БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

А.М. Ковальчук

Ю.А. Луцик

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

В 2-х частях

Часть 1

Минск – 2007

Министерство образования Республики Беларусь
Учреждение образования
Белорусский государственный университет
информатики и радиоэлектроники

А.М. Ковальчук, Ю.А. Луцик

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по дисциплине «Основы алгоритмизации и программирования»
для студентов специальности
I-40 02 01 «Вычислительные машины, системы и сети»

В 2-х частях

Часть 1

Минск 2007

УДК
ББК

Ковальчук, А.М., Луцик, Ю.А.
Лабораторный практикум по дисциплине «Основы алгоритмизации и программирования» для студентов специальности I-40 02 01 «Вычислительные машины, системы и сети»: в 2ч.
Ч.1/А.М. Ковальчук, Ю.А. Луцик. – Мн.:БГУИР,2007. –
ISBN

1-я часть лабораторного практикума содержит лабораторные работы по дисциплине «Основы алгоритмизации и программирования», читаемой студентам специальности «Вычислительные машины, системы и сети» в первом семестре. 1-я часть практикума посвящена построению блок-схем алгоритмов, изучению основных типов данных, операторов, циклов, массивов, указателей и функций. Приведены варианты заданий по изучаемым темам.

Введение

Курс "Основы алгоритмизации и программирования" (ОАиП) занимает важное место в учебном процессе по подготовке инженера-системотехника.

Методические указания составлены на основе программы дисциплины "ОАиП" для высших учебных заведений по специальности 40 02 01 "Вычислительные машины, системы и сети".

В результате выполнения лабораторных работ по курсу студенты должны получить знания по теоретическим основам алгоритмизации задач и проектирования программ, создания программ на алгоритмических языке С(С++), научиться отлаживать и выполнять на ЭВМ конкретные задачи с использованием современных методов программирования.

В процессе выполнения лабораторных работ студенты изучают некоторые стандартные алгоритмы решения задач, а также разрабатывают на их основе новые алгоритмы. Графически разрабатываемые алгоритмы изображаются в виде блок-схем для построения, которых студенты знакомятся со стандартами. Разрабатываемые алгоритмы, реализуются на языке С. Для этого студенты используют знания, получаемые в теоретическом курсе.

Основные вопросы, рассматриваемые при выполнении лабораторных работ: организация ввода/вывода информации, операции и операторы языка, структурированные типы данных, массивы, указатели и операции над указателями, строковые данные, функции.

Материал рассматриваемый на лабораторных работах является основой для изучения дисциплин: "Конструирование программ и языки программирования", "Объектно-ориентированное программирование" и др.

Лабораторная работа №1

Построение блок-схем алгоритмов вычислительных процессов

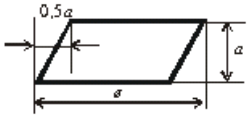
Цель работы: Изучить принципы построения блок-схем алгоритмов линейных, разветвляющихся, циклических и итерационных вычислительных процессов.

Краткие теоретические сведения

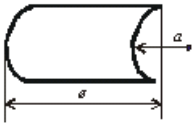
Линейный вычислительный процесс

Вычислительный процесс называется линейным, если направление его продолжения на любом этапе вычислений является единственным. Линейный процесс можно представить в виде следующих этапов: первый – задание исходных данных; второй – реализация вычислений; третий – вывод результатов. Правила построения блок-схем алгоритмов приводятся в ГОСТ 19-002-80 и ГОСТ 19-003-80. Согласно этим ГОСТ, все размеры фигур связаны с двумя величинами: a и b, где a – величина, кратная 5, а b вычисляется по формуле b=1,5a, допускается b=2a. В январе 1992 года введен новый ГОСТ 19-701-90. Он описывает, как и где следует использовать фигуры. Схемы алгоритмов состоят из имеющих заданное значение символов, краткого пояснительного текста и соединяющих линий.

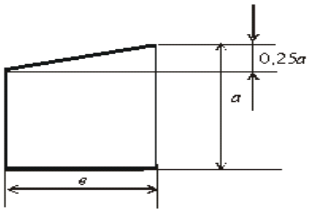
Описание символов



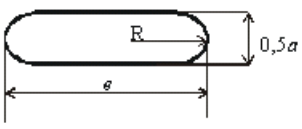
Символ отображает данные, носитель которых не определен



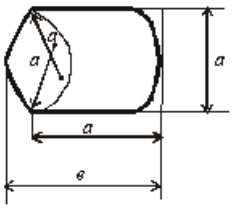
Символ отображает хранимые данные в виде, пригодном для обработки, носитель данных не определен



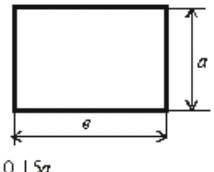
Ручной ввод. Символ отображает данные, вводимые вручную во время обработкис устройств любого типа (клавиатура, переключатели, кнопки световое перо).

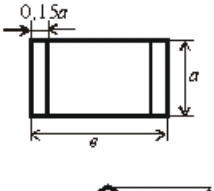


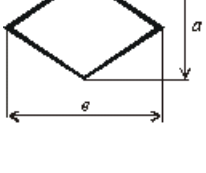
Символ отображает выход во внешнюю среду и вход из внешней среды (начало или конец схемы программы).

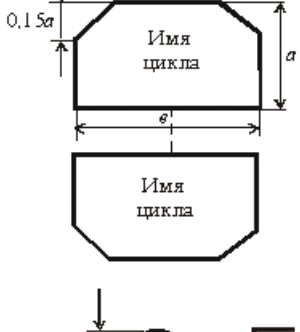


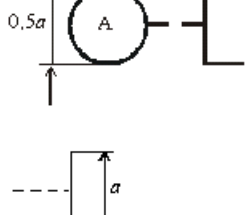
Символ отображает данные, представляемые в человекочитаемой форме на носителе в видеотображающего устройства (экран для визуального наблюдения)

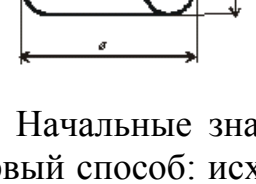












Символ отображает функц
(выполнение определенной
водящее к изменению значе
ции

Символ отображает пр
одной или нескольких опера
определены в другом месте (

Символ отображает решени
имеющую один вход и ряд
один из которых может б
условий, определенных вн
результаты вычисления мог
отображающими эти пути.

Символ, состоящий и
цикла. Обе части символа
Условия для инициализа
помещаются внутри символ
расположения операции, про

Начальные значения переменных можно задать несколькими способами. Первый способ: исходные данные переменным задаются с помощью оператора присваивания. Второй способ: исходные данные переменным задаются путем ввода их из внешних устройств. На третьем этапе необходимо предусмотреть вывод результатов на удобный для чтения носитель информации.

Пример: Составить блок-схему вычисления значения функции

$$y = \frac{5a * x}{b} - \frac{cx^2}{x^3 - a}$$ при следующих значениях переменных: x=0.1; a=2; b=2.5; c=3.

Блок-схема алгоритма приведена на рис.1.

Ошибка! Ошибка связи.

Символ отображает в
ти этой схемы и использует
другом месте. Соответств
содержать одно и то же уник

Рисунок 1. Блок-схема линейного вычислительного процесса

Разветвляющимся называют вычислительный процесс, который в зависимости от исходных данных и результатов промежуточных вычислений осуществляется по одному из нескольких возможных вариантов. Варианты (направление вычислений), по которым может реализовываться вычислительный процесс, называют ветвями. Выбор ветви для продолжения вычислений зависит от результатов проверки некоторого условия. Если условие выполняется, то выбирается одна ветвь, если не выполняется, то другая ветвь (рис.2).

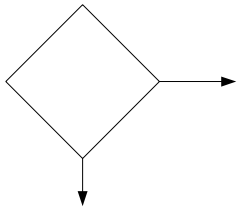
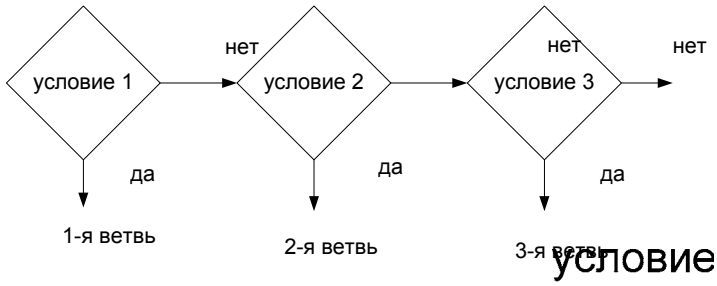


Рисунок 2. Выбор ветви выполнения вычислительного процесса

Условие, определяющее выбор той или иной ветви вычислений, может быть сложным, т.е. состоять из нескольких простых условий (рис.3).



нет

условие

2-я ветвь

Пример. Составить блок-схему алгоритма нахождения корней квадратного уравнения. Коэффициенты уравнения ввести из клавиатуры (рис.4).

