимается любой текст не содержащий символа пробел).

Если слово командной строки должно содержать символ пробел, то при записи его в командную строку оно должно быть заключено в кавычки.

Функция main может иметь и третий параметр, который принято называть argp, и который служит для передачи в функцию main параметров операционной системы (среды) в которой выполняется программа.

Примеркомандной строки:

A:\>cprog	working	“C program”	1		
argv [0]	argv [1]	argv [2]	argv [3]	NULL	

Равносильно вызову:

```
main (4, argv[4]);
```

Операционная система поддерживает передачу значений для параметров argc, argv, argp, а на пользователе лежит ответственность за передачу и использование фактических аргументов функции main.

Пример печати, полученных параметров:

```
int main ( intargc, char *argv[], char *argp[])
{  int i=0;

    printf ("\n Имя программы %s", argv[0]);
    for    (i=1; i>=argc; i++)
    printf ("\n аргумент %d равен %s", argv[i]);
printf ("\n   Параметры операционной системы:");
while (*argp)
       {  printf ("\n %s",*argp);
argp++;
          }
    return 0;
     }
```

18.3 Примеры передачи структур данных по адресу

Пример 1. Передача одномерного массива

Обратите внимание, передача массива через указатель требует одновременной передачи размера.

```
#include <stdio.h>
void sa_arr(float *a, int n){
```

```
        double summ=0;
        for (int i=0; i<n; i++)
              summ+=a[i];
        printf("Mean= %.2f\n", summ/n);
}
int main()
{
float *arr
arr = malloc (sizeof(float)* N);
 for (int i=0; i<N; i++)
            scanf("%f", &arr[i]);
sa_arr(arr,N);
free(arr);
return 0;
}
```

Пример 2. Передача строк

В данном случае в функцию передается только указатель на строку. Передача размера массива не нужна, так как есть однозначное указание на конец массива – нулевой символ в конце строки. По достижению которого, внутренний цикл заканчивается string[t]==’/0’.

```
#include <stdio.h>
#include <ctype.h>
void print_upper(char *string);
int main(void) /* вывод строки в верхнем регистре */
{
char s[80];
gets (s);
print_upper(s) ;
printf("%s", s);

return 0;
}
/* функция изменения символов строки на прописные */
void print_upper(char *string)
{
int t;
for(t=0;     string[t];     ++t)     string[t]     =
toupper(string[t]);
}
```

Пример 3. Передача многомерного массива

Выделение динамической памяти с помощью функций языка Си и передачи многомерного массива в функцию в качестве параметра

```
void arrprint (int*a, int n, int m);// прототип функции
```

```
int main() {
  int *a;  // указатель на массив
  int i, j, n, m;
  setlocale(0,"Russian");
  printf("Введите количество строк: ");
  scanf("%d", &n);
  printf("Введите количество столбцов: ");
  scanf("%d", &m);
  a = (int*) malloc(n*m*sizeof(int)); // Выделение
памяти
  // Ввод элементов массива
  for(i=0; i<n; i++)  // цикл по строкам
   {
    for(j=0; j<m; j++)  // цикл по столбцам
     {
      printf("a[%d][%d] = ", i, j);
      scanf("%d", (a+i*m+j));
     }
   }
  arrprint(a, n, m); //Вызов функции, печатающую
элементы массива
  free(a);// Высвобождение памяти
system("pause"); return 0;}
 void arrprint (int*a, int n, int m) {
     // Вывод элементов массива
  for(int i=0; i<n; i++)  // цикл по строкам
  {
    for(int j=0; j<m; j++)  // цикл по столбцам
      {
 printf("%5d  ", *(a+i*m+j)); // 5 знакомест под
элемент массива
      }
    printf("\n");
  }
 }
```

Для динамического выделения памяти под двумерный массив - с использованием массива указателей. Для этого необходимо:

- выделить блок оперативной памяти под массив указателей;
- выделить блоки оперативной памяти под одномерные массивы, представляющие собой строки искомой матрицы;
- записать адреса строк в массив указателей.

При таком способе выделения памяти компилятору явно указано количество строк и количество столбцов в массиве.

```
 void arrprint (int**a, int n, int m) {
```

```
      // Вывод элементов массива
    for(int i=0; i<n; i++)  // цикл по строкам
    {
      for(int j=0; j<m; j++)  // цикл по столбцам
        {
          printf("%5d ", *(*(a+i)+j)); // 5 знакомест
      под элемент массива
        }
      printf("\n");
    }
  for(int i = 0; i < n; i++)
  free(a[i]);   // цикл освобождение памяти занимаемой
строками
  }

  int main() {
    int **a;  // память под массив указателей на строку
    int i, j, n, m;
    printf("Введите количество строк: ");
    scanf("%d", &n);
    printf("Введите количество столбцов: ");
    scanf("%d", &m);
   // Выделение памяти под указатели на строки
    a = (int**)malloc(n*sizeof(int*));
    // Ввод элементов массива
    for(i=0; i<n; i++)  // цикл по строкам
    {
  // Выделение памяти под хранение строк
      a[i] = (int*)malloc(m*sizeof(int));
      for(j=0; j<m; j++)  // цикл по столбцам
  {
    *(*(a+i)+j) = -10+rand()%21;//
        }
     }
    arrprint(a, n, m);
  free(a); // освобождение памяти массива указателей
  system("pause");
   return 0;}
```

