КАЗАНСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ФИЗИКИ

Кафедра радиоастрономии

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

ЧИСЛЕННЫЕ МЕТОДЫ НА ЯЗЫКЕ СИ

Учебно-методическое пособие

Казань – 2019

УДК 681.924

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

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

Рецензент:
кандидат физико-математических наук,
доцент кафедры радиофизики КФУ **Р.Р. Латыпов**

Зыков Е.Ю., Колчев А.А., Хуторова О.Г., Шерстюков О.Н. Численные методы на языке Си / Е.Ю.Зыков, А.А. Колчев, О.Г. Хуторова, О.Н. Шерстюков. – Казань: Казан. ун-т, 2019. – 71 с.

Данное учебно-методическое пособие предназначено для проведения практических занятий по дисциплинам, связанным с введением в информационные технологии, обязательным к изучению в Институте физики КФУ. Пособие содержит базовый теоретический материал по методам численного решения некоторых типовых задач, сведения по основам программирования на языке Си и практические задания для самостоятельной реализации.

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

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данное пособие посвящено вопросам решения некоторых математических задач численными методами и особенностям программирования соответствующих алгоритмов на языке Си. Оно предназначено для методического обеспечения проведения практических занятий по дисциплинам, связанным с введением в информационные технологии, преподаваемым на 1-2 курсах Института Физики КФУ.

Пособие включает в себя численные методы нахождения корней алгебраического уравнения, вычисления определенного интеграла, решения систем линейных уравнений, разложения функции в ряд. Дополнительно включены разделы, связанные с работой с числовыми массивами. Кроме того, пособие включает в себя справочный материал по базовым операторам и структурам языка Си.

Теоретический материал сопровождается большим количеством примеров практической реализации соответствующих алгоритмов.

1. ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА СИ

Программы пишутся для того, чтобы решить какую-то конкретную задачу. Информация, необходимая для решения, записывается в оперативную память компьютера в виде переменных и констант. Переменные и константы, а также функции и файлы имеют свои уникальные имена (идентификаторы).

В языке Си идентификаторы должны начинаться с буквы или с символа подчеркивания (_), состоять из строчных или прописных букв (a...z, A...Z), цифр (0..9) и символа подчеркивания.

Пример.

```
Alpha, alpha, _alpha1, alpa2_2      /* несколько различных идентификаторов
```

Идентификаторы для новых объектов не должны совпадать с ключевыми словами языка и именами стандартных функций из библиотек.

Переменные и их типы

Переменная в языке Си – это именованная область памяти, в которой содержится определенное значение.

Синтаксис объявления переменной:

тип имя_переменной (идентификатор**)** [**= значение**];

Тип характеризует размер выделяемой памяти, диапазон и точность представления числовых значений; **значение** (необязательный параметр): можно сразу присвоить (инициализировать) определенное значение переменной, в противном случае значение не определено.

В таблице 1 выделены некоторые основные типы языка Си.

Таблица 1.

Тип	**Характеристика**	**Размер ячейки памяти**	**Диапазон значений**
char	Символьный тип, значениями являются различные символы из кодовой таблицы ('F', ':', 'j'); при записи в программе они заключаются в одинарные кавычки	1 байт	от 0 до 256
int	Переменная целого типа (-5, 21), размер соответствует размеру целого машинного слова	обычно 4 байта	от -2147483648 до 2147483647
unsigned int	Переменная целого типа без знака, принимает только положительные значения	обычно 4 байта	от 0 до 4294967295
float	Определяет переменные вещественного типа. Их значения имеют дробную часть, отделяемую точкой, например: -5.6, 31.28 и т.п. Вещественные числа могут быть записаны также в форме с плавающей точкой, например: -1.09e+4. 7 значащих цифр	4 байта	от 10^{-38} до 10^{+38}
double	Позволяет определить вещественную переменную двойной точности. 15 значащих цифр.	8 байт	от 10^{-308} до 10^{+308}
void	Ключевое слово void (не имеющий значения) используется для нейтрализации значения объекта, например, для объявления функции, не возвращающей никаких значений.	1 байт	

Пример.

```
int a1,b2,c11;          /* Объявляем три переменные типа int */
float F10, d = 2.01;    /* Объявляем одну переменную без инициализации, а
                           вторую инициализируем значением 2.01*/
```

Арифметические операции

В Си определены следующие арифметические операции:

Операция	**Запись**
Сложение	a + b
Вычитание	a - b
Деление	a / b
Умножение	a * b
Остаток от деления целых чисел	a % b
Инкремент (увеличение числа на единицу)	i++ (i = i + 1; эквивалентно инкременту)
Декремент (уменьшение числа на единицу)	i--; (i = i - 1; эквивалентно декременту)

Операции сравнения

В компьютер изначально заложена булева (двоичная) логика. В логических операциях 0 – это FALSE, а 1 – это TRUE. Эти понятия TRUE и FALSE тесно связанны с **операциями сравнения**:

Операция	**Назначение**	**Пример**
= =	равенство	a = = b
!=	не равно	a != b
>	больше	a > b
<	меньше	a < b
<=	меньше или равно	a <= b
>=	больше или равно	a >= b

Стандартные математические функции

Стандартные математические функции хранятся в библиотеке math.h . Аргумент тригонометрической функции задается в радианах. Все функции принимают один аргумент типа double. Возвращают также число типа double.

Таблица 2.

Форма записи различных математических функций в языке Си

Математическая функция	**Название**	**Функция на языке Си**
arccos(x)	Арккосинус	`acos(x)`
arcsin(x)	Арксинус	`asin(x)`
arctan (x)	Арктангенс	`atan(x)`
ceil(x)	Округление в большую сторону	`ceil(x)`
cos(x)	Косинус	`cos(x)`
exp(x)	*e* в степени *x*	`exp(x)`
|x|	Модуль	`fabs(x)`
floor(x)	Округление в меньшую сторону	`floor(x)`
mod(x, y)	Остаток от деления x на y	`fmod(x, y)`
ln(x)	Натуральный логарифм	`log(x)`
lg(x)	Десятичный логарифм	`log10(x)`
x^y	Возведение основания ***x*** в степень ***y***	`pow(x, y)`
sin(x)	Синус	`sin(x)`
$\sqrt{x}$	Квадратный корень	`sqrt(x)`
tg(x)	Тангенс	`tan(x)`

Примечание.