да

1-я ветвь

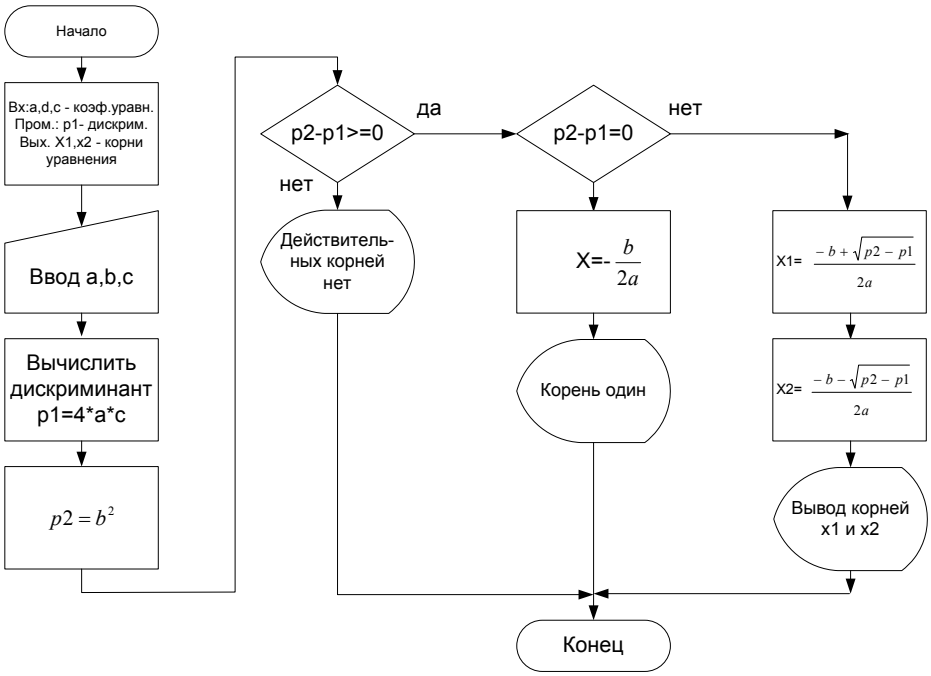


Рисунок 4. Блок-схема алгоритма вычисления корней квадратного уравнения

Построение блок-схем алгоритмов циклических вычислительных процессов

В большинстве задач, встречающихся на практике, необходимо вычисления по некоторой группе формул осуществлять многократно. Этот многократно повторяющийся участок вычислительного процесса называют циклом. Цикл, не содержащий в себя другие циклы, называют простым. Цикл называется сложным, если он содержит внутри себя другие циклы или разветвления. Обычно при каждом повторении цикла вычисления осуществляются с новыми значениями переменных. В любом циклическом процессе в ходе вычислений необходимо решать вопрос: повторять вычисления или нет? Ответ на этот вопрос получают в результате анализа значений одной или нескольких переменных, т.е. анализ некоторого условия. Анализируемую переменную называют параметром цикла. Из вышеизложенного следует, что циклический процесс является разветвляющимся вычислительным процессом с двумя ветвями, из которых одна возвращается на предыдущие блоки, т.е. реализует цикл. Блок-схема циклического процесса, независимо от многообразия сводящихся к нему задач, должна содержать блок задания начального значения параметру цикла (2), блок реализации необходимых вычислений (3), блок изменения параметра цикла (4) и блок проверки условия окончания цикла (5)(рис.5.)

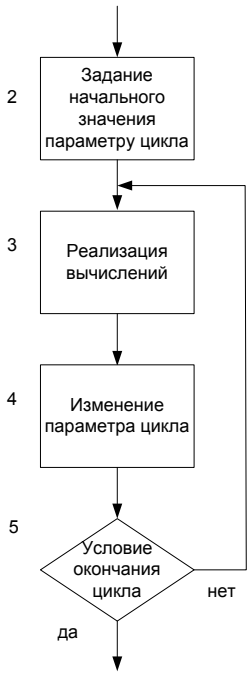


Рисунок 5. Блок-схема циклического процесса

При каждом выполнении цикла выполняются по формулам вычисления (3), изменяются параметры цикла (4), и в зависимости от результата проверки цикл повторяется, начиная с блока 3, или заканчивается.
Пример: составить блок-схему нахождения суммы чисел от 1 до 15 (рис.6).

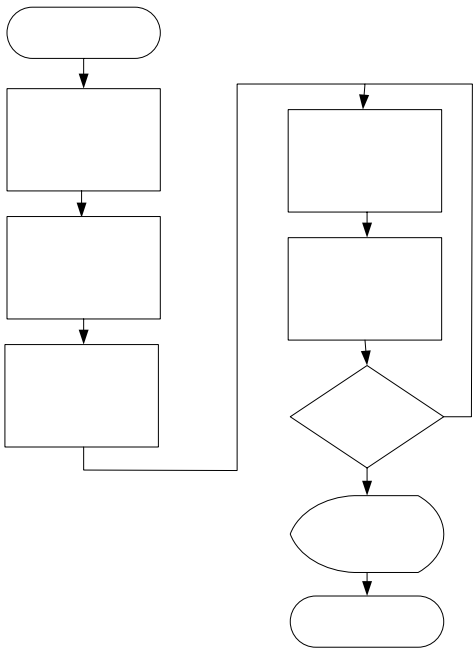


Рисунок 6. Блок-схема алгоритма нахождения суммы чисел

Используя символы для изображения циклов, эту блок-схему алгоритма можно представить в следующем виде (рис.7).

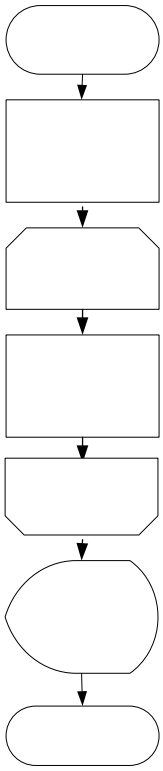


Рисунок 7. Блок-схема циклического процесса

Построение блок-схем алгоритмов итерационных процессов

Итерационным вычислительным процессом называется такой циклический процесс, который продолжается до тех пор, пока разность между соседними, уточняемыми на каждом шаге цикла (итерации) значениями, не окажется меньше или равной некоторой заданной величине. Характерной особенностью итерационного процесса является то, что в нем количество повторений вычислений заранее неизвестно и становится определенным только после окончания вычислений. Решение об окончании вычислений принимается тогда, когда результаты счета (значение функции, искомые величины на очередной отличаются от предыдущих или эталонных не более, чем на некоторую, наперед заданную величину, т.е. найдены с заданной точностью.

Второй особенностью итерационного процесса является то, что результаты вычислений очередного выполнения цикла используются как исходные данные при следующем выполнении цикла, т.е. решение находится последовательными приближениями, путем уточнения на каждом шаге цикла.

Пример: для 15 значений переменной X вычислить функцию cosx, используя формулу Тейлора.

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \ldots$$

Вычисления ряда необходимо прекратить, как только очередной учитываемый член ряда окажется по абсолютной величине не больше заданного числа ξ (но в разложении ряда не должно быть более 50 членов). Знаменатель очередного члена ряда рекомендуется вычислить используя значения предыдущего знаменателя и умножая его на очередные два числа натурального ряда чисел. Числитель каждого последующего члена получается из предыдущего умножением на $-x^2$. Это одновременно позволит изменять знак каждого очередного члена (рис.8).

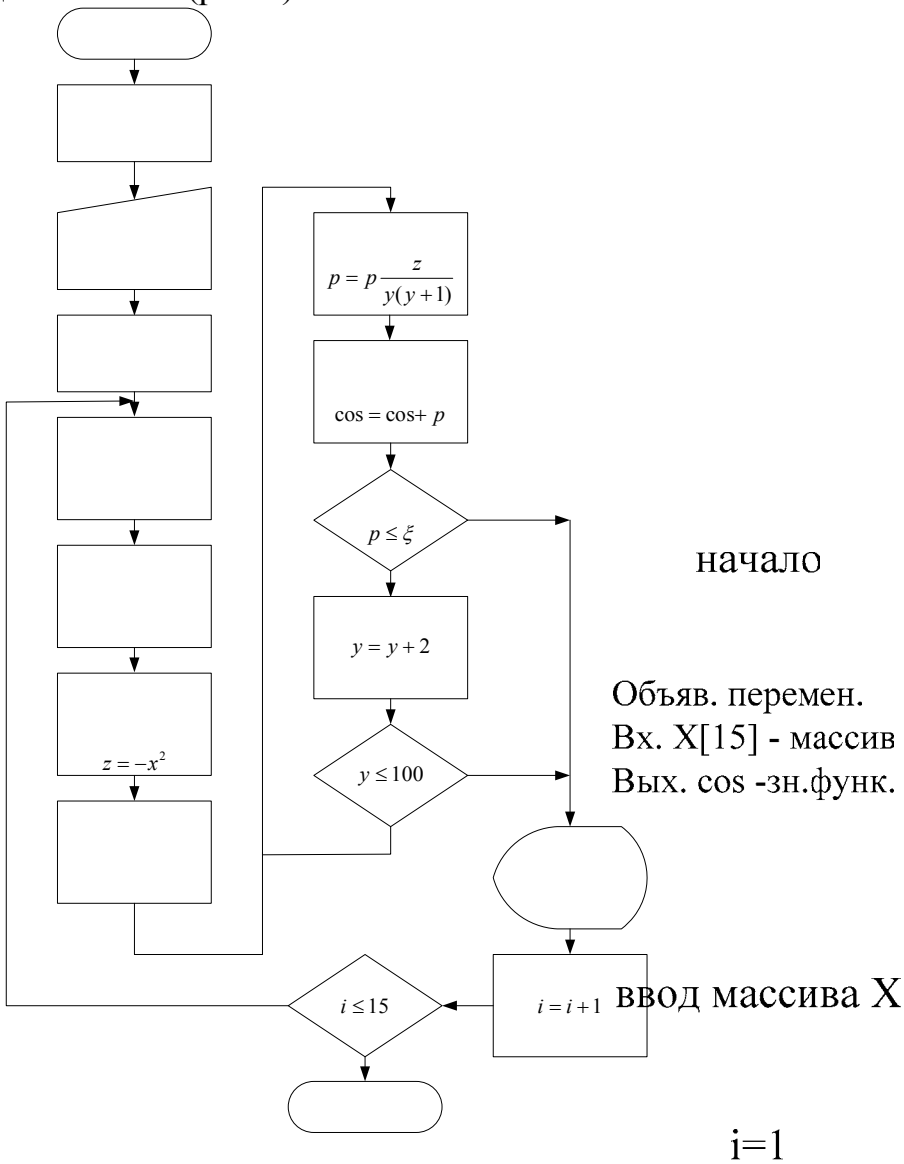


Рисунок 8. Блок-схема алгоритма итерационного вычислительного процесса.