Пример 4. Передача структуры по значению и адресу

```
#include <stdio.h>
/* объявление типа структуры */
struct data_type {
int y, d;
```

```
char *m;
};
void print_data (struct data_type parm);
int main(void)
{
char month[][20]={"january","february","march", ...};
struct data_type arg = {2015, month[3],7};
print_data (arg);
return 0;
}
void print_data(struct data_type parm) {
printf("%04d/%s/%02d", parm.y, parm.m, parm.d);
}
```

Для передачи структуры по указателю следует описать указатель на структурную переменную в качестве аргумента:

```
void print_data (struct data_type *pData);
```

Поскольку структура адресуется указателем, доступ к полям структуры осуществляется оператором->

```
void print_data(struct data_type *prt) {
printf("%04d/%s/%02d", prt->y, prt->m, prt->d);
}
```

Соответственно в функции main() должна быть организована передача адреса инициализированной структуры в функцию:

```
print_data (&arg);
```

Пример 4. Использование указателя на функцию

```
// Объявление типа указателя на функцию
typedef double (*TFun)(double);
// Объявление прототипов функций
double fun1(double);
double fun2(double);
void Out_Rez (TFun,double,double,double);

void main()
{
double a, b, h;
puts("Input a,b,h");
scanf("%lf%lf%lf", &a, &b, &h);
puts("\n\t Function - 2*exp(x)");
Out_Rez (fun1,a,b,h);
puts("\n\t Function - sin(x)*sin(x)");
Out_Rez (fun2,a,b,h);
getch();
}
/*----Описание пользовательских функций -----*/
```

```
double fun1( double r)
{
        return 2*exp(r*r);
}
double fun2(double r)
{
returnpow(sin(r), 2);
}
/*----Функций вывода результата-------------*/
void Out_Rez (TFun f, double xn,double xk,double h)
{
    for(double x=xn; x<=xk; x+=h)
    printf("|x = %5.2lf| y = %8.4lf|\n",x,f(x));
}
```

18.4 Практические задания

1. Напишите функцию, которая меняет значения переменных a, b, c местами так, чтобы a>b>c. Для этого в основной программе объявите:

int a, b, c;

Организуйте ввод значений пользователем.

Опишите дополнительную функцию change (&a, &b); для замены значений переменных a>b.

Вызовите ее в основной программе необходимое число раз с соответствующими параметрами.

Продемонстрируйте результат работы программы с использованием последовательного отображение переменных в основной программе.

2. Для одномерного массива произвольной размерности напишите пять отдельных функций: вычисления среднего значения, определения максимального и минимального значения, подсчет количества элементов больших и равных заданному значению. Протестируйте работу этих функций на случайном наборе вещественных значений.

3. Разработайте программу работы с прямоугольной матрицей

Для этого:

А) Объявите указатель на двухмерный массив:

```
int** pMat
```

Организуйте ввод размерности матрицы с клавиатуры

```
  printf("Введите количество строк: ");
  scanf("%d", &n);
  printf("Введите количество столбцов: ");
  scanf("%d", &m);
```

Б) Выделите память для массива:

```
*pMat=malloc(sizeof(int)*n);
    for(inti = 0; i<n; i++)   //цикл выделения памяти
```

```
под каждую строку массива
    pMat [i] = malloc(sizeof(int)*m);
```

Заполните массив случайными числами.

В) Напишите функцию подсчета отрицательных элементов в строках, которые содержат хотя бы один нулевой элемент:

```
    int count_negatives (int **p, int n, int m)
    {
        int j, i = 0;
        int count = 0;
        bool  flag;                                  //флаг о
нахождения один нулевой элмент в строке
    while (i < n)
    {
        flag = false;
        for (j = 0; j < m; j++)
            if (p[i][j]==0)
    {
        flag = true;
        break;
            }
        if (flag)
        for (j = 0; j < m; j++)
            if (p[i][j] < 0)
        count++;
        i++;
        }
        return count;}
```

Г) Напишите функцию вывода элементов массива:

```
    arrprint(p, n, m);//   p – указатель на массив
```

Д) Организуйте вызов функций и убедитесь в работоспособности программы.