Аргументы тригонометрических функций всегда в радианах! Исходные значения в градусах перед подстановкой в тригонометрическую функцию необходимо перевести в радианы.

Функция printf в Си

Функция `printf( )` используется для вывода информации.

Для вывода значений на экран нужно в функции написать правильный **спецификатор формата**. Ниже представлены некоторые спецификаторы формата для целых и вещественных чисел:

Спецификатор	Назначение
%d	для вывода целых чисел в десятичной форме
%u	для вывода целых чисел в десятичной форме без знака
%f	для вывода вещественных чисел в обычной форме
%e	для вывода вещественных чисел в экспоненциальной форме

Пример.

```
int a = 24;
float b = 24;
printf ("%d %f %e", a, a, a); // Вывод на экран: 24 24.000000 2.400000e+01
```

Кроме того, при выводе вещественных чисел после написания знака (%) можно указать количество выводимых знаков вещественной составляющей числа.

Пример.

```
double a = 24;
printf ("%.3f %.1e",a,a); // Вывод на экран: 24.000 2.4e+01
```

Существует набор управляющих последовательностей:

1. '\n' перевод на новую строку
2. '\t' табуляция

Пример.

```
int a = 24;
float b = 24;
printf ("%d\n %f\t %e", a, a, a);
// Вывод на экран:
24
24.000000       2.400000e+01
```

Примеры программ

Программа, написанная на языке Си, состоит из инструкций для компьютера (операторов, функций, директив препроцессору).

Рассмотрим простую программу на языке Си, текст которой нужен нам для выявления некоторых основных черт любой программы, написанной на языке Си.

```
#include <stdio.h>
int main( )
{
    printf ("Hello!");
    return 0;
}
```

Эта программа выводит на экран компьютера приветствие "Hello!". Разберем текст программы построчно.

В начале программы используется так называемая директива препроцессору:

`#include` – требует включения другого файла.

Вся первая строка — `#include <stdio.h>` — означает «включить заголовочный файл «stdio.h». В этом файле объявляются функции библиотеки

стандартного ввода-вывода, например, средства взаимодействия программы с вашим терминалом. Буква «h» после точки означает заголовок (англ. header). В заголовочных файлах объявляются предоставляемые соответствующими им библиотеками функции, типы данных, константы и определения препроцессора.

В следующей строке идёт определение функции **main**. Оно начинается с объявления:

`int main( )`, что означает – функция с именем **main**, которая возвращает целое число (число типа **int** от англ. integer). Программа, написанная на языке Си, всегда начинает выполняться с функции, называемой **main( ).** Круглые скобки указывают на то, что **main** – имя функции.

Открывающая фигурная скобка `{` отмечает начало последовательности операторов, образующих тело (или определение) функции. Конец определения отмечается закрывающей фигурной скобкой }.

`printf("Hello!");` – оператор вывода на печать. С его помощью выводится на экран выражение, заключенное между двойными кавычками `"Hello!"`. Функция `printf()` является универсальной функцией форматного вывода. Ее прототип содержится в заголовочном файле `stdio.h`. Точка с запятой в конце строки указывает на то, что в ней содержится оператор языка Си. Все операторы в языке С заканчиваются символом `;` “точка с запятой”.

`return 0;` – заставляет функцию (в данном случае `main()`) прекратить свое выполнение и передать 0 в вызывающий системный процесс. В Си при нормальном завершении функции возвращается нулевое значение (и ненулевое — при аварийном и др.).

Рассмотрим следующую программу, в которой находится значение выражения $2a+2^b$ и результат выводится на экран.

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

```
int main()
{
    float a, b, c;
    a = 3; b = 5;
    c = 2 * a + pow(2, b);
    printf ("c= %f \n", c);
    return 0;
}
```

Вторую, третью, четвертую и последние две строки программы мы рассмотрели ранее. Рассмотрим отличающиеся строки программы.

Строка `#include <math.h>` подключает библиотеку математических функций.

Строка `float a, b, c;` объявляет переменные `a, b` и `c` и сообщает компилятору, что это переменные вещественного типа. В языке Си все переменные должны быть объявлены прежде, чем они будут использованы. Процесс объявления переменных включает в себя определение имени и указание типа переменных.

В строке `a = 3; b = 5;` переменным `a` и `b` присваиваются значения 3 и 5. В языке Си в операторе присваивания используется просто знак равенства = .

`c = 2 * a + pow (2, b);` – переменной `c` присваивается значение выражения `2a+2b` (заканчивается точкой с запятой).

Вызов стандартной функции `printf()` в строке `printf ("c= %f \n", c);` отличается от предыдущего примера. Здесь строка состоит из двух частей: имени функции `printf()` и двух ее аргументов `"c= %f \n"` и `c`, разделенных запятой. Первый аргумент функции `printf()` – это строка в кавычках `"c= %f \n"`, которую называют управляющей строкой. Эта строка может содержать любые символы или спецификации формата, начинающиеся

с символа %. Обычные символы просто отображаются на экран в том порядке, в котором они следуют. Спецификация формата, начинающегося с символа %f, указывает формат, в котором будет выводиться значение переменной c, являющейся вторым аргументом функции printf(). Комбинация символов '\n' сообщает функции printf() о необходимости перехода на новую строку.

Базовые алгоритмические структуры языка Си

В соответствии с теоремой о структуировании известно, что любой сколь угодно сложный алгоритм можно реализовать с помощью трех базовых алгоритмических структур: следование, ветвление и цикл. Эти три типа базовых алгоритмических структур имеют операторную поддержку во всех языках программирования. Они позволяют создавать однократно и линейно выполняемые участки программы, участки программы с развлетвлением программного кода в зависимости от условия и участок программы, повторяемый некоторое количество раз.

Рассмотрим основные реализации этих структур в Си.

Следование

Следование – это самая простая и понятная алгоритмическая структура. В ней операторы располагаются в необходимой для выполнения последовательности и выполняются один раз.