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.

Варианты заданий

Составить блок-схемы следующих вычислительных процессов:

1. Вычислить значение функции

$$y = 1.5x * \sin x^2 + \sqrt{\cos^2 x + 1} - \frac{24x^3 - \sqrt{x}}{\cos(x + 0.2)} + e^{x+1} \quad при \text{ x} = 0.9$$

2. Вычислить $Y = \sum_{i=1}^{30} i^2$

3. Дан массив из 10 элементов. Сформировать массив, поменяв местами элементы, стоящие на четных и нечетных местах:

а) сформировать массив на месте исходного массива;

б) сформировать новый массив.

4. Дан массив из 10 элементов. Сформировать массив, у которого первым элементом будет последний элемент исходного массива. Вторым – предпоследний и т.д. (т.е. расположить элементы массива в обратном порядке):

а) сформировать массив на месте исходного массива;

б) сформировать новый массив.

5. Дан двухмерный массив A(6,9). Найти сумму элементов каждой строки.

6. Умножить матрицу A(4,5) на вектор B(5). Результат произведения сохранить в новой матрице A1.

7. Дана матрица A(6,9). Найти сумму элементов каждой строки. Результат сохранить в массиве B.

8. Вычислить функцию по формуле $Y = \frac{ax + bx + c}{(bx^2 - \sqrt{x + a\cos x})c}$ для x от 0.2 до 1.5 с шагом 0.01. Если знаменатель меньше 0.0001 по абсолютному значению, то положить $Y=10^5$.

9. Дан массив C(20). Вычислить $y = a^k + d$

$a = \sum_{i=2,4,\ldots}^{20} C_i$; $d = \sum_{i=1,3,\ldots}^{19} C_i$; $k = 3$, если $C_i > 0$ и $k = 2$, иначе, т.е. $C_i < 0$.

10. Дан массив X(20). Вычислить

$Y = \frac{b^4 + x_i^2}{b - x_i}$, если $x_i \le 0$ и $Y = \frac{(b_i + x_i)^2}{x_i}$, если $x_i > 0$

где $b = \max(x)$, т.е. максимальный элемент массива X.

Лабораторная работа №2

Типы данных, основные операции языка С, структура простой программы на языке С. Функции ввода/вывода.

Цель работы: Изучить типы данных, основные операции языка С и форматированные функции ввода/вывода. Рассмотреть структуру простой программы.

Теоретические сведения

Элементы программы

Составными элементами программы являются лексемы, которые в свою очередь, состоят из символов, относящихся к базовому словарю, распознаваемому компилятором. Множество используемых символов включает в себя следующие:
строчные символы - a-z, прописные символы - A-Z, цифры - 0-9 и
символы +, =, -, _, (,), *, ;, %, #, &, !, ”, ’, \, [,], {, }, ~ , ^.
Различают пять разновидностей лексем: ключевые слова, идентификаторы, константы, операторы и пунктуация.

Ключевые слова строго зарезервированы и имеют фиксированный смысл. Они используются для объявления типов (такие как int, char, float и др.), для синтаксических операторов (такие как do, for, if и др.) и другие.

Идентификатор (или имя переменной, функции и др.) – это последовательность букв, цифр и подчеркиваний. Идентификатор не может начинаться с цифры. Прописные и строчные буквы обрабатываются как различные символы.

Литералы – это постоянные значения, такие как 1 или 3.14519. Все встроенные типы данных С имеют такие литералы, как символы, целые числа, числа с плавающей точкой и указатели. Допускаются также строковые константы. Например:
5 – целочисленная константа;
5u – u или U определяет unsigned;
5l - l или L определяет long;
05 – восьмеричная;
0x5 – шестнадцатеричная;
5.0 – с плавающей точкой;
‘5’ – символьная константа;
“5” – строковая константа, состоящая из ‘5’ и ‘\0’.

Операторы и знаки пунктуации

Большому количеству символов и символьных последовательностей в С придается особое значение, например:
+, -, *, /, % - арифметические операторы;
&&, !, || - логические операторы;
=, +=, *= - операторы присвоения.
Операторы используются в выражениях. Они имеют фиксированный приоритет. Знаки препинания включают круглые скобки, фигурные скобки, запятую и двоеточие.

Простые типы

В С имеются следующие простые встроенные типы: double, int и char. Простые типы в С могут быть модифицированы с помощью ключевых слов short, long, signed, unsigned.

Простые типы данных

char	signed char	unsigned char
short	int	long
unsigned short	unsigned	unsigned long
float	double	long double

Диапазоны целочисленных значений, представленных в вашей системе, определены в стандартном файле limits.h. Диапазон значений чисел с плавающей запятой находятся в стандартном файле float.h.

Инициализация

Объявление переменной связывает тип с ее именем. Большинство объявлений переменной являются так же определениями. Определение переменной распределяет под нее память. Переменная может быть проинициализирована следующим образом:

```
double radius = 5.5;   // объявлено, определено и инициализировано
double a;              // объявлено и определено, но не проинициализировано
```

Инициализация может включать произвольное выражение при условии, что все переменные и функции, используемые в выражении, определены. Нельзя ссылаться на неинициализированную переменную. С позволяет многократное присвоение в одном операторе.

```
y=z=3.5;          //  эквивалентно z=3.5; y=3.5;
a=b+(c=3);        //  эквивалентно c=3; a=b+c;
a+=b;             // a=a+b;
a*=a+b;           // a=a*(a+b);
```

С поддерживает операторы инкрементации ++ и декрементации -- в префиксной и постфиксной формах.

```
++i;      // префиксная форма записи
--i;
i++;      // постфиксная форма записи
i--;
```

Преобразование типов

Если выражение имеет смешанный тип операндов, то согласно иерархии типов

int < unsigned < long < unsigned long < float < double

преобразуется к значению, совместимому с левосторонней переменной.

Функции ввода/вывода

Функция scanf

Функция scanf выполняет форматированный ввод данных из входного потока. В общем виде формат функции scanf следующий:

scanf(“формат ввода переменных”,адреса переменных, которым присваиваются значения);

Типы некоторых форматов вводимых переменных следующие:

%d - целое десятичное число;
%c - отдельный символ;
%s - символьная строка;
%f - число с плавающей запятой в записи с фиксированной десятичной точкой;
%e - значение со знаком в формате;
%p - значение указателя, т.е. адрес.

Функция scanf возвращает число правильно считанных полей.

Пример: ввод числа.

```
void main()
{
    int a,b;
    int n;
    printf(“\nВведите два целых числа a и b\n”);
    n=scanf(%d%d”,&a,&b);
    if(n!=2)
    {
        printf(“\nДанные введены неверно\n”);
        return;
    }
}
```

Функция gets считывает символьную строку из стандартного входного потока и размещает ее по адресу, заданному указателем. Прототип функции

char * gets(char *str); Чтение строки заканчивается, когда функция встречает символ ‘\n’. Данный символ заменяется ‘\0’.

Функция getchar считывает символ из стандартного входного потока. Прототип функции: int getchar(void);

Функция printf выводит форматированные данные в поток. Общий формат функции printf можно записать в следующем виде

printf(“форматы вывода данных”,имена переменных, значения которых необходимо вывести);

Форматы вывода данных используются такие же, как и для функции scanf.

Пример использования функции scanf и printf.

```
void main()
{
    int a,b,c;
    printf(“\nВведите два целых числа\n”);
    c=scanf(“%d%d”,&a,&b);
    if(c!=2)
    {
        printf(“Данные введены неверно\n”);
        return;
    }
    printf(“Значение a=%3d, b=%3d\n”,a,b);
}
```

Операторы if и if_else

Обобщенная форма оператора if имеет вид

```
if( выражение)
        оператор
```

если значение “выражение” отлично от нуля (true), то оператор выполняется, если равно нулю – оператор пропускается. Выражение в операторе if – это сравнение, равенство или логическое выражение.

Оператор **if_else** имеет форму:

```
if( выражение)
    оператор1
else
    оператор2
```

если выражение отлично от нуля, то выполняется оператор1, а оператор2 пропускается, если выражение равно нулю, то пропускается оператор1 и выполняется оператор2.

```
if(x<y)
    min=x;
else
    min=y;
printf(“\nmin=%3d”, min);
```

Структура простой программы

Любая программа на языке С состоит из одной или более функций. Функциям можно давать любые имена, но среди них есть одна головная функция, имя которой main. Выполнение программы всегда начинается с этой функции. Для выполнения определенных действий функция main() обычно обращается к другим функциям, часть из которых находится в той же самой программе, а часть – в библиотеках, содержащих ранее написанные функции.

Приведем структуру программы на языке С, в которой функция main() обращается к другой функции – prog(), которой не передаются параметры и она ничего не возвращает.

```
#include<stdio.h>
void main(void)                    // тело функции main
{
      void prog(void);             // прототип функции prog
       int a=10;                   // определение данных
       int b=13;
       int c;
       c=a*b;                      // вычисления
       printf("\n a+b=%4d",c);     // вывод результата
       prog();                     // вызов функции prog на выполнение
}
void prog(void)                    // тело функции prog
{
      printf("\n Выполняется функция prog");
}
```

Исходная программа состоит из следующих объектов: директив, указаний компилятору, объявлений и определений. Директивы задают действия препроцессора по преобразованию текста программы перед компиляцией. Указания компилятору – это команды, выполняемые компилятором во время компиляции. Объявления задают имена и атрибуты переменных, функций и типов, используемых в программе. Определения – это объявления, определяющие переменные и функции. Определение переменной в дополнение к ее имени и типу задает начальное значение объявленной переменной.