18.5 Контрольные задания

1. Составить программу, которая формирует двумерный массив, каждый элемент которого равен сумме его индексов.
2 .Составить программу, которая формирует двумерный массив и вычисляет значение среднего арифметического его элементов.
3. Составить программу, которая формирует двумерный массив и транспонирует его.
4. Составить программу, которая формирует двумерный массив и вычисляет сумму элементов каждого столбца.
5. Найти максимальный элемент в двумерном массиве и заменить его значение значением суммы его индексов.

6. Создайте двумерный массив целых чисел. Удалите из него строку и столбец, на пересечении, которых расположен минимальный элемент.

7. Дана матрица C. Заменить каждый элемент, лежащий на главной диагонали, наибольшим элементом соответствующей строки, и вывести полученную матрицу на экран.

8. Дана матрица C, состоящая из положительных и отрицательных элементов. Вместо отрицательных элементов в матрицу записать нули и подсчитать их количество. Вывести полученную матрицу и количество замен на экран.

9. Дана матрица F. Найти наибольший элемент матрицы. Поделить на этот элемент строку, где он расположен. Полученную матрицу вывести на кран.

10. Дана матрица A. Если элемент главной диагонали в строке положителен, строку оставить без изменений. В противном случае сменить знаки всех элементов строки на противоположные. Полученную матрицу вывести на экран.

ЛАБОРАТОРНАЯ РАБОТА № 19
АЛГОРИТМЫ СОРТИРОВКИ

Цель работы: Получение практических навыков работы с методами сортировок

Программные средства: MICROSOFT VISUAL STUDIO

19.1 Теоретические сведения

Сортировка – это упорядочение исходного набора сравнимых элементов (например, чисел по возрастанию или убыванию).

Ключ сортировки – это часть данных, определяющая порядок элементов, которая участвует в сравнениях, но при обмене элементов происходит перемещение всей структуры данных.

При решении задач сортировок массивов ключ и данные совпадают.

Основное требование к методам сортировки массивов – экономное использование памяти. В этом смысле говорят, что сортировку нужно выполнять *in site* (на том же месте).

Методы сортировки массивов "на месте" можно разбить на три основных класса:

- сортировка обменом (методом "пузырька" или простого обмена);
- сортировка выбором (простой перебор);
- сортировка вставками (сдвиг-вставка, вставками, вставка и сдвиг).

Рассмотрим массив целых или действительных чисел $a_1,...,a_n$. Пусть требуется представить элементы этого массива так, чтобы после перестановки они были упорядочены по неубыванию: a1≤ a2≤...≤ an. Для решения этой задачи можно воспользоваться, например, следующими алгоритмами:

а) Найти элемент массива, имеющий наименьшее значение, переставить

его с первым элементом, затем проделать то же самое, начав со второго элемента и т.д. (Сортировка выбором.)

б) Последовательным просмотром чисел a_1, ..., a_n найти наименьшее i такое, что $a_i > a_{i+1}$. Поменять местами a_i и a_i+1, возобновить просмотр с элемента ai_{+1} и т.д. Тем самым наибольшее число передвинется на последнее место. Следующие просмотры начинать опять сначала, уменьшая на единицу количество просматриваемых элементов. Массив будет упорядочен после просмотра, в котором участвовали только первый и второй элементы.(Сортировка обменами.)

в) Просматривать последовательно a_2, ..., a_n и каждый новый элемент a_i вставлять на подходящее место в уже упорядоченную совокупность a_1, ..., a_{i-1}. Это место определяется последовательным сравнением a_i с упорядоченными элементами a_1, ..., a_{i-1}. (Сортировка простыми вставками.)

Быстрая сортировка.

Входит в класс обменных сортировок (сортировка перемешиванием, пузырьковая сортировка и др.), выделяясь при этом высокой скоростью работы. Быстрая сортировка не требует дополнительной памяти.

Отличительной особенностью быстрой сортировки является операция разбиения массива на две части относительно опорного элемента. Например, если последовательность требуется упорядочить ***по возрастанию***, то в левую часть будут помещены все элементы, значения которых меньше значения опорного элемента, а в правую элементы, чьи значения больше или равны опорному. Вне зависимости от того, какой элемент выбран в качестве опорного, массив будет отсортирован, но все же наиболее удачным считается ситуация, когда по обеим сторонам от опорного элемента оказывается примерно равное количество элементов. Если длина какой-то из получившихся в результате разбиения частей превышает один элемент, то для нее нужно рекурсивно выполнить упорядочивание, т. е. повторно запустить алгоритм на каждом из отрезков.

Таким образом, алгоритм быстрой сортировки включает в себя два основных этапа:

- разбиение массива относительно опорного элемента;
- рекурсивная сортировка каждой части массива.

Время работы алгоритма зависит от того, как разбивается массив на каждом шаге. Если разбиение происходит на примерно равные части, время работы алгоритма будет наименьшим (сопоставимым с сортировкой слияния). Если размеры частей существенно отличаются, сортировка может занимать качественно большее время (подобно сортировке вставками).