Рис. 1.1. Блок-схема структуры следование

Пример:

```
a = a + b;
c = 15;
d = d + c;
```

Условный оператор (оператор ветвления)

Первый тип постановки условия (простое условие) может быть изображен в виде блок-схемы, как показано на Рис.1.2. Если условие верно, то выполняется блок из одного или более операторов, в противном случае они пропускаются и указатель команд переходит к оператору, расположенному после условного оператора. Синтаксис на языке Си выглядит следующим образом:

Рис. 1.2. Блок-схема условного оператора

```
if (условие) {блок из одного или более операторов}
```

Пример:
```
if (a > 0)
    {
        c++;
        b=b+a;
    }
```

Второй тип – это постановка полного условия с альтернативными операторами. Блок-схема такого оператора изображена на Рис. 1.3. В этом случае некоторый набор действий выполняется в случае истинности условия, и некоторый другой набор действий в случае его ложности. Если условие верно, то выполняется блок 1 из одного или более операторов, а если не верно, то блок 2.

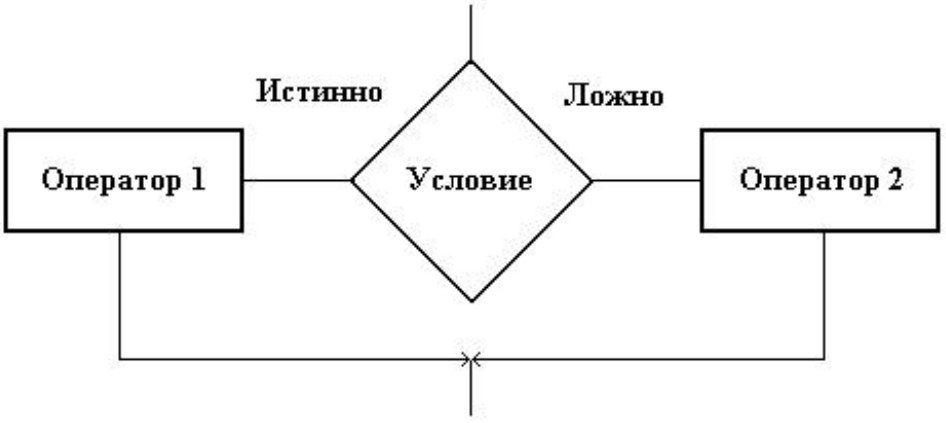

Рис. 1.3. Блок-схема оператора ветвления

```
if (условие)
{
    блок 1 из одного или более операторов
}
```

```
else
{
    блок 2 из одного или более операторов
}
```

Пример:

```
if (a > 0)
{
    c++;
    b=b+a;
}
else
{
    c--;
    a = 0;
}
```

Циклические алгоритмические структуры

Циклические структуры используются в участках программы, которые необходимо повторить некоторое известное количество раз или повторять до выполнения условия. Обязательным условием при организации цикла является то, что в теле цикла необходимо разместить оператор, изменяющий значение переменной, указанной в условии выхода из цикла. В циклах с параметром изменение переменной указывается сразу в заголовке цикла. В большинстве языков программирования циклические операторы бывают трех видов: цикл с предусловием, цикл с постусловием и цикл с параметром. Цикл с параметром можно считать частным случаем цикла с предусловием.

В цикле с предусловием оператор начала цикла и условие выхода из цикла размещают в начале этой алгоритмической структуры. Конец цикла определяется местом размещения закрывающей фигурной скобки. Если фигурные скобки не используются, то в цикле учавствует только один оператор, который размещен сразу после оператора начала цикла. Цикл данного типа – это

цикл с неизвестным числом повторений. Если условие истинно, то выполняется блок из одного или нескольких операторов и происходит возврат на проверку условия. Если условие ложно, происходит выход из цикла.

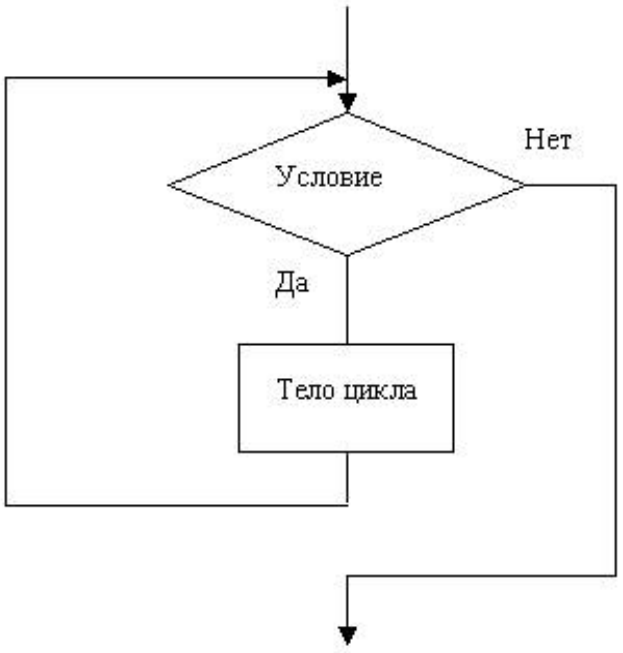

Рис. 1.4. Блок-схема цикла с предусловием

```
while (условие)
{
    блок из одного или нескольких операторов
}
```

Пример:
```
while (a < 3.14)
        {
            a+=0.01;
            b=b+a;
        }
```

В цикле с постусловием условие выхода из цикла размещается в конце алгоритмической структуры. Конец цикла определяется местом размещения

оператора с условием. Поскольку условие проверяется в конце цикла, то тело цикла выполняется как минимум один раз.