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу.
4. Отладить и выполнить программу.

Варианты заданий

1. Даны натуральные числа n, p, целые числа $a_1,\ldots,a_n$. Получить произведение членов последовательности $a_1,\ldots,a_n$ кратных p.

2. Даны действительные числа a, b, c. Удвоить эти числа, если $a \ge b \ge c$, и заменить их абсолютными значениями, если это не так.

3. Даны натуральное число n, целые числа $a_1,\ldots,a_n$. Найти количество и сумму тех членов данной последовательности, которые делятся на 5 и не делятся на 7.

4. Вычислить значение переменной t при следующем условии: при выполнении условия x-y < 0, то $t = \frac{(a+b)}{2} - a^2 + \frac{b^3}{a}$, в противном случае, если $x - y = 0$, то вычислить значение переменной t по формуле $t = \frac{a+b}{2}$, в противном случае $t = 3$.

$a = 3; b = 2; y = 4; x$ изменяется от 0 до 5.

5. Вычислить значение переменной

$a = x^y$, если $y > 0$ и $z > 0$,

$a = x^{-y}$, если $y < 0$ и $z < 0$,

$a = 0$, в остальных случаях.

x изменяется от 0 до 10; $y = 2; z = 2$.

6. Даны действительные числа a, b, c. Удвоить эти числа, если a >= b >= c, и заменить их абсолютными значениями, если это не так.

7. Вычислить функцию $Y = \frac{\cos^2 x - \sin(x+0.5) - x^3}{x+1} + \sqrt{\frac{\cos x}{x^2+0.1}}$ для $x = 0(0.1)1.5$

8. Вычислить функцию

$Y = \frac{x^2 + \sin x + 3\sqrt{\cos 2x}}{1.5 \ln x}$ для x = a(h)b. Значения a, b, h ввести

9. Вычислить значение функции

$Z = 2y_1^3 - 1.5y_3 + 0.7y_2$ где $y_1 = x - 0.5; y_2 = 3x^2 - 0.1; y_3 = 0.2x^2 - 0.3x$

Значение x ввести.

10. Вычислить значение функции

$Y = 1.5x \sin x^2 + \sqrt{\cos^2 x + 1} - \frac{24x^3 - \sqrt{x}}{\cos(x+0.2)} + e^{x+1}$ при $x = 0.9$

11. Вычислить значение $Y = x^2 + 0.5x + 0.2$ если x > 0.2

$Y = 1$ если x ≤ 0.2

Лабораторная работа №3

Операторы цикла и операторы передачи управления

Цель работы: Изучить синтаксис и работу операторов цикла и операторов передачи управления

Теоретические сведения

Оператор while

Обобщенная форма оператора **while**:

```
while(выражение)
    оператор
```

Вначале вычисляется выражение. Если результат отличен от нуля, тогда выполняется оператор и управление переходит обратно к началу цикла while. Это приводит к выполнению тела цикла while, а именно оператора, который будет выполняться до тех пор, пока выражение не станет равным нулю. Пример:

```
int i=1, sum=0;
while(i<=10)
{
    sum+=i;
    ++i;
}
```

Оператор for

Оператор for – итерационный оператор, обычно используемый с переменной, которая увеличивается или уменьшается. Конструкция оператора for следующая:

```
for(выражение1; выражение2; выражение3)
    оператор
```

Сначала вычисляется выражение1. Обычно выражение1 инициализирует переменную, используемую в цикле. Это выражение вычисляется только один раз. Затем вычисляется выражение2. Если оно отлично от нуля, то выполняется оператор, обрабатывается выражение3, проверяется выражение2 и т.д., пока выражение2 не станет равным нулю. В операторе for могут отсутствовать любое, либо все выражения, но должны оставаться точки с запятой. Пример:

```
int i, sum=0;
for(i=0; i<=10; i++)                // вычисление суммы 10 чисел
    sum+=i;

for(i=1, sum=0;  ; sum+=i++);   // бесконечный цикл
for(;;);
```

Оператор do_while

Конструкция оператора do_while следующая:

```
do
    оператор
while(выражение);
```

Сначала выполняется оператор, затем вычисляется выражение. Если его результат отличен от нуля, то управление переходит обратно к началу оператора do. Например: суммировать положительные числа.

```
int i=0, sum=0;
do
{
    sum+=i;
    scanf("%d",&i);
} while(i>0);
```

Операторы передачи управления

Оператор switch

В программе часто необходимо в зависимости от того или иного результата реализовать одну либо другую группу инструкций. Оператор switch позволяет выбрать одну из нескольких альтернатив. Он записывается в следующем формальном виде:

```
switch(целое выражение)
{
    case   метка1: вариант 1; break;
    case   метка2: вариант 2; break;
           . . .
    case   метка n: вариант n; break;
    default: вариант n+1; break;
}
```

Порядок работы оператора switch следующий:

1. Вычисляется выражение в круглых скобках, стоящих за switch.
2. Выполняется метка case, совпадающая с тем значением, которое было найдено на этапе 1; если ни одна из case не соответствуют этому значению, выполняется метка default; если метки default нет, switch прерывается.
3. Выполнение switch прерывается, когда встречается инструкция break или когда достигается конец switch.

Оператор break

Оператор break используется в операторах цикла for, while, do_while и в операторе switch. Оператор break вызывает выход из самого глубоко вложенного цикла или оператора switch. Например:

```
// выход из цикла по отрицательному значению
int i;
float x;
for(i=0; i < 10; i++)
```

```
{
    printf("\nВведите число\n");
    scanf("%f",&x);
    if( x< 0.0 )
    {
        printf("\nЧисло отрицательное\n");
        break;      //выход из цикла по отрицательному значению
    }
}
```

Оператор continue

Оператор continue заставляет прекратить текущую итерацию цикла и начать следующую.

```
// вычисление суммы положительных чисел
int i;
float x, sum=0;
for(i=0; i < 10; i++)
{
    printf("\nВведите число\n");
    scanf("%f",&x);
    if( x< 0.0 ) continue;
    sum+=x;
}
```

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу.
4. Отладить и выполнить программу.

Варианты заданий

1. Даны натуральное число n и целые числа $a_1,\ldots a_n$. Вычислить количество и сумму тех членов данной последовательности, которые делятся на 5 и не делятся на 7.

2. Используя оператор цикла, написать программу, в которой вычисляется наибольшее положительное целое число n, удовлетворяющее условию: $7n^3 + 81n^2 - 10^6 < 0$. Значение переменной n вывести на печать.

3. Даны натуральное число n и целые числа $a_1,\ldots a_n$. Вычислить количество и сумму положительных, отрицательных и равных нулю членов данной последовательности.

4. Получить все шестизначные счастливые номера. Про целое число n, удовлетворяющее условию 0 <= n <= 999999, говорят, что оно представляет собой счастливый номер, если сумма трех его первых цифр равна сумме трех

его последних цифр; если в числе меньше шести цифр, то недостающие начальные цифры считаются нулями.

5. Дано натуральное число n (n <= 99). Получить все способы выплаты суммы с помощью монет достоинством 1, 5, 10 и 20 коп.

6. Дано натуральное число n. Как наименьшим количеством монет можно выплатить n копеек? Предполагается, что в достаточно большом количестве имеются монеты в 1, 2, 3, 5, 10, 15, 20 и 50 коп.

7. День Учителя ежегодно отмечается в первое воскресенье октября. Дано натуральное число n, означающее номер года. Определить число, на которое в октябре указанного года приходится День Учителя.

8. Рассмотрим некоторое натуральное n (n > 1). Если оно четно, то разделим его на 2, иначе умножим на 3 и прибавим 1. Если полученное число не равно 1, то повторяется тоже действие и т.д., пока не получится 1. До настоящего времени неизвестно, завершается ли этот процесс для любого n > 1. Даны натуральные числа k, n, m (1 < k <n). Поверить, верно ли, что для любого натурального n из диапазона от k до n процесс завершается не позднее, чем после m таких действий.

9. Написать программу, которая выдает все способы представления числа n в виде суммы $n=b_1 + \ldots + b_k$, где k, $b_1,\ldots,b_k > 0$.

10. Даны натуральные числа a, b, c и a1, b1, c1, где a, a1 – означают день, b, b1 – месяц, c, c1 – год. Вычислить количество дней прошедших между двумя датами и количество полных лет.

Лабораторная работа №4

Работа с массивами. Одномерные массивы

Цель работы: Ознакомиться со структурой массивов. Понять, как объявлять одномерный массив и обращаться к отдельным элементам массива.

Теоретические сведения

Массив - это тип данных, который используется для представления последовательности однородных значений. Массив представляет собой группу элементов одного типа. Объявляется массив следующим образом:

```
int temp[20];
```

Квадратные скобки ([]) говоря о том, что temp – имя массива, а число, заключенное в скобки, указывает количество элементов массива. Нумерация элементов массива начинается с нуля, поэтому temp[0] является первым, а temp[19] последним элементом массива. Отдельный элемент массива определяется при помощи его номера или индекса. Элементы массива размещаются в памяти последовательно, друг за другом. Имя массива является указателем на первый элемент массива.

```
float mas[10];     // массив mas содержит 10 элементов типа float
int mas1[20];      // массив mas1 содержит 20 элементов типа int
int n=5;                    temp[0]                          temp[1]
int mm[n];         // ошибка, переменная не может задавать размер массива
```

При объявлении массивы можно инициализировать. Для этого при объявлении указывается список начальных значений элементов, заключенных в фигурные скобки:

```
int mas[5]={1,2,3,4,5};
```

Количество элементов в фигурных скобках не должно превышать размерность массива. Инициализировать можно не все элементы, а любое количество первых элементов:

```
int mas[10]={1,2,3,4,5,6};
```

Все остальные четыре элемента будут проинициализированы нулями.
Инициализировать массив можно следующим образом:

```
int mas[]={11,22,33,44,55};    // массив из пяти элементов
```

В этом случае число элементов массива определяется по списку инициализации.
Пример: Ввести массив чисел и вычислить сумму положительных значений

```
#include<stdio.h>
#include<conio.h>
#define N 100
void main(void)
{
    int mas[N];                    // массив чисел
    int summa=0;                   // сумма положительных значений
    int n;                         // количество чисел
    int i;
    printf("\nВведите количество чисел не более %3d",N-1);
    scanf("%d",&n);
    printf("\nВведите %3d элементов массива",n);
    for(i=0; i < n; i++)           // цикл по элементам массива
        scanf("%d", &mas[i]);      // ввод i-го элемента массива
    for(i=0; i < n; i++)
    {
        if(mas[i]>0)               // если i-ый элемент массива положительный
        summa+=mas[i];             // суммируются положительные элементы
    }
    printf("\n сумма положительных значений равна %3d",summa);
}
```

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу.
4. Отладить и выполнить программу.