Таким образом, алгоритм быстрой сортировки

Массив a[0]...a[N] и опорный элемент p, по которому будет производиться разделение.

1. Введем два указателя: i и j. В начале алгоритма они указывают, соответственно, на левый и правый конец последовательности.

2. Будем двигать указатель i с шагом в 1 элемент по направлению к концу массива, пока не будет найден элемент a[i] >= p. Затем аналогичным образом начнем двигать указатель j от конца массива к началу, пока не будет найден a[j] <= p.

3. Далее, если i <= j, меняем a[i] и a[j] местами и продолжаем двигать i,j по тем же правилам...

4. Повторяем шаг 3, пока i <= j.

5. Рекурсивно упорядочиваем подмассивы, лежащие слева и справа от опорного элемента.

Выбирать всегда средний элемент в качестве опорного – достаточно рискованно. Если известно, какой элемент будет выбран в качестве опорного, то можно подобрать такую последовательность, для которой сортировка будет происходить максимально медленно, за время порядка n^2. Поэтому в качестве элемента, относительно которого будет сортировка, берут либо случайный элемент, либо медиану из первого, последнего и среднего элементов.

```
mid = a[(first + (last- first)/2)];
```

Сортировка слиянием

Многие алгоритмы по своей природе рекурсивны: решая некоторую задачу, они вызывают самих себя для решения её подзадач. Делить на части можно до тех пор, пока в силу своего размера подзадача не станет тривиальной. Далее результаты решения подзадач объединяются. В этом состоит основная идея метода «разделяй и властвуй».

Этапы сортировки слиянием:

1. массив рекурсивно разбивается пополам, и каждая из половин делиться до тех пор, пока размер очередного подмассива не станет равным единице;

2. далее выполняется операция алгоритма, называемая слиянием. Два единичных массива сливаются в общий результирующий массив, при этом из каждого выбирается меньший элемент (сортировка по возрастанию) и записывается в свободную левую ячейку результирующего массива. После чего из двух результирующих массивов собирается третий общий отсортированный массив, и так далее. В случае если один из массивов закончиться, элементы другого дописываются в собираемый массив;

3. в конце операции слияния, элементы перезаписываются из результирующего массива в исходный массив.

19.2 Практические задания

Разработать программу в виде программного комплекса, включающего в себя модули (функции):

- модули сортировки;
- модуль формирования массива;
- печать массива;
- модуль замера времени;

• программу, реализующую диалог с пользователем.

Разрабатываемый программный комплекс должен обеспечивать

• вывод на экран меню;
• ввод исходной информации;
• формирования массивов с большим числом элементов;
• выбор метода сортировки;
• сортировку массива;
• печать результата;
• замеры времени выполнения сортировок массива.

Программа должна обеспечить сортировку массивов размером произвольной длины до 1000 элементов и выводить для контроля:

- при небольшом количестве элементов (например, менее 25) - неупорядоченный массив и массив после сортировки для каждого из предложенных алгоритмов;

- при значительном объеме данных (более 25) выводить время сортировки одного и того же массива для всех четырех предложенных алгоритмов.

Содержимое массива рекомендуется формировать с помощью генератора псевдослучайных чисел, замеры времени производить средствами модуля System.

В модуле сортировки реализуются:

- метод сортировки выбором;
- метод сортировки пузырьком;
- метод сортировки включением;
- метод сортировки слиянием.
- метод быстрой сортировки

Демонстрация работоспособности разработанных программных средств должна обеспечивать два варианта контроля: контроль работоспособности каждого из методов и контроль временных характеристик всех реализованных методов. Эффективность различных алгоритмов сортировки массивов, состоящих из *n* сортируемых элементов проводиться по двум критериям: числу необходимых сравнений элементов **C** и числу перестановок элементов **M**. Для всех прямых методов сортировки можно дать точные аналитические формулы. Они приведены в таблице 20.1.

Таблица 19.1

Сравнение прямых методов сортировки

Метод	Минимальное		Среднее	Максимальное
Прямое включение	$C =$ $M =$	$n - 1$ $2(n - 1)$	$(n^2 + n - 2) / 4$ $(n^2 - 9n - 10) / 4$	$(n^2 - n) / 2 - 1$ $(n^2 - 3n - 4) / 2$
Прямой выбор	$C =$ $M =$	$(n^2 - n) / 2$ $3(n - 1)$	$(n^2 - n) / 2$ $n * (\ln n + 0.57)$	$(n^2 - n) / 2$ $n^2 / 4 + 3(n - 1)$
Прямой обмен	$C =$ $M =$	$(n^2 - n) / 2$ 0	$(n^2 - n) / 2$ $(n^2 - n) * 0.75$	$(n^2 - n) / 2$ $(n^2 - n) * 1.5$

ЗАКЛЮЧЕНИЕ

Несмотря на появление все новых и новых языков программирования, Си остается по-прежнему широко распространен и востребован. Только язык Си позволяет быстро создавать компиляторы для новых платформ, драйверы устройств. Максимальная приближенность Си к архитектуре вычислительной машины позволяет писать на нем быстрые и эффективные программы. А современные тенденции к повсеместному использованию микроконтроллеров делают его незаменимым при создании управляющих программ. Поэтому владение языком Си важно для большинства специалистов в сфере информационных технологий и вычислительной техники.