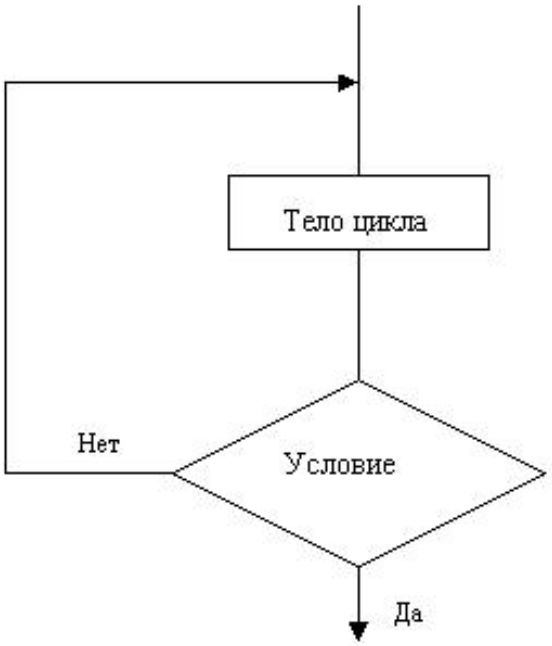

Рис. 1.5. Блок-схема цикла с постусловием

```
do
{
    блок из одного или нескольких операторов
}
while (условие);
```

Пример:

```
do
     {
          a+=0.01;
          b=b+a;
     }
while (a < 3.14);
```

Цикл с параметром (со счетчиком) обычно используется в случаях, когда заранее известно число повторений тела цикла. Конец цикла определяется местом размещения закрывающей фигурной скобки. Блок-схема данного типа цикла совпадает с таковой для цикла с предусловием.

```
for (начальное выражение; контрольное выражение; счетчик)
{
    блок из одного или нескольких операторов
}
```

Пример:

```
for (i = 0; i >= b; i++)
     {
         a+=0.01;
         b=b+a;
     }
```

В качестве начального выражения можно использовать несколько операторов, их нужно разделять запятыми. Например:

```
for (i=a, s=0; i>=b; i++) s+=i;
```

2. ТАБУЛЯЦИЯ ФУНКЦИИ

Цель данной работы – научиться работать с основными синтаксическими конструкциями языка Си, организовывать циклы, выполнять математические вычисления.

Под табуляцией функции понимается вычисление значений функции в зависимости от аргумента, который меняется в определенных пределах с постоянным шагом. Решение задачи табуляции является достаточно характерным примером реализации циклического алгоритма.

Формулировка задачи табуляции: дана функция $y = f(x)$. Требуется получить значения $f(x)$ для аргумента x, меняющегося в интервале от x_{min} до x_{max} с шагом Δx. На рис. 2.1 приведен алгоритм решения задачи табуляции, использующий цикл с предусловием. Число повторений цикла, необходимое для решения поставленной задачи, можно определить по формуле:

$$n = \left[\frac{x_{\max} - x_{\min}}{\Delta x}\right] + 1,$$

в которой квадратные скобки означают функцию взятия целой части от заключённого в них выражения.

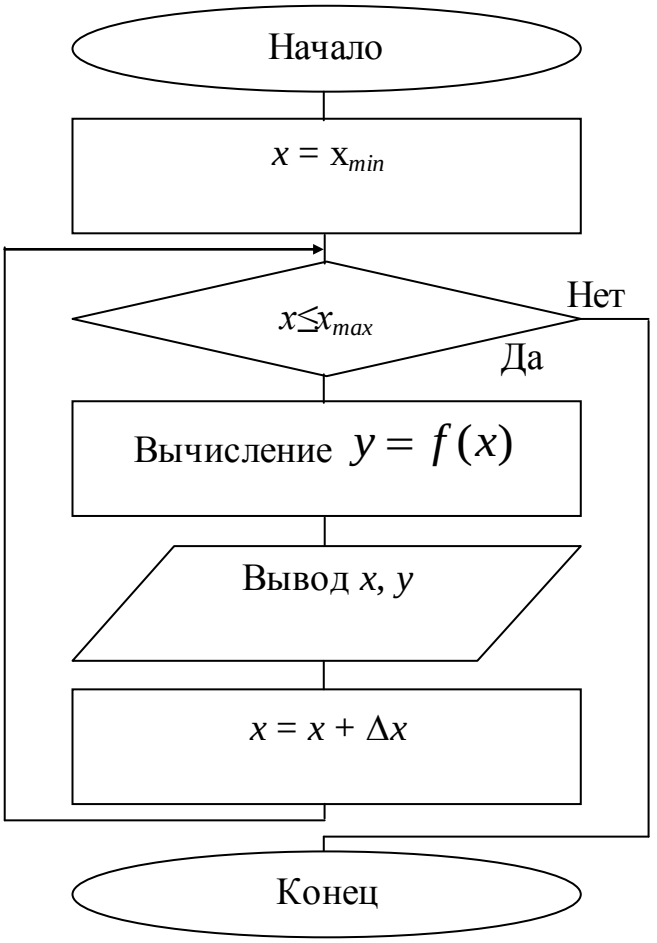

Рис.2.1. Блок-схема реализации задачи табуляции функции

При выполнении заданий необходимо учесть, что алгоритмы, вычисляющие значения тригонометрических функций, предусматривают подстановку аргумента, выраженного в радианах.

Приведем пример программы, выполняющей табулирование функции $y = \sin^2\left(\frac{7\pi}{8} - 2x\right)$ в интервале от $x_{min} = a$ до $x_{max} = b$ с шагом $\Delta x = h$.

Текст программы:

```
#include <stdio.h>
#include <math.h>
int main()
{
    float a,b,h,x,y;
    printf("Start x =  "); scanf("%f",&a);
    printf("End x =  "); scanf("%f",&b);
    printf("Step h = "); scanf("%f",&h);
    float pi = 3.142;
    x = a;
    while(x <= b)
    {
        float y = pow(sin(7*pi/8 - 2*(x*pi)/180),2);
        printf("x =  %f ;\t y = %f\n",x,y);
        x = x + h;
    }
    return 0;
}
```

Заметим, что в языке Си под функцией подразумевается именованный логически целостный фрагмент программы. Данные могут передаваться в функцию, и функция может возвращать значение. Функции могут быть размещены как в одном файле программы, так и в разных файлах. Концепция функции в

языке Си покрывает все типы подпрограмм в других языках программирования: функции и процедуры.

Стандартный вид объявления функций следующий:

```
спецификатор_типа имя_функции (список параметров)
{
    тело функции
}
```

Спецификатор_типа определяет тип значения, возвращаемого функцией с помощью оператора `return`. Это может быть любой допустимый тип. Если тип не указан, предполагается, что функция возвращает целочисленные значения. Список параметров - это разделенный запятыми список переменных, получающий значение аргументов при вызове функции. Функция может быть без параметров и в таком случае список параметров должен состоять из ключевого слова void.

Пример функции, возвращающей квадрат вещественного числа:

```
float f ( float x)
{
    return x*x;
}
```

Приведем текст программы, реализующей процедуру табуляции с использованием функции Си для задания математической функции.