Варианты заданий

1. Дано натуральное n (n >= 2). Найти все меньшие n простые числа используя решето Эратосфена. Решетом Эратосфена называют следующий способ. Выпишем подряд все целые числа от 2 до n. Первое простое число 2. Подчеркнем его, а все большие числа, кратные 2, зачеркнем. Первое из оставшихся чисел 3. Подчеркнем его как простое, а все большие числа, кратные 3, зачеркнем. Первое число из оставшихся теперь 5, так как 4 уже зачеркнуто. Подчеркнем его как простое, а все большие числа, кратные 5, зачеркнем и т.д.

2. Пусть дан массив $a_1,\ldots,a_n$. Требуется переставить $a_1,\ldots,a_n$ так, чтобы вначале в массиве шла группа элементов, больших того элемента, который в исходном массиве располагался на первом месте, затем – сам этот элемент, потом – группа элементов, меньших или равных ему.

3. Пусть по некоторому каналу связи передается сообщение, имеющее вид последовательности нулей и единиц. Из-за помех возможен ошибочный прием некоторых сигналов: нуль может быть воспринят как единица и наоборот. Можно передавать каждый сигнал трижды, заменяя, например, последовательность 1,0,1 последовательностью 1,1,1, 0,0,0, 1,1,1. Три последовательные цифры при расшифровке заменяются той цифрой, которая встречается среди них по крайней мере дважды. Такое утраивание сигналов существенно повышает вероятность правильного приема сообщения. Написать программу расшифровки.

4. Задан массив положительных чисел $a_1,\ldots,a_n$. Для каждого a[i] среди элементов массива следующих по порядку за a[i] и больших чем a[i], выберем элемент с наименьшим номером j и заменим значение a[i] на a[j]. Если такого элемента a[j] не найдется, то заменим значение a[i] нулем. Распечатать получившийся массив.

Пояснение. Например, массив 2,9,8,5,9,3,4,5,2 после замены станет таким 9,0,9,9,0,4,5,0,0.

5. Дан массив $a_1,\ldots,a_n$. Исключить из него пять минимальных элементов, сдвинув оставшиеся элементы к левому краю.

6. Даны два целочисленных массива $a_1,\ldots,a_n$ и $b_1,\ldots,b_n$. Вывести на печать все пары индексов, для которых a[i]* b[i] > 10. Подсчитать число пар и сумму этих произведений.

7. Дан одномерный массив $a_1,\ldots,a_n$. Найти и напечатать номер элемента, произведение которого с предыдущим максимально.

8. Рассортировать одномерный массив по возрастанию (убыванию) элементов (метод Шелла).

9. Рассортировать одномерный массив по возрастанию (убыванию) элементов (метод пузырька).

10. В одномерном массиве $a_1,\ldots,a_n$ заменить отрицательные элементы нулями, подсчитать число замен m, вычислить m!.

Лабораторная работа №5

Работа с массивами. Двухмерные массивы

Цель работы: Понять, как объявляются двухмерные массивы, как располагаются двухмерные массивы в памяти компьютера, как обращаться к элементам массива

Теоретические сведения

Двухмерный массив – это массив массивов, т.е. массив, элементами которого являются массивы.

Двухмерный массив объявляется следующим образом:

```
int mas[4][5];
```

Число в первых квадратных скобках указывает количество строк массива, а число во вторых квадратных скобках указывает количество столбцов массива. Для доступа к элементу двухмерного массива необходимо указать все его индексы:

```
rez= mas[1][2]; //переменной rez присваивается значение третьего элемента второй строки.
```

Элементы многомерных массивов располагаются в памяти компьютера построчно. Многомерные массивы можно инициализировать. Например:

```
int mas[3][3]={1,2,3,
               4,5,6,
               7,8,9};
```

Если необходимо проинициализировать не все элементы строки, то в списке инициализации можно использовать фигурные скобки, охватывающие значения для этой строки. Например:

```
int mas[3][3]={{1},
               {2,3},
               {4,5,6}
              };
```

Пример программы: отсортировать главную диагональ двухмерного массива по возрастанию

```
#include<stdio.h>
#include<conio.h>
#include N 4
int main()
{
    int i,j;
    int temp;
    int mas[N][N];                    // двухмерный массив
    printf("\nВведите элементы массива\n");
```

```
for(i=0; i < N; i++)                // цикл по строкам массива
  for(j=0; j < N; j++)              // цикл по столбцам массива
     scanf("%d",&mas[i][j]);    // ввод элемента массива
printf("\nИсходный массив\n");
for(i=0; i < N; i++)
{
  printf("\n");
  for(j=0; j < N; j++)
     printf("%4d", mas[i][j]);
}
for(j=N-1; j > 0; j--)
     for(i=0; i< j; i++)
          if(mas[i][i] > mas[i+1][i+1}
          {
               temp=mas[i][i];
               mas[i][i]=mas[i+1][i+1];
               mas[i+1][i+1]=temp;
          }
printf("\nРезультат\n");
for(i=0; i < N; i++)
{
  printf("\n");
  for(j=0; j < N; j++)
     printf("%4d", mas[i][j]);
}

}
```

В приведенном примере имя массива является постоянным указателем, этот указатель в программе изменить нельзя.

Порядок выполнения работы

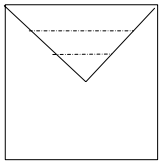
1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу. Размеры массивов брать как константу.
4. Отладить и выполнить программу.

Варианты заданий

1. Даны действительные числа $a_1,\ldots,a_n$. Получить квадратную матрицу порядка n следующего вида:

a_1 a_2 a_3 … a_{n-2} a_{n-1} a_n
a_2 a_3 a_4 … a_{n-1} a_n a_1
a_3 a_4 a_5 … a_n a_1 a_2
. . . . . .
a_n a_1 a_2 … a_{n-3} a_{n-2} a_{n-1}

2. Дана действительная квадратная матрица порядка n. Найти наибольшее из значений элементов, расположенных в заштрихованной части матрицы.



3. Дан двухмерный массив A размером n*m. Определить количество положительных, отрицательных и равных нулю элементов матрицы A.

4. Написать программу сортировки i-ой строки матрицы MM методом «пузырька». Исходную и преобразованную матрицу вывести на печать.

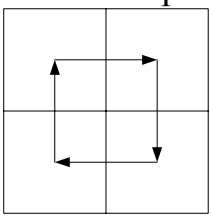
5. Отсортировать строки массива A размером n*m по убыванию.

6. В двухмерном массиве определить сумму элементов, имеющих нечетные значения и стоящих на четных местах.

7. В матрице целых чисел определить максимальный элемент на главной диагонали, и есть ли такой элемент ниже главной диагонали, если есть, то определить его координаты.

8. В двухмерном массиве определить количество элементов, кратных минимальному элементу массива.

9. Дана действительная квадратная матрица порядка 2n. Получить новую квадратную матрицу, переставляя ее блоки размера n*n следующим образом:



10. Дана квадратная матрица порядка n. Выполнить транспонирование матрицы относительно главной диагонали. Вывести на экран исходную и транспонированную матрицу.

Лабораторная работа №6

Указатели. Динамическое распределение памяти

Цель работы: Изучить операции работы с указателями и научиться использовать функции динамического выделения памяти для одномерных массивов.

Теоретические сведения

Указатели в С используются для связи переменных с машинными адресами. В программах указатели используются для доступа к памяти и манипуляций с адресами. Если v-переменная, то &v- это адрес или место в памяти, где хранится ее значение.
Объявляется указатель следующим образом:
type *name;
где type – тип, на который будет указывать указатель;
* - звездочка определяет тип “указатель”;
name – имя переменной.
Например: Объявление int *p; говорит о том, что переменная p будет иметь тип «указатель на целое». Если p-указатель, то *p – значение переменной, на которую указывает p.

```
int  i=5,j;
int *p;
p=&i;    //p указывает на i
j=*p;    // j=5
```

Адреса переменных можно использовать в качестве аргументов функций. В результате значения переменных могут изменяться в вызывающем окружении. Указатели используются в списке параметров для определения адресов переменных, значения которых могут изменяться. Например:

```
void order(int *, int *);
int main()
{
        int i=7, j=3;                   // инициализация переменных i и j
        printf(“\n i=%3d j=%3d”,i, j);
        order(&i,&j);                   // вызов функции order
        printf(“\n i=%3d j=%3d”,i, j);
}
void order(int *p, int *q)
{
        int temp;
        if(*p > *q)                     // перемена значений по адресам p и q
        {
            temp=*p;
            *p=*q;
            *q=temp;
        }
}
```

В языке С для обработки элементов массивов удобно использовать указатель на этот массив. Любой доступ к элементу массива может быть выполнен при помощи указателя. Например:

```
        int mas[10];   //  массив
        int *ptr;         // указатель на int
```

в результате присваивания

```
        ptr=&mas[0];
```

ptr будет указывать на нулевой элемент массива mas, иначе говоря, ptr будет содержать адрес элемента mas[0]. Если ptr указывает на некоторый элемент массива, то ptr+1 указывает на следующий элемент массива. Поскольку имя массива есть адрес

начального элемента массива, то присваивание ptr=&mas[0]; можно реализовать и так: ptr=mas;

Используя указатели, память под массивы можно отводить динамически, т.е. размещать в свободной памяти (free store). Свободная память – это предоставляемая системой область памяти для объектов, время жизни которых напрямую управляется программистом. В языке С функции для динамического выделения памяти объявлены в стандартной библиотеке в заголовочном файле stdlib.h – malloc(), calloc(), realloc(), free(); Функции динамического управления памятью делятся на функции динамического выделения и освобождения памяти. К функциям выделения памяти относятся malloc() и calloc(). Функция освобождения памяти – free().