Язык Си служит базовой платформой для изучения объектно-ориентированных языков C++, Java, C#. Он также относится к классу универсальных языков и с его помощью можно решить широкий круг прикладных задач. Полученные навыки программирования на Си обеспечат легких переход к другим языкам высокого уровня и явятся достаточным инструментом для решения большинства профессиональных задач.

Данное пособие предназначено для изучения языка Си, как первого языка программирования. Она охватывает такие «классические» принципы программирования, как линейные и разветвленные алгоритмы, организацию циклов и многовариантного управления процессом обработки, включает рассмотрение стандартных типов данных, массив, структур, работы со стандартными и файловыми потоками. В пособии представлено множество примеров кода для анализа, практических заданий и задач различного уровня сложности. Такой практикориентированный подход к изложению материала должен способствовать формированию необходимых для будущего ИТ-специалиста навыков программирования.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ашарина, И.В. Основы программирования на языках С и С++/И.В. Ащарина. – Москва: Горячая линия – Телеком, 2002 – 207 с.
2. Давыдов, В.Г. Программирование и основы алгоритмизации/ В.Г. Давыдов. – Москва: Высшая школа, 2003 – 447 с.
3. Златопольский, Д.М.Сборник задач по программированию/ Д.М. Златопольский. – СПб: Питер, 2011. – 304 с.
4. Керниган, Б. И. Язык программирования С/ Б. И. Керниган, Д.М. Ритчи. – 2-е издание,: Пер. с англ. – М.:Издат. дом «Вильямс» , 2007. – 304 с.: ил.
5. Кнут, Д.Э. Искусство программирования/ Д.Э. Кнут. – Том 1. Основные алгоритмы. Третье издание. Перевод с англ. Под общей редакцией Ю.В. Козаченко. – Москва: Издательский дом «Вильямс», 2002 – 692 с
6. Мартин, Роберт Чистый код. Создание, анализ и рефакторин/ Роберт, Мартин – СПб: Питер, 2010. – 464 с.
7. Окулов, С.М. Основы программирования [Электронный ресурс]/ Окулов С.М.— Электрон.текстовые данные.— М.: БИНОМ. Лаборатория знаний, 2012.— 336 с.— Режим доступа: http://www.iprbookshop.ru/6449.
8. Описание языка программирования ANSI C - https://www.opennet.ru/docs/RUS/ansi-c/
9. Павловская, Т.А. С/С++. Программирование на языке высокого уровня/Т.А. Павловкая. – СПб: Питер, 2013. – 461 с.: ил.
10. Павловская, Т.А.С/С++. Процедурное и объектно-ориентированное программирование [Текст]: учебник: допущено Министерством образования и науки Российской Федерации/ Т.А. Павловская. – Москва; Санкт-Петербург; Нижний Новгород [и др.] : Питер, 2015 (СПб. : Первая Академ. тип."Наука", 2014). - 495 с.
11. Павловская, Т. А. С++. Структурное программирование. Практикум/Т.А. Павловская, Ю.А. Щупак. – СПб: Питер, 2013. – 461 с.: ил.
12. Подбельский, В.В. Курс программирования на языке/ В.В. Подбельский, С.С. Фомин. – Си. М.: ДМК Пресс, 2007.– 239 с.
13. Шилдт, Г. С: Полное руководство, классическое издание/ Г.С. Шилдт – Москва: Издательский дом «Вильямс», 2002 – 704 с.
14. ISO/IEC 9899:2011 Information technology - Programming languages – C

ПРИЛОЖЕНИЕ

Варианты контрольных работ по темам 1-7

Время выполнения работы – 2 аудиторных часа.

Сложность определяется количеством баллов.

Вариант 1.

1. (3 балла) Вывести все нечетные числа кратные трем в интервале от A до B (возможно A>B и B>A) включительно.

2. (2 балла). С клавиатуры вводятся два целых числа. Сравнить эти числа. Результат вывести в виде 3<5 или 3=3, или 3>2. Примечание функцию printf() для вывода результата разрешается использовать только один раз.

3. (1 балл) Составить программу, которая генерирует 10 случайных чисел в заданном пользователем диапазоне (от A до B).