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

float f ( float x)
{
    float pi = 3.142;
```

```
    return pow(sin(7*pi/8 - 2*(x*pi)/180),2);
}

int main()
{
    float a,b,h,x,y;
    printf("Start x =  "); scanf("%f",&a);
    printf("End x =  "); scanf("%f",&b);
    printf("Step  h = "); scanf("%f",&h);
    x = a;
    while(x <= b)
    {
        printf("x =  %f ;\t y = %f\n",x,f(x));
        x = x + h;
    }
    return 0;
}
```

Результаты работы двух приведенных выше программ идентичны.

2.1. Задания на табуляцию функции

№	*табулируемая функция*	*параметр p*	*диапазон изменения аргумента*	Δx
1	$\sqrt{5}\cdot\sin^2\left(x^2+\frac{p}{2}\right)+\cos x$	3,1	$0^0 \le x \le 45^0$	5^0
2	$\operatorname{arctg}\sqrt{\sin\frac{p-x}{p}}$	3,65	$0^0 \le x \le 50^0$	10^0

3	$\frac{\sqrt{p}\cdot\sin x}{\cos^2 x+p}$	4,66	$5^0 \le x \le 45^0$	5^0
4	$\frac{\sin\sqrt{3,2+x}}{p+\cos\left(x+\frac{\pi}{7}\right)}$	2,4	$10^0 \le x \le 50^0$	10^0
5	$\frac{\sqrt{p}\cdot\cos(x+1,7)}{3\sin px}$	2,41	$8^0 \le x \le 72^0$	8^0
6	$1-\operatorname{tg}\frac{x^2+3p}{\sqrt{p^2+x}}$	1,4	$20^0 \le x \le 80^0$	$7,5^0$
7	$p(p^2+1)\cdot\sqrt{\sin\frac{px^2}{x+1}}$	1,9	$5^0 \le x \le 45^0$	5^0
8	$\frac{\sqrt{p}\cdot\sin^2 x}{p+\cos x}$	6,87	$5^0 \le x \le 80^0$	15^0
9	$\frac{\sin^2(x+0,7p)}{p^2+0,3\pi}$	4,55	$8^0 \le x \le 80^0$	12^0
10	$\frac{\sin(p-x^2)-\sqrt{p^4+p^2}}{p^2\cdot\sqrt{p}}$	2,21	$35^0 \le x \le 65^0$	5^0
11	$\frac{p\cdot\sin x+(\sin x+\cos 2x)^2}{\sin x+0,5\pi}$	1,84	$0^0 \le x \le 72^0$	9^0
12	$\sin\sqrt{2+x}\cdot\operatorname{tg}\frac{x^2+p^2}{\sqrt{p^2+1}}$	0,43	$5^0 \le x \le 35^0$	5^0
13	$\frac{p\cdot\sin x}{\sqrt[3]{3\cos x+2}}+e^{px}$	1,89	$20^0 \le x \le 44^0$	4^0
14	$\frac{\ln\cos x}{p\cdot\sin x}+\sqrt{p\cdot\operatorname{tg} x}$	4,81	$10^0 \le x \le 35^0$	5^0
15	$\arcsin p+\frac{\sqrt[3]{p}\cdot\cos 3x}{\sin 5x+2}$	0,21	$3^0 \le x \le 27^0$	3^0

16	$e^{p}+\ln \operatorname{tg}^{3}\left(\frac{x}{p}-\frac{x}{6}\right)$	$3\cdot\ln 3$	$12^0 \le x \le 30^0$	3^0
17	$\sqrt[3]{\sin 3x}\cdot\cos 6x - p\cdot \operatorname{arctg} p$	1,19	$6^0 \le x \le 36^0$	5^0
18	$\sqrt{\operatorname{tg} x+1}+\frac{\cos 3x}{\sqrt[5]{\sin^2 xp}}$	5,12	$5^0 \le x \le 40^0$	7^0
19	$\frac{\cos\sqrt{x+\frac{\pi}{9}}}{p\cdot\sin\sqrt{x+\frac{\pi}{9}}}$	$\frac{\pi}{18}$	$0^0 \le x \le 36^0$	6^0
20	$5\cos^2\left(\frac{p}{4}-\frac{x}{3}\right)+\sin\frac{p}{x}$	$0{,}3\pi$	$5^0 \le x \le 35^0$	3^0

3. МЕТОДЫ НАХОЖДЕНИЯ КОРНЕЙ УРАВНЕНИЙ

Как известно, компьютеры в первую очередь начали применять для выполнения трудоемких расчетов и для получения хотя бы приблизительных значений каких-либо величин при решении таких задач, для которых получить точное аналитическое решение либо невозможно, либо нецелесообразно. Примером таких задач являются задачи на уточнение корней уравнений. При их решении используют различные численные методы. Здесь представлены три метода уточнения корней уравнений, используя которые можно находить корни уравнений с требуемой точностью.