Прототип функции
void *malloc(unsigned size); Эта функция выделяет область памяти размером size байт. Функция malloc возвращает указатель на начало выделенного блока памяти, если для выделения блока не хватает памяти, то возвращается NULL.

Прототип функции
void *calloc(unsigned num, unsigned size);
Функция calloc выделяет блок памяти и возвращает указатель на первый байт блока. Размер выделяемой памяти равен величине num*size, т.е. функция выделяет память, необходимую для хранения массива из num элементов по size байт каждый. Если память не выделена, то функция calloc возвращает NULL.

Прототип функции
void *realloc(void *ptr, unsigned size);
Эта функция изменяет размер динамически выделенной памяти, на которую указывает *ptr, на size(новый размер). Значение size задает новый размер блока. Если указатель не является значением, которое ранее было определено функциями malloc, calloc или realloc, то функция ведет себя неопределенно.
Прототип функции void free(void *ptr);
Функция освобождает область памяти, ранее выделенную при помощи функций malloc, calloc или realloc.

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу. Память под двухмерный массив необходимо выделять как для одномерного.
4. Отладить и выполнить программу.

Варианты заданий

1. В квадратной матрице порядка nнайти наибольший по модулю элемент. Получить квадратную матрицу порядка n-1 путем выбрасывания из исходной

матрицы какой-нибудь строки и столбца, на пересечении которых расположен элемент с найденным значением.

2. В двухмерном массиве среди чисел, стоящих на четных местах, определить минимальный положительный элемент массива и его индексы.

3. В матрице порядка n найти седловую точку (элемент максимальный в строке и минимальный в столбце).

4. Рассортировать положительные элементы каждой строки матрицы по убыванию. Отрицательные элементы оставить на своих местах.

5. Рассортировать отрицательные элементы каждого столбца матрицы по возрастанию. Положительные элементы оставить на своих местах.

6. Дана матрица. Поменять местами максимальный элемент среди всех отрицательных элементов матрицы на минимальный элемент среди всех положительных.

7. Рассортировать элементы главной диагонали квадратной матрицы порядка n по возрастанию.

8. В некоторых видах спортивных состязаний выступление каждого спортсмена независимо оценивается несколькими судьями, затем из всей совокупности оценок удаляется наиболее высокая и наиболее низкая, а для оставшихся оценок вычисляется среднее арифметическое, которое и идет в зачет спортсмену. Если наиболее высокую оценку выставило несколько судей, то из совокупности оценок удаляется только одна такая оценка; аналогично поступают с наиболее низкими оценками. Даны натуральное число n, действительные положительные числа $a_1,\ldots,a_n$ (n>=3). Считая, что числа $a_1,\ldots,a_n$ – это оценки, выставленные судьями одному из участников соревнований, определить оценку, которая пойдет в зачет этому спортсмену.

9. Даны действительные числа $a_1,\ldots,a_n$. Требуется умножить все члены последовательности $a_1,\ldots,a_n$ на квадрат ее наименьшего числа, если $a_1 >= 0$, и на квадрат ее наибольшего члена, если $a_1 < 0$.

10. У прилавка магазина выстроилась очередь из n покупателей. Время обслуживания продавцом i-ого покупателя равна t_i (i=1,…,n). Пусть даны натуральное n и действительные $t_1,\ldots,t_n$. Получить $c_1,\ldots,c_n$, где c_i – время пребывания i-ого покупателя в очереди (i=1,…,n). Указать номер покупателя, для обслуживания которого продавцу потребовалось самое малое время.

Лабораторная работа №7

Указатель на указатель для работы с многомерными массивами

Цель работы: Научиться использовать указатель на указатель при работе с двухмерными массивами.

Теоретические сведения

Можно объявлять переменные, имеющие тип «указатель на указатель». Например: int **mas; Указатель на указатель – это адрес ячейки, хранящий адрес указателя. При определении указатель на указатель можно инициализировать. Например:

```
int mm=10;              // переменная типа int
int *ptr=&mm;           //  указатель на переменную типа int
int **pptr=&ptr;        // указатель на указатель
```

Для доступа к переменной mm теперь можно использовать операции взятия по адресу и индексы: ptr[0], *ptr, pptr[0][0], **pptr.

Выделить память под двухмерный массив используя указатель на указатель можно следующим образом:

```
int **ptr;
int n;              // количество строк
int m;              // количество столбцов
printf("\n Введите количество строк и столбцов\n");
scanf("%d%d",&n,&m);
ptr=(int **)calloc(n,sizeof(int *));
for(int i=0; i < n; i++)
      ptr[i]=(int *)calloc(m,sizeof(int));
printf("\n Введите элементы массива\n");
for(int i=0; i < n; i++)
      for(int j=0; j < m; j++)
            scanf("%d",(ptr+i)+j);
printf("\n Исходный массив\n");
for(int i=0; i < n; i++)
{
      printf("\n");
      for(int j=0; j < m; j++)
            printf("%4d",ptr[i][j]);
}
```

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу. Память под массивы выделять динамически как указатель на указатель.
4. Отладить и выполнить программу.

Варианты заданий

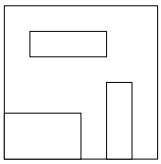
1. Рассортировать отрицательные элементы каждого столбца матрицы по возрастанию. Положительные элементы оставить на своих местах.

2. Даны натуральное число n, действительное число x, действительная матрица размера n*2n. Получить последовательность $b_1,\ldots,b_n$ из нулей и единиц, где $b_i=1$, если элементы i-ой строки матрицы не превосходят x, и $b_i=0$ в противном случае.

3. Даны целочисленная матрица размера n*m, целые числа k,k1 (1 <=k <=n, 1 <= k1 <=n, $k \neq k1$). Преобразовать матрицу так, чтобы строка с исходным номером k непосредственно следовала за строкой с исходным номером k1, сохранив порядок следования остальных строк.

4. Дана действительная квадратная матрица порядка n. Рассмотрим те элементы, которые расположены в строках, начинающихся с отрицательного элемента. Найти суммы этих элементов, которые расположены соответственно ниже, выше и на главной диагонали. Суммы найденных элементов хранить в массиве. Память под массивы выделять динамически.

5. На квадратном листе клетчатой бумаги размера n*n клеток нарисовано несколько прямоугольников. Различные прямоугольники не накладываются друг на друга и не соприкасаются. Определить число прямоугольников. Память под массив выделять динамически.



6. Таблица футбольного чемпионата задана квадратной матрицей порядка n, в которой все элементы, принадлежащие главной диагонали равны 0, а каждый элемент, не принадлежащий главной диагонали, равен 2, 1 или 0 (число очков набранных в игре: 2 – выигрыш, 1 – ничья, 0 – проигрыш).

а) найти число команд, имеющих больше побед, чем поражений;

б) определить номера команд, прошедших чемпионат без поражений;

в) выяснить, имеется ли хотя бы одна команда, выигравшая более половины игр. Память под массивы отводить динамически.

7. В действительной квадратной матрице порядка n найти наибольший по модулю элемент. Получить квадратную матрицу порядка n-1 путем выбрасывания из исходной матрицы какой-нибудь строки и столбца, на пересечении которых расположен элемент с найденным значением.

8. Даны действительные числа $a_1,\ldots a_n$, действительная квадратная матрица порядка n. Получить действительную матрицу размера n*(n+1), вставив в исходную матрицу между j и j+1 столбцами новый столбец с элементами $a_1,\ldots a_n$.

9. Латинским квадратом порядка n называется квадратная таблица размера n*n, каждая строка и каждый столбец которой содержит числа 1,2,…,n. Дана целочисленная квадратная матрица; определить, является ли она латинским квадратом.

10. Дана действительная квадратная матрица размера n. Расположить элементы матрицы следующим образом:

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

Лабораторная работа №8

Символьные строки

Цель работы: Освоить методику работы с символьными строками. Научиться использовать функции для работы со строками.

Теоретические сведения

В языке Си отсутствует специальный строковый тип. Строки в языке Си представляются как массив элементов типа char, в конце которого помещен символ ‘\0’ – признак конца строки. Такой массив называется строкой. При объявлении строки необходимо предусмотреть место для ‘\0’, которым должна заканчиваться строка.

Если объявить char str[10]; и инициализировать следующим образом:

```
for(int i=0; i < 10; i++)
{
    printf(“\nВведите символ”);
    scanf(“%c”,&str[i]);
}
```

то в выделенную область запишутся символы, и это будет массив символов. Если инициализировать так: gets(str);, то функция gets() в конец строки дописывает ‘\0’ – признак конца строки и это будет строка. Например:

```
char *ss;
int n;
printf(“\nВведите длину строки”);
scanf(“%d”, &n);                    // ввод длины строки
ss=(char *)calloc(n,sizeof(char));  // выделить память под строку
// ss=new char[n];
gets(ss);                           // ввод строки
```

Функции для работы со строками

Ввод строк

Прототип функции gets();

```
gets(char *);
```

Аргументом функции является указатель на строку, читает строку до тех пор, пока не встретит символ ввода ‘\n’. После считывания символа ‘\n’ превращает его в символ конца строки ‘\0’ и добавляет его в конец строки, поэтому в строке необходимо оставить место для символа’\0’.