Примечание. Используйте функцию rand(), которая возвращает псевдослучайное целое число в диапазоне от 0 до RAND_MAX.

Вариант 2

1. (3 балла) Вывести все четные числа кратные двум в интервале от A до B (не включая A и B).

2. (1 балл). С клавиатуры вводятся два целых числа A и B. Сравнить эти числа. Результат вывести в виде сообщения («числа равны», «числа неравны»).

3. (2 балла) Составить программу, которая генерирует 10 случайных рациональных чисел в интервале от -1 до 1.

Примечание. Используйте функцию rand()которая возвращает псевдослучайное целое число в диапазоне от 0 до RAND_MAX.

Вариант 3

1. (3 балла) Вывести все четные числа кратные пяти в интервале от A до B (возможно A>B и B>A) включительно.

2. (1 балл) Дан номер года. Определить количество дней в этом году. Примечание. При решении задачи принять во внимание, что в современном (григорианском) календаре каждый год, номер которого делится на 4, является високосным, за исключением тех номеров, которые делятся на 100 и не делятся на 400.

3. (2 балла) Напечатать таблицу, выводящую ежемесячную выплату по кредиту в течение года, если задана процентная ставка и сумма кредита. Примечание. Кредит дается на 1 год, процентная ставка – % от всей суммы кредита, сумма + проценты по кредиту распределяются равномерно на 12 месяцев.

Варианты контрольных работ по темам 7-15

Время выполнения работы – 2 аудиторных часа.

Сложность определяется количеством баллов.

Вариант 1

1. (2 балла) Написать программу, которая считывает целые числа из файла и вычисляет их среднее значение.

2. (3 балла) Получить из заданного файла все простые числа, не превосходящие заданного числа X.

3. (2 балл) С клавиатуру вводиться год. Распечатать таблицу в первой колонке, которой указан месяц в виде строки, а во второй – количество дней. Процедуру печати таблицы реализовать в виде отдельной функции.

4. (3 балла) Из введенной пользователем строки удалить окончания всех слов (знаки препинания оставить без изменения).

5. (1 балл) Написать программу, которая отвечает – делятся ли введенные пользователем числа нацело в виде фразы (например, «число 10 делиться на 2»).

Вариант 2

1. (1 балл) Записать в файл последовательность случайных вещественных чисел не превышающих 0,1 заданной пользователем длины.

2. (3 балла) В заданной пользователем строке заменить все первые буквы слов на строчные (большие).

3. (2 балла) Из файла, содержащего множество целых чисел выбрать все, соответствующие степени 5. Процедуру выбора чисел реализовать в виде отдельной функции.

4. (1 балл) Написать программу, которая после введенного с клавиатуры числа копеек, вывод сообщение вида «N рубля/рублей и k копеек/копейки».

5. (1 балл) Напечатайте заданный пользователем символ kраз в строке и n строк (k иn – задает пользователь).

Вариант 3.

1. (1 балл) Записать в файл все четные числа кратные пяти в интервале от A до B (возможно A>B и B>A) включительно.

2. (3 балла) Для каждого вещественного числа считанного из файла сформируйте таблицу из трех столбцов – число, целая часть, дробная. Процедуру печати таблицы реализуйте в виде отдельной функции.

3. (2 балла) В заданной пользователем строке удвоить все буквы.

4. (1 балл) Написать программу, которая якобы считывает текущее время (реализовать функцию генерирующую время в секундах от 1 января 2000 года случайным образом) и выводит на экран дату и время.

5. (3 балла) Из введенной пользователем строки удалить все цифры. Специально реализовать функцию распознавания цифр.

Варианты контрольных работ по темам 15-19

Требования к работе.

1. Разрабатываемая программа должна быть представлена в виде программного комплекса, включающего в себя модули (функции).

2. Предоставление возможности пользователю инициализировать многомерный массив, как с консоли, так и случайным образом.

3. На экран должен выводиться исходный и преобразованный массив, а так же результаты с комментариями. Элементы массива выводить с их порядковыми номерами.

Для обеспечения уникальности значений элементов в рамках массива, использовать алгоритм, реализующий последовательный ввод элементов массива X размером N с клавиатуры. После ввода значения очередного элемента выполняется проверка наличия аналогичных значений в массиве. В том случае, если введенное значение раньше уже встречалось в массиве, пользователю предлагается осуществить ввод повторно.

Оценка «отлично» ставится, если:

- работа выполнена полностью;

- в графическом изображении алгоритма (блок-схеме), в теоретических выкладках решения нет пробелов и ошибок;

- в тексте программы нет синтаксических ошибок (возможны одна-две различные неточности, описки, не являющиеся следствием незнания или непонимания учебного материала).