Очень часто на практике приходится решать уравнения вида $f(x) = 0$, где функция $f(x)$ определена и непрерывна в интервале $a \leq x \leq b$. Под решением уравнения понимают нахождение некоторого значения x_0, которое обращает функцию $f(x)$ в ноль. Значение x_0 и называется корнем уравнения $f(x) = 0$.

Если функция представляет собой многочлен, то уравнение $f(x) = 0$ называют алгебраическим. Если же в функцию входят элементарные функции (тригонометрические, показательные, логарифмические и пр.), то такое уравнение называют трансцендентным.

3.1. Метод половинного деления

Метод половинного деления, или метод деления отрезка пополам (дихотомии), чрезвычайно прост, и его алгоритм легко реализуется на ЭВМ.

Пусть необходимо решить уравнение $f(x) = 0$, где функция $f(x)$ непрерывна на отрезке $[a; b]$, и только один корень x_0 заключен в том же интервале. Таким образом, функция $f(x)$ будет знакопеременной на концах отрезка $[a; b]$ (рис. 3.1). Математически это можно записать как $f(a) \cdot f(b) < 0$. Разделим отрезок $[a; b]$ пополам, т. е. найдем $x = \frac{a+b}{2}$, и вычислим значение функции $f(x)$ в этой точке.

Если окажется, что $f(x) = 0$, то x – корень уравнения. Если $f(x) \neq 0$, то выбираем ту половину отрезка $[a; x]$ или $[x; b]$, на концах которой функция $f(x)$

имеет противоположные знаки. На рис. 3.1 это отрезок [*x*; *b*]. Половина отрезка, не содержащая корня [*a*; *x*], отбрасывается. Это означает, что левая граница интервала перемещается в точку деления пополам (*a* = *x*).

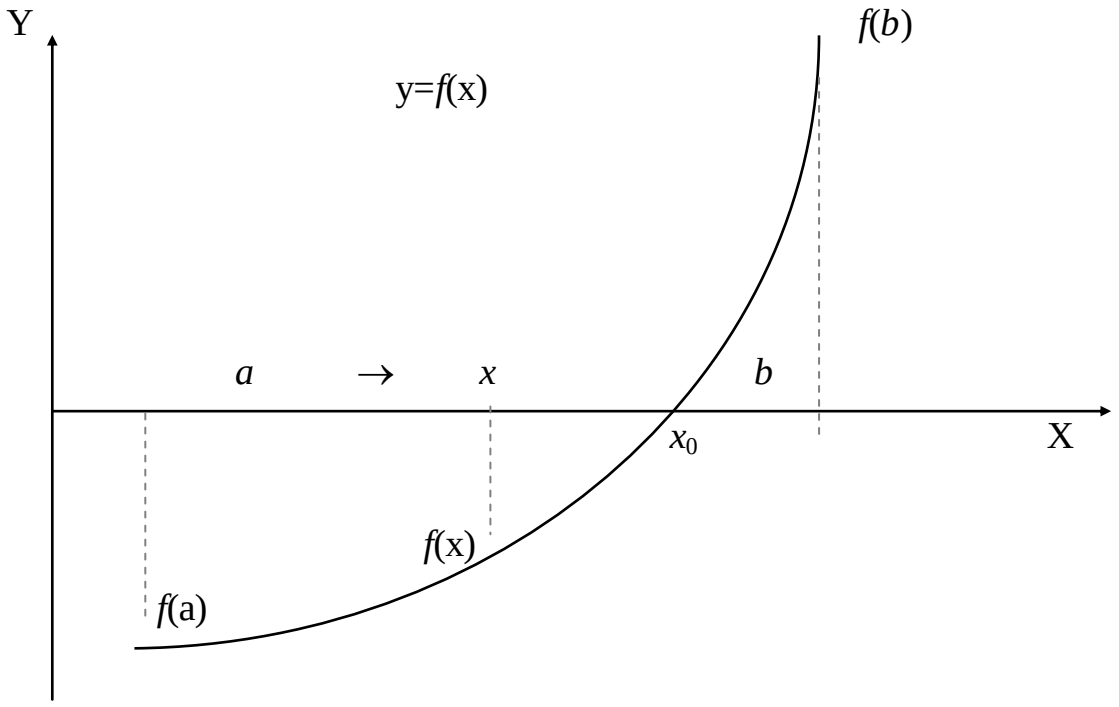

Рис. 3.1. Геометрическое представление метода половинного деления

При повторном делении производятся те же самые операции: новый отрезок [*a*; *b*] делится пополам, вычисляется значение функции в точке деления *f*(*x*) и определяется отрезок, содержащий истинный корень x_0. Процесс деления продолжают до тех пор, пока длина отрезка, содержащего корень, не станет меньше некоторого наперед заданного числа ε (точности) или пока значение функции в точке деления *y* = *f*(*x*) превышает ε по абсолютной величине.

Алгоритм метода половинного деления приведен на рис. 3.2. Отметим, что выделение знакопеременного интервала на основе вычисления произведения *f*(*a*)·*f*(*x*) не является оптимальным, так как необходим лишь знак этого произведения. Поэтому разумнее определять знак на основе использования логических функций сравнения > или <, а также логических функций умножения и сложения: “и“ (&&) и “или“ (||).

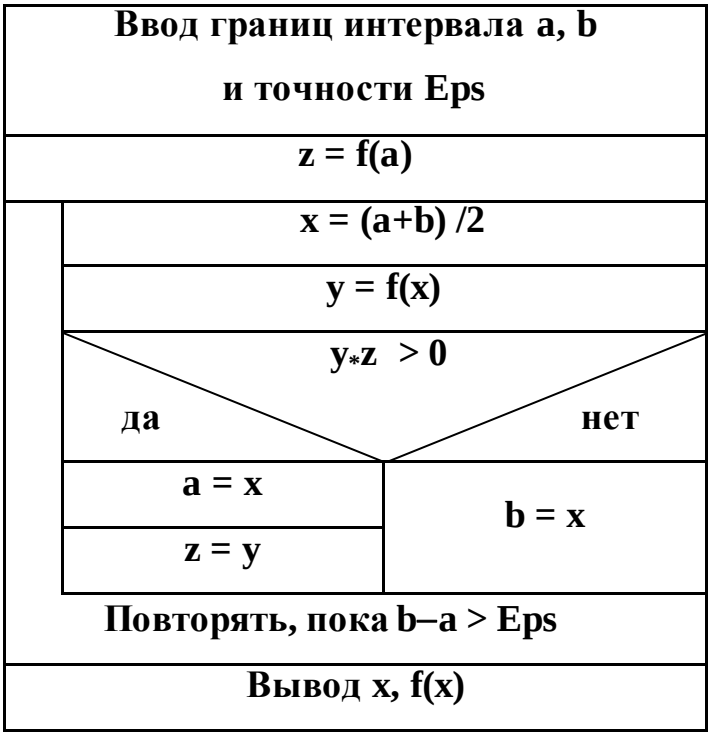

Рис. 3.2. Структурограмма метода половинного деления (метод дихотомии)