Функция scanf() по формату %s читает до первого пустого символа в строке.

```
char str[10];
scanf(“%s”,str);
снег_снег     // будет прочитано все
снег снег     // будет прочитано только слово снег
```

Вывод строк

Функция puts(). Аргументом функции является указатель на строку. Функция puts() прекращает работу, когда встречает признак конца строки ‘\0’.

```
char str[10]=”Зима”;
puts(str);
```

Приведем прототипы некоторых, часто используемых, функций для работы со строками. Прототипы этих функций находятся в файле <string.h>

```
unsigned strlen(char *str); // вычисляет длину строки не включая признак конца
                            // строки ’\0’.
```

int strcmp(char *str1, char *str2);

Сравнивает строки str1 и str2. Результат отрицательный, если str1 < str2, равен нулю, если str1==str2, и положителен, если str1 > str2. Функция возвращает разницу между кодами ASCII первой пары несовпадающих символов.

char * strcpy(char *str1, char *str2); // копирует строку str2 в место памяти, на которое указывает str1;

char * strcat(char *srt1, char *str2); // присоединяет строку str2 в конец строки str1;

char * strstr(char *str1,const char *str2); // отыскивает первое вхождение строки str2 в строку str1.

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу. Память под строки выделять динамически.
4. Отладить и выполнить программу.

Варианты заданий

1. Даны натуральное число n, символы $s_1,\ldots,s_n$. Преобразовать последовательность $s_1,\ldots,s_n$, удалив каждый символ * и повторив каждый символ, отличный от *.

2. Даны натуральное число n и символы $s_1,\ldots,s_n$ среди которых есть двоеточие. Получить все символы, расположенные между первым и вторым

двоеточием. Если второго двоеточия нет, то получить все символы, расположенные после единственного имеющегося двоеточия.

3. Даны натуральное число n и символы $s_1,\ldots,s_n$. Группы символов, разделенные пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Найти длину самого короткого слова.

4. Даны натуральное число n и группы символов $s_1,\ldots,s_n$ и $a_1,\ldots,a_n$. Удалить из строки $s_1,\ldots,s_n$ все символы, которые принадлежат строке $a_1,\ldots,a_n$.

5. Даны натуральное число n и символы $s_1,\ldots,s_n$. Известно, что среди данных символов есть хотя бы один, отличный от пробела. Требуется преобразовать последовательность $s_1,\ldots,s_n$ следующим образом. Удалить группы пробелов, которыми начинается и которыми заканчивается последовательность, а также заменить каждую внутреннюю группу пробелов одним пробелом. Если указанных групп нет в данной последовательности, то оставить последовательность без изменения.

6. Преобразовать текст, состоящий только из прописных букв, в текст, состоящий из прописных и строчных букв. Первая буква после точки – прописная, остальные – строчные.

7. Даны натуральное число n и символы $s_1,\ldots,s_n$. Удалить из каждой группы идущих подряд цифр, в которой более двух цифр и которой предшествует точка, все цифры, начиная с третьей (например, ab+0.1973-1.1 преобразуется в ab+0.10-1.1).

8. Даны натуральное число n и символы $s_1,\ldots,s_n$. Группы символов, разделенные пробелами (одним или несколькими) и не содержащими пробелов внутри себя, будем называть словами:

а) подсчитать количество слов в данной последовательности;

б) преобразовать данную последовательность, заменяя всякое вхождение слова str1 длинной n, на слова str2 длинной m.

9. Даны натуральное число n и символы $s_1,\ldots,s_n$. Строку $s_1,\ldots,s_n$ обработать так, чтобы все слова, состоящие только из цифр, были удалены, их сумма стояла последним словом в строке.

10. Даны натуральное число n и символы $s_1,\ldots,s_n$. Среди символов этой строки особую роль играет знак #, появление которого означает отмену предыдущего символа строки; k знаков # отменяет k предыдущих символов. Преобразовать строку с учетом роли знака #. Например, строка «VR#Y##HELO#LO» должна быть «HELLO». Результирующую строку вывести на экран.

Лабораторная работа №9

Массивы символьных строк

Цель работы: Научиться объявлять и использовать массивы строк и функции для работы со строками.

Теоретические сведения

Массив символьных строк можно объявить следующим образом:

```
char **str;       // массив указателей
int n;            // количество строк
int m;            // количество символов в строке
// Выделить память под массив строк
printf("\n Введите количество  строк");
scanf("%d",&n);
str=(char **)calloc(n,sizeof(char *));          // выделить память под массив указателей
printf("\n Введите количество  символов в строке");
scanf("%d",&m);
for(i=0; i<n; i++)
        str[i]=(char *)calloc(m+1,sizeof(char));//выделить память под строку
// инициализировать массив строк
printf("\n Введите %3d строк  длинной не более %3d символов",n,m);
for(i=0; i<n; i++)
        gets(str[i]);
```

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Составить блок-схему алгоритма.
3. По разработанной блок-схеме алгоритма написать программу. Память под строки выделять динамически.
4. Отладить и выполнить программу.

Варианты заданий

1. Дан текст, состоящий из n предложений. Предложение представляет собой арифметическое выражение. Создать массив, включающий в себя идентификаторы из всех предложений.

2. Дан текст. Слова в тексте разделены пробелами и знаками препинания. Получить n наиболее часто встречающихся слов и число их появлений.

3. Дан текст. Строки текста содержат символьную и цифровую информацию. Слова могут состоять только из букв или только из цифр. Выполнить конкатенацию нецифровых слов, причем слова меньше четырех символов удалить. Найти сумму всех чисел в строке и записать ее в конец новой строки.

4. Дан массив строк. Строки являются изображением целых чисел. Рассортировать строки в порядке убывания их числовых значений.

5. Дан массив строк. Выделить слова из строк и записать их в массив. Необходимо рассортировать массив слов по их длинам.

6. Дан текст. Текст состоит из предложений, которые содержат символьную и цифровую информацию. В предложении записана фамилия, имя, отчество и зарплата по месяцам (не более двенадцати). Необходимо:

а) рассортировать строки в алфавитном порядке;

б) рассортировать строки по возрастанию общей суммы зарплаты.

7. Дан текст. Выделить все слова в предложениях, состоящих только из цифр. Определить сумму чисел во всех предложениях.

8. Дан массив строк. На месте исходного массива создать новый массив, такой, чтобы в конце была расположена четвертая часть первых по алфавиту строк. Последовательность расположения остальных строк в массиве сохранить.

9. Дан текст, каждый символ которого может быть малой буквой, цифрой или одним из знаков +, -, *. Группой букв будем называть такую совокупность последовательно расположенных букв, которой непосредственно не предшествует и за которой непосредственно не следует буква. Аналогично определим группу цифр и группу знаков. Выяснить, верно ли что, в данном тексте больше групп букв, чем групп знаков.

10. Дано n1 матриц. Размер матрицы n*m. Каждый элемент матрицы – строка знаков длинной k. Слова в предложении разделены одним или несколькими пробелами. Определить в каждом предложении количество слов, которые справа и слева читаются одинаково и записать их в массив.

Лабораторная работа №10

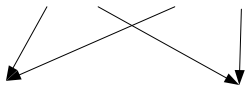
Функции

Цель работы: Научиться создавать и использовать новые функции пользователя

Теоретические сведения

Программа на С составляется из одной или более функций, одна из которых main(), она называется головной функцией. Выполнение программы всегда начинается с функции main().

Функция – самостоятельная единица программы, спроектированная для реализации конкретной задачи. Что нам требуется знать о функции? Необходимо знать, как функцию необходимо определить, т.е. написать код функции, как к функции обращаться, т.е. вызвать функцию на выполнение и как устанавливать связи между функцией и программой ее вызывающей. Чтобы установить связь между функцией и программой ее вызывающей, необходимо знать прототип функции. Прототип функции – это объявление функции. Прототип функции имеет следующую форму:



Например: float s_z(int a, int b);

Список аргументов может быть пустым, содержать один аргумент или несколько аргументов, разделенных запятыми. Если функция не имеет аргументов, допускается использования ключевого слова void. Если функция не имеет аргументов и она ничего не возвращает, то ее прототип можно записать в следующем виде:

void prim(void);

Вызов функции

Вызов функции осуществляется по ее имени. Для вышеописанной функции s_z(), ее надо вызвать на выполнение так:

```
int a=10, b=20;      // объявление и инициализация переменных
float rez;
rez=s_z(a,b);        // вызов функции на выполнение
```

Определение функции

В С код, описывающий, что делает функция, называется определением функции. Формально это выглядит так:

```
тип_возв_рез имя_ф(тип имя_n1, тип имя_n2, …)
{
    тело функции
}
```

Синтаксически аргументы – это идентификаторы, они могут использоваться внутри тела функции. Иногда параметры в определении функции называют формальными параметрами. Формальные параметры – это то, вместо чего будут подставлены фактические значения, передаваемые функции в момент ее вызова. После вызова функции значение аргумента, соответствующее формальному параметру, используется в теле выполняемой функции. В С такие параметры являются передаваемыми по значению. Когда применяется вызов по значению, переменные передаются функции как аргументы, их значения копируются в соответствующие параметры функции, а сами переменные не изменяются в вызывающем окружении.