Оценка «хорошо» ставится, если:

- работа выполнена полностью, но обоснования шагов решения недостаточны (если умение обосновывать рассуждения не являлось специальным объектом проверки);

- допущена одна ошибка или два-три недочета в чертежах, выкладках, чертежах блок-схем или тексте программы.

Оценка «удовлетворительно» ставится, если:

- допущены более одной ошибки или двух-трех недочетов в выкладках, чертежах блок-схем или программе, но студент владеет обязательными умениями по проверяемой теме.

Оценка «не удовлетворительно» ставится, если:

- допущены существенные ошибки, показавшие, что студент не владеет обязательными знаниями по данной теме в полной мере.

- работа показала полное отсутствие у студента обязательных знаний и умений по проверяемой теме.

Индивидуальные варианты к темам 14-18

1	Написать программу, которая удаляет все четные элементы целочисленного массива
2	Написать программу, которая удаляет элемент из массива, если среднее арифметическое его «соседей» меньше 5
3	Написать программу, которая добавляет между двумя соседними элементами целочисленного массива их среднее арифметическое, если

	оно больше 5
4	Написать программу, которая удаляет все элементы, большие 5 из целочисленного массива, и при этом подсчитывает их количество
5	Написать программу, которая находит минимальный элемент целочисленного массива, исключает его и затем включает его в конец того же массива
6	Написать программу, которая включает после каждого отрицательного элемента целочисленного массива его модуль
7	Дан массив целых чисел. Удалить элементы с 4-го по 8-й в массиве из N элементов
8	Записать отрицательные элементы массива A в массив B.
9	Написать программу, которая удаляет из целочисленного массива одинаковые подряд идущие элементы, оставляя их в массиве по одному
10	В упорядоченный массив по убыванию, необходимо вставить, элемент b не нарушив упорядоченности массива
11	Вывести новый массив значений, в котором каждое значение равно наименьшему значению из трех соседних элементов исходного массива
12	Вывести массив в обратном порядке
13	Написать программу, которая находит максимальный элемент целочисленного массива, исключает его и затем включает его в начало того же массива

Индивидуальные варианты к темам 16-18.

1.	Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы размером 10 на 10. Найти сумму модулей элементов, расположенных выше главной диагонали.
2.	На основе исходного двумерного массива C с рабочим размером n x n, сформировать одномерный массив A. Массив A должен содержать только те числа, которые встречаются в массиве C более одного раза.
3.	Уплотнить заданную матрицу, удаляя из нее строки и столбцы, заполненные нулями. Найти номер первой из строк, содержащих хотя бы один положительный элемент.
4.	Осуществить циклический сдвиг элементов прямоугольной матрицы на *n* элементов вправо или вниз (в зависимости от введенного режима), *n*может быть больше количества элементов в строке или столбце.
5.	Осуществить циклический сдвиг элементов квадратной матрицы размерности MxN вправо на *k* элементов таким образом: элементы 1-й строки сдвигаются в последний столбец сверху вниз, из него — в последнюю строку справа налево, из нее — в первый столбец снизу вверх, из него — в первую строку; для остальных элементов — аналогично.

6.	Дана целочисленная прямоугольная матрица. Определить номер первого из столбцов, содержащих хотя бы один нулевой элемент. Определить номер первой из строк, содержащей хотя бы один нулевой элемент.
7.	Дана целочисленная прямоугольная матрица. Найти номер первого из столбцов, не содержащих ни одного отрицательного элемента. Подсчитать количество положительных и отрицательных элементов в матрице.
8.	Путем перестановки элементов квадратной вещественной матрицы добиться того, чтобы ее максимальный элемент находился в левом верхнем углу, следующий по величине — в позиции (2,2), следующий по величине — в позиции (3,3) и т. д., заполнив, таким образом, всю главную диагональ. Найти номер первой из строк, не содержащих ни одного положительного элемента
9.	Дана целочисленная квадратная матрица. Определить: сумму элементов в тех строках, которые не содержат отрицательных элементов; минимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.
10.	Дана целочисленная прямоугольная матрица. Определить: количество отрицательных элементов в тех строках, которые содержат хотя бы один нулевой элемент; номера строк и столбцов всех седловых точек матрицы. Матрица A имеет седловую точку A_{ij}если A_{ij}является минимальным элементом в i-й строке и максимальным в j-м столбце.