Попробуйте самостоятельно определить вид такого «логического» перемножения. Учтите, что возможны два варианта знакопеременности: на левой границе – минус, на правой – плюс, и наоборот.

3.2. Итерационные методы

Метод половинного деления обладает существенным недостатком – медленной сходимостью процесса вычисления корней. При увеличении точности значительно возрастает объем вычислительной работы. Для убыстрения вычислительного процесса применяют методы последовательных приближений (итерационные методы). В их основе – идея вычисления каждого последующего приближения на основании предыдущего, то есть использование рекуррентных формул вида:

$$x_{n+1} = f(x_n). \qquad (2.1)$$

Метод Ньютона (метод касательных)

Пусть уравнение $f(x) = 0$ имеет один корень на отрезке $[a; b]$, а его первая и вторая производные $f'(x)$ и $f''(x)$ определены, непрерывны и сохраняют постоянные знаки в этом интервале.

Геометрически метод Ньютона эквивалентен замене небольшой дуги кривой $y = f(x)$ касательной в некоторой точке кривой (рис. 3.3).

Выберем в качестве первого приближенного значения корня точку $x_1 = b$, для которой выполняется условие $f(x_1) \cdot f''(x_1) > 0$, т.е. значения функции и ее второй производной в точке x_1 имеют одинаковые знаки. Напомним, что знак второй производной от функции $f(x)$ определяет выпуклость кривой: если $f''(x)$ положительна, то график функции имеет выпуклость вниз, как это изображено на рис. 3.3.

Через точку R1 с координатами $[x_1, f(x_1)]$ проведем касательную к кривой $y = f(x)$. В качестве второго приближения x_2 корня возьмем абсциссу точки пересечения этой касательной с осью Ox. Через точку R_2 снова проведем касательную, абсцисса точки пересечения которой даст нам следующее приближение x_3 корня, и т. д.

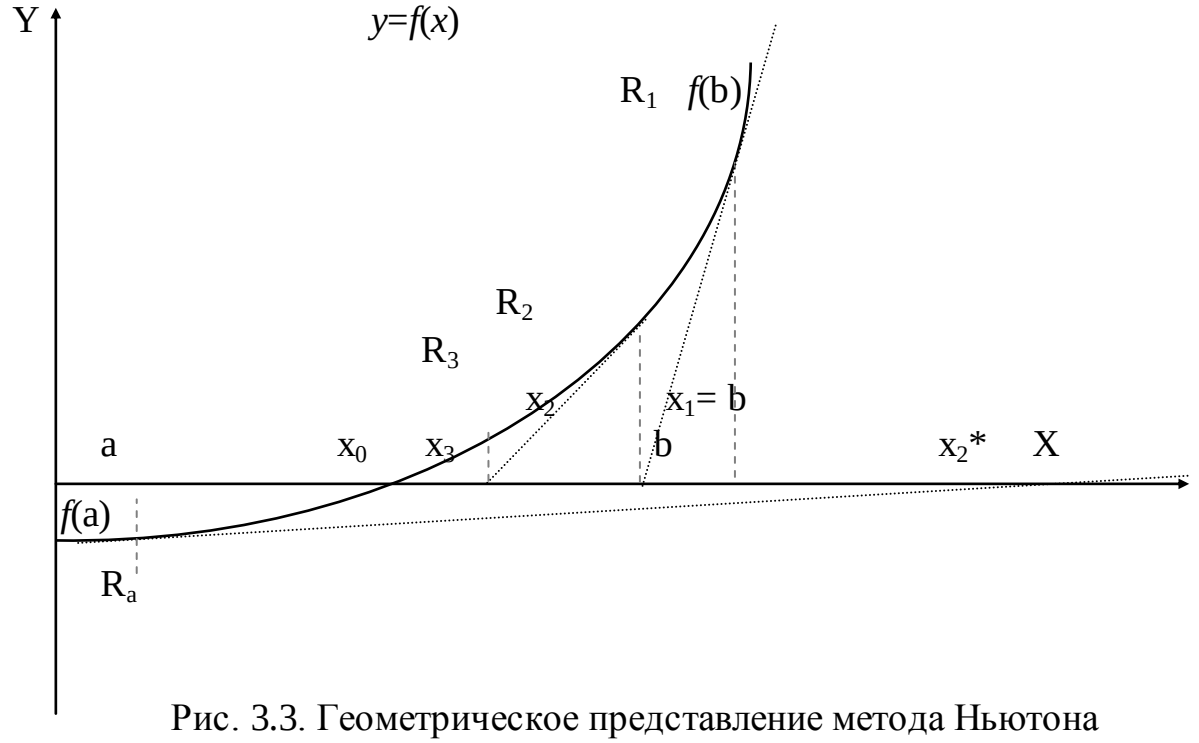

Рис. 3.3. Геометрическое представление метода Ньютона

Уравнение касательной, проходящей через точку x, имеет вид

$$y - f(x_1) = f'(x_1)\cdot(x - x_1). \quad (2.2)$$

Полагая $y = 0$, а $x = x_2$, найдем абсциссу x точки пересечения касательной с осью Ox:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}. \quad (2.3)$$

Следующее приближение x_3 находим по формуле:

$$x_3 = x_2 - f(x_2) / f'(x_2). \quad (2.4)$$

Применяя формулы уточнения значения корня многократно, получим общий вид рекуррентной формулы метода касательных:

$$x_{n+1} = x_n - f(x_n) / f'(x_n). \quad (2.5)$$

Процесс вычислений заканчивают обычно по условию

$$|x_{n+1} - x_n| < \varepsilon. \quad (2.6)$$

При значениях $\varepsilon < 10^{-4}$ погрешность вычисления корня приблизительно равна абсолютной погрешности ε.