Инструкция return

Инструкция return используется для двух целей. Когда она выполняется, управление программой немедленно передается обратно в вызывающее окружение. Кроме того, если за ключевым словом return следует какое-либо выражение, то его значение также передается в вызывающее окружение. Инструкция return имеет следующие формы записи:

```
return;
return выражение;
return (выражение);
```

Структура программы, содержащей несколько функций пользователя:

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

```
        объявление переменных
        ввод данных
        вызов функций на выполнение
}
```

Например: написать функцию, которая вычисляет среднее значение двух целых чисел

```
#include <stdio.h>
void main(void)
{
        float s_z(int a, int b);   // прототип функции
        int a,b;                   // переменные
        float rez;                 // результат
        printf("\nВведите два целых числа\n";
        scanf("%d%d",&a,&b);
        rez=s_z(a,b);              // вызов функции на выполнение
        printf("\nСреднее значение равно %3d",rez);
}
float s_z(int a, int b)
{
        float r;
        r=((float)a+b)/2;       // или return (((float)a+b)/2);
        return r;
}
```

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Программу разбить на функции. Составить блок-схему алгоритма каждой функции.
3. По разработанной блок-схеме алгоритма написать программу. Память под данные выделять динамически.
4. Отладить и выполнить программу.

Варианты заданий

1. Составить функцию, позволяющую определить позицию первого вхождения в заданную строку какого-либо символа из второй заданной строки. Результатом работы функции должна быть -1, если первая строка не содержит ни одного символа, принадлежащего и второй строке.

2. В головной функции ввести текст. В первой функции подсчитать количество слов в тексте. Во второй – подсчитать количество слов, у которых первый и последний символы совпадают.

3. Дан текст. Слова в предложениях разделены одним или несколькими пробелами. В головной функции ввести текст. Во второй функции определить в

каждом предложении количество слов, которые слева и справа читаются одинаково.

4. Дан текст, состоящих из n предложений. Слова в предложениях разделены одним или несколькими пробелами. В головной функции ввести текст. Во второй функции записать без повторения в новый массив слова, встречающиеся в тексте более одного раза.

5. Даны действительная матрица размером n*(n+1), действительные числа $a_1,\ldots a_{n+1}$, $b1,\ldots b_{n+1}$, натуральные числа p, q (p<=n, q<=n+1). Образовать новую матрицу размером (n+1)*(n+2) вставкой после строки с номером p данной матрицы новой строки с элементами $a_1,\ldots a_{n+1}$, и последующей вставкой после столбца с номером q нового столбца с элементами $b1,\ldots b_{n+1}$. Память под массивы отводить динамически. Программу разбить на функции.

6. Дана действительная квадратная матрица порядка n. Построить последовательность действительных чисел $a_1,\ldots a_n$ по правилу: если в i-ой строке матрицы элемент, принадлежащий главной диагонали, отрицателен, то a_i равно сумме элементов i-ой строки, предшествующих первому отрицательному элементу, иначе a_i равно сумме последних элементов i-ой строки, начиная по порядку неотрицательного элемента. Память под массивы отводить динамически. Программу разбить на функции.

7. Даны две квадратные матрицы порядка n. Получить новую матрицу:

а) умножением элементов каждой строки первой матрицы на наибольшее из значений элементов соответствующей строки второй матрицы;

б) прибавлением к элементам каждого столбца новой матрицы, произведения элементов соответствующих строк второй матрицы.

Память под матрицы отводить динамически. Программу разбить на функции.

8. Назовем допустимым преобразованием матрицы перестановку двух строк и двух столбцов. Дана действительная квадратная матрица порядка n. С помощью допустимых преобразований добиться того, чтобы один из элементов матрицы, обладающий наименьшим значением, располагался в левом нижнем углу матрицы. Память под матрицу выделять динамически. Программу разбить на функции.

9. Обработать n1 матриц. Размеры матрицы n*m. Элементы матрицы – строка символов. В строке есть слово, состоящее только из цифр. Найти это слово, преобразовать в число и сохранить в массиве. Полученный массив вывести на экран. Ввод матриц, поиск слова, преобразование слова в число реализовывать в отдельных функциях. Память под матрицы отводить динамически.

10. В головном модуле объявить n матриц. Элементами матрицы являются строки знаков. Все строки и столбцы матрицы, в которых хотя бы один из ее элементов совпадают со строкой введенной из клавиатуры, удалить. Полученную матрицу уплотнить. Память под матрицы выделять динамически. Программу разбить на функции.

Лабораторная работа №11

Указатели на функции. Рекурсия.

Цель работы: Научиться использовать в программах указатели на функции

Теоретические сведения

Сама функция не может быть значением переменной, но можно определить указатель на функцию и работать с ним как с обычной переменной: присваивать ей значения, размещать в массиве, передавать в качестве параметра в другую функцию, возвращать как значение из функции, а также вызывать функцию через указатель на нее. Указатель на функцию, это такая переменная, которой можно присваивать адрес точки входа в функцию, т.е. адрес первой исполняемой команды. Используя массив указателей на функции можно организовывать меню в программа.

Общий формат объявления указателя на функцию имеет вид:

тип_результата (*имя_указателя_на_функцию)(список аргументов);

Например: float (*fun)(int a, int b);

где fun – указатель на функцию, возвращающую результат типа float и принимающую два параметра типа int.

Пусть имеется прототип функции

int prim(int);

и указатель на эту функцию

int (*ff)(int);

Тогда оператор ff=prim; присвоит указателю ff адрес входа в функцию prim. После этого вызвать функцию prim на выполнение можно следующими способами:

```
prim(x);        // вызов функции, используя ее имя
(*ff)(x);       // вызов функции через указатель
ff(x);          // вызов функции также через указатель
```

Рекурсия

Рекурсивный способ реализации функции – это когда функция вызывает саму себя. При работе с рекурсивной функцией необходимо выполнять следующие условия:

- при каждом вызове функции, в вызываемую функцию должны передаваться измененные данные;
- используя операторы if и return, следует предусмотреть возможность завершения рекурсивной функции в соответствии с условием задачи.

Пример: заданы два числа a и b. Необходимо большее из этих чисел разделить на меньшее.

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

```
#include<conio.h>
void main(void)
{
      float a,b;
      float div(float a, float b);                  // прототип функции
      printf(“\nВведите значения чисел a и b\n”);
      scanf(“%f%f”,&a,&b);                          // ввод значений a и b
      printf(“\nрезультат %.3f”,div(a,b));          // вызов функции
}
float div(float a, float b)                         // рекурсивная функция
{
      if(a<b)
            return div(b,a);                        // рекурсивный вызов функции
      else  return a/b;                             // выход из функции
}
```

Порядок выполнения работы

1. Изучить краткие теоретические сведения.
2. Разработать алгоритм функции с переменным числом параметров.
3. По разработанному алгоритму написать программу.
4. Отладить и выполнить программу.

Варианты заданий

1. Дана матрица M размером n*m. Элементами матрицы являются слова. Расположить элементы главной диагонали по возрастанию, а элементы побочной диагонали по убыванию. Память под матрицу отводить динамически. Использовать указатели на функции.

2. Дан массив строк. Удалить из массива первую и последнюю строку по алфавиту. Полученный массив уплотнить. Память под массив строк отводить динамически. Использовать указатели на функции.

3. Дано n1 матриц, размер матрицы n*m. Рассортировать столбцы каждой матрицы следующим образом: первый столбец - по возрастанию, второй – по убыванию и т.д. Память под матрицы отводить динамически. Использовать указатели на функции.

4. Дан массив строк. Найти строку, содержащую слово с наибольшим числом символов. Память под массив строк отводить динамически. Использовать указатели на функции.

5. Дан массив строк. Необходимо сформировать массив указателей на первые по алфавиту n слов из этих предложений. Память под массивы отводить динамически. Использовать указатели на функции.

6. Дан текст. Слова в тексте разделены одним или несколькими пробелами. Записать без повторения в новый массив слова, встречающиеся в тексте более одного раза. Используя рекурсию, каждое слово в массиве

записать в обратном порядке, и затем массив рассортировать в алфавитном порядке. Память под массивы отводить динамически. Использовать указатели на функции.

7. Дано n предложений, представляющих собой арифметические выражения. Создать массив, включающий в себя идентификаторы из всех выражений. Используя рекурсию, записать идентификаторы в обратном порядке и рассортировать их в алфавитном порядке. Память под текст отводить динамически. Использовать указатели на функции.

8. Ввести значение n и вычислить n!.

9. Не объявляя массива ввести группу данных и вывести их в обратном порядке.

10. Ввести арифметическое выражение и используя рекурсивную функцию, проверить в выражении правильность расстановки скобок любого вида.

Литература

1. Уэйт, М., Прата, С., Мартин, Д. Язык Си. Руководство для начинающих. – М.:Мир, 1988. – 512 с.

2. Керниган, Б., Ритчи, Д. Язык программирования Си. –М.:Финансы и статистика, 1992. – 272 с.

3. Юлин, В.А., Булатова, И.В. Приглашение к Си. – Мн.: Выш. Шк., 1991.

4. Подбельский, В.В., Фомин, С.С. Программирование на языке Си. – М.: Финансы и статистика, 2000. – 600 с.

5. Демидович, Е.М. Основы алгоритмизации и программирования. Язык Си. – Бестпринт, 2001. – 411 с.

Содержание

Св. план 2007, поз.

Учебное издание

Ковальчук Анна Михайловна
Луцик Юрий Александрович

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по дисциплине «Основы алгоритмизации и программирования» для студентов специальности I-40 02 01 «Вычислительные машины, системы и сети»

В 2-ч частях

Часть 1

Редактор
Корректор

Подписано в печать	Формат	Бумага офсетная.
Гарнитура «Таймс»	Печать ризографическая	Усл. печ. л.
Уч.-изд.л.	Тираж	Заказ

Издатель и полиграфическое исполнение: Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛП №02330/0131666 от 30.04.2004.
220013, Минск, П. Бровки, 6