Индивидуальные варианты к темам 16, 18, 19

1	Отсортируйте по неубыванию методом "пузырька" одномерный целочисленный массив, заданный случайными числами на промежутке [-100; 100). Выведите на экран исходный и отсортированный массивы.
2	Отсортируйте по невозрастанию методом простого выбора одномерный вещественный массив, заданный случайными числами на промежутке [0; 50). Выведите на экран исходный и отсортированный массивы.
3	Отсортируйте по возрастанию методом простого включения одномерный целочисленный массив, заданный с клавиатуры различными числами. Выведите на экран исходный и отсортированный массивы.
4	Отсортируйте по убыванию одномерный целочисленный массив, заполненный случайными числами с помощью алгоритма Шейкера. Выведите на экран исходный и отсортированный массивы.
5	Отсортируйте по убыванию одномерный вещественный массив, заполненный случайными числами с помощью алгоритма быстрой сортировки. Выведите на экран исходный и отсортированный массивы.
6	Упорядочить по возрастанию отдельно элементы, стоящие на четных

	местах, и элементы, стоящие на нечетных местах.
7	Преобразовать массив таким образом, чтобы сначала располагались все элементы, отличающиеся от максимального элемента не более чем на 20%, а потом — все остальные.
8	Задан массив из 20 натуральных чисел. Создать новый массив, в котором числа расположить в порядке убывания их значений. Напечатать в две строки элементы исходного массива и элементы нового массива, а также напечатать индексы (номера элементов) исходного массива после сортировки. Так же вывести минимальное и максимальное значения массива с порядковым номером до сортировки и после сортировки. Применить алгоритм простым выбором.
9	Задан массив из N действительных чисел (N может быть задано не более 25). Отрицательные числа в заданном массиве упорядочить по убыванию их значений. Вывести на экран исходный и новый массивы. Применить алгоритм Шейкера.
10	Задан массив из N натуральных чисел (N может быть задано любым двузначным числом). Сформировать новый массив, поместив в него из исходного только простые числа. Полученный массив упорядочить по убыванию их значений. Вывести на экран новый массив.
11	Отсортировать по возрастанию четные элементы массива. Применить алгоритм простыми вставками.
12	Отсортировать по убыванию элементы, стоящие на четных местах.
13	Объединить два одинаково упорядоченных массива разного размера в один, так же упорядоченный (сортировать новый массив нельзя): а) элементы расположить также как в исходных массивах; б) элементы расположить в обратную сторону.
14	Определите количество совпадающих элементов в двух одинаково упорядоченных массивах. Размеры массивов не обязательно одинаковы. При вводе массивов проверять их упорядоченность.
15	Переупорядочить заданный одномерный массив таким образом, чтобы его элементы шли в порядке возрастания их близости к целым числам.

Индивидуальные варианты к темам 17 - 19

1	Дана действительная матрица размера n×m, упорядочить (переставить) строки матрицы (используя алгоритмы сортировки): по неубыванию значений первых элементов строк
2	Дана действительная матрица размера n×m, упорядочить (переставить) столбцы матрицы (используя алгоритмы сортировки): по неубыванию значений наименьших элементов столбцов
4	Дана действительная матрица размера n×m, упорядочить (переставить) строки матрицы (используя алгоритмы сортировки): по неубыванию значений наименьших элементов строк

5	Дана действительная матрица размера n×m, упорядочить (переставить) строки матрицы (используя алгоритмы сортировки): по невозрастанию значений наибольших элементов строк.
6	На основе исходных массивов A[n] и B[m] (n и m – рабочие размеры массивов) сформировать массив C, который будет состоять из чисел, которые входят в массив B, но при этом не входят в массив A. Упорядочить массив C по возрастанию, используя метод «пузырька». Вывести элементы массивов A, B и C на экран.
7	На основе исходного двумерного массива C с рабочим размером n x n, сформировать одномерный массив A. Массив A должен содержать только те числа, которые встречаются в массиве C только один раз. Упорядочить массив A по убыванию, используя метод выбора. Вывести массив A на экран.
8	Используя метод простого перебора, упорядочить по возрастанию значения, содержащиеся в столбцах двумерного массива C с рабочим размером n x m. Вывести массив C на экран.
9	Используя метод «пузырька», упорядочить по возрастанию значения, содержащиеся в строках двумерного массива C с рабочим размером n x m. Вывести массив C на экран.
10	Используя метод простого перебора, упорядочить по убыванию значения, содержащиеся в строках двумерного массива C с рабочим размером n x m. Вывести массив C на экран.
11	Используя метод «пузырька», упорядочить по возрастанию значения, содержащиеся в столбцах двумерного массива C с рабочим размером n x m. Вывести массив C на экран.

Практикум

Курипта Оксана Валериевна
Минакова Ольга Владимировна
Проскурин Дмитрий Константинович

ОСНОВЫ ПРОГРАММИРОВАНИЯ И АЛГОРИТМИЗАЦИИ

Отпечатано в авторской редакции

Подписано в печать 10. 11. 2015.Формат 60 x 84 1/16 Уч.-изд.л. 8,3.
Усл.-печ.л. 8,4. Бумага писчая. Тираж 500 экз. Заказ № 468.

Отпечатано: отдел оперативной полиграфии
издательства учебной литературы и учебно-методических пособий
Воронежского ГАСУ

396006, Воронеж, ул. 20-летия Октября,84