Заметим, что если в нашем случае в качестве первого приближенного значения корня положить $x = a$, то, проведя касательную в точке R_a, мы получили бы точку x_2* (рис. 3.3), которая лежит вне отрезка $[a, b]$. В этом случае знаки функции и ее второй производной в точке a противоположны, и сходимость метода не гарантируется.

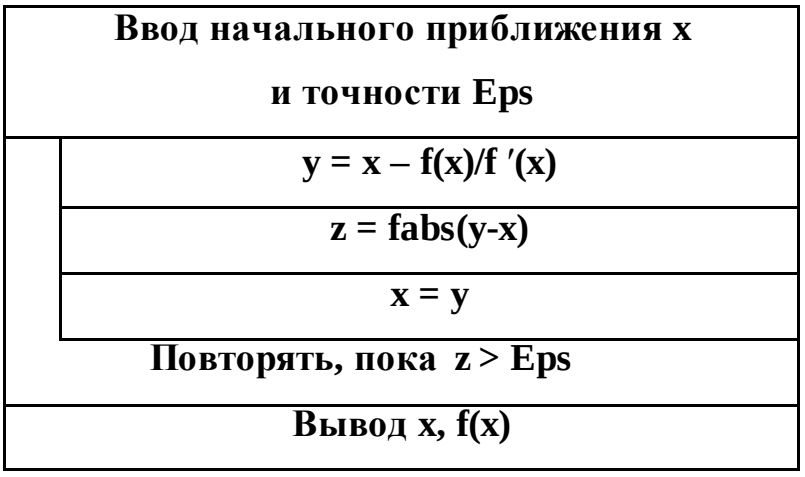

Рис. 3.4. Структурограмма метода Ньютона

Таким образом, для того, чтобы использовать метод Ньютона, необходимо прежде всего вычислить значения первой и второй производных, а затем выбрать начальное приближение корня, исходя из условия $f(x) \cdot f''(x) > 0$. Заметим, что вычисления по этому методу могут оказаться очень долгими или вообще невозможными, если функция в области корня имеет малую крутизну.

Как видно из изложенной теории, алгоритм вычислений очень прост. Изобразим его в виде структурограммы (рис. 3.4).

Метод последовательных приближений

Метод итераций – один из наиболее распространенных способов численного решения уравнений. Сущность метода заключается в следующем. Пусть дано уравнение

$$f(x) = 0, \qquad (2.7)$$

и необходимо определить его действительные корни.

Представим уравнение (2.7) в форме

$$x = \varphi(x). \qquad (2.8)$$

Для такого преобразования необходимо исходную функцию $f(x)$ «разорвать» на две части – x и $\varphi(x)$, т.е. расписать $f(x)$ как $f(x) = x - \varphi(x)$.

Выберем каким–либо способом на отрезке $[a; b]$, содержащем корень, приближенное значение корня x_1 (первое приближение) и подставим его в правую часть уравнения (2.8). Получим некоторое число:

$$x_2 = \varphi(x_1). \qquad (2.9)$$

Подставляя теперь в правую часть равенства (2.9) вместо x_1 число x_2, получим новое число $x_3 = \varphi(x_2)$. И вообще, для некоторого значения x_{n+1} будем иметь рекуррентную формулу:

$$x_{n+1} = \varphi(x_n). \qquad (2.10)$$

Процесс последовательного вычисления уточненных значений корня по формуле (2.10) и называется методом итераций.

Геометрически способ итераций можно пояснить следующим образом (рис. 3.5). На плоскости xOy построим графики двух функций: $y = x$ и $y = \varphi(x)$. Абсцисса точки пересечения этих двух графиков является корнем уравнения (2.8), а, следовательно, и равносильного ему уравнения (2.7).

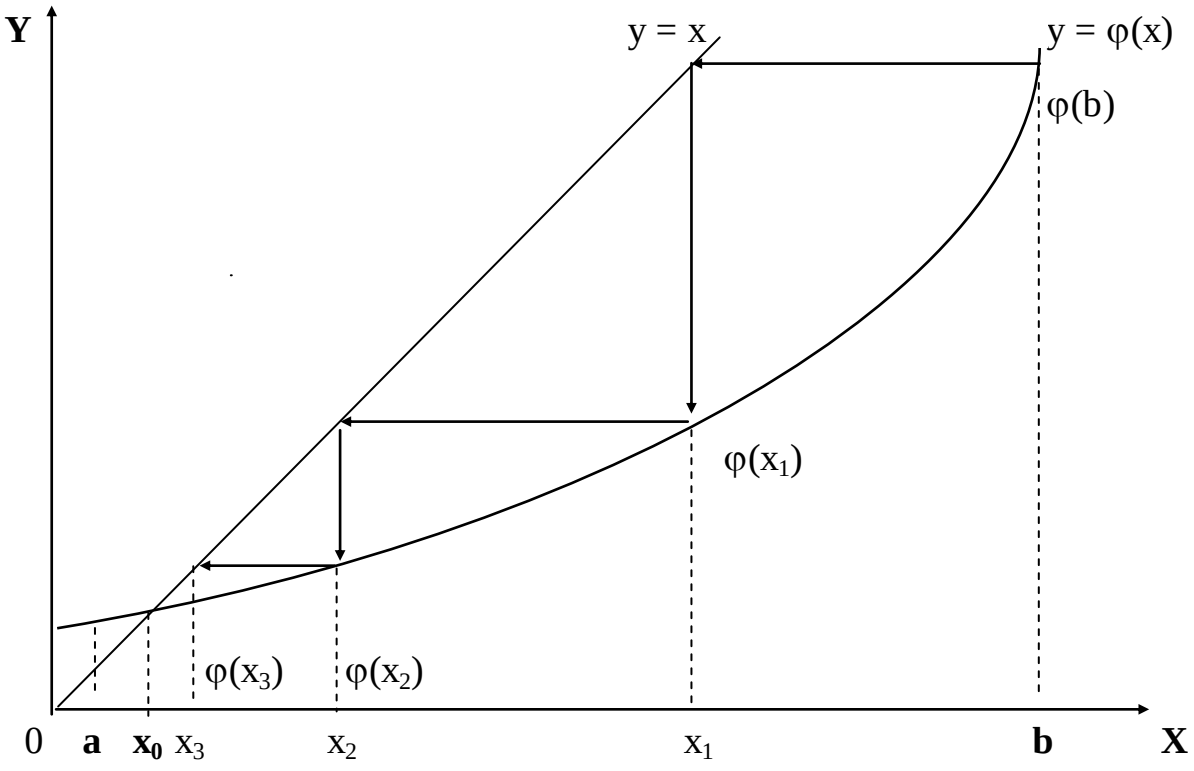

Рис. 3.5. Геометрическое представление метода итераций

Процесс итераций (2.10) сходится, если значение первой производной от правой части уравнения (2.8) по абсолютной величине меньше единицы: $|\varphi'(x)| < 1$.

При практическом вычислении корней по методу итераций нужно при переходе от уравнения (2.7) к уравнению (2.8) стремиться представить функцию $\varphi(x)$ так, чтобы ее производная $\varphi'(x)$ была меньше единицы по абсолютной величине. За условие окончания вычислений можно принять соотношение $|x_{n+1} - x_n| < \varepsilon$.

Алгоритм метода итераций приведен на рис. 3.6. В него добавлен счётчик числа итераций n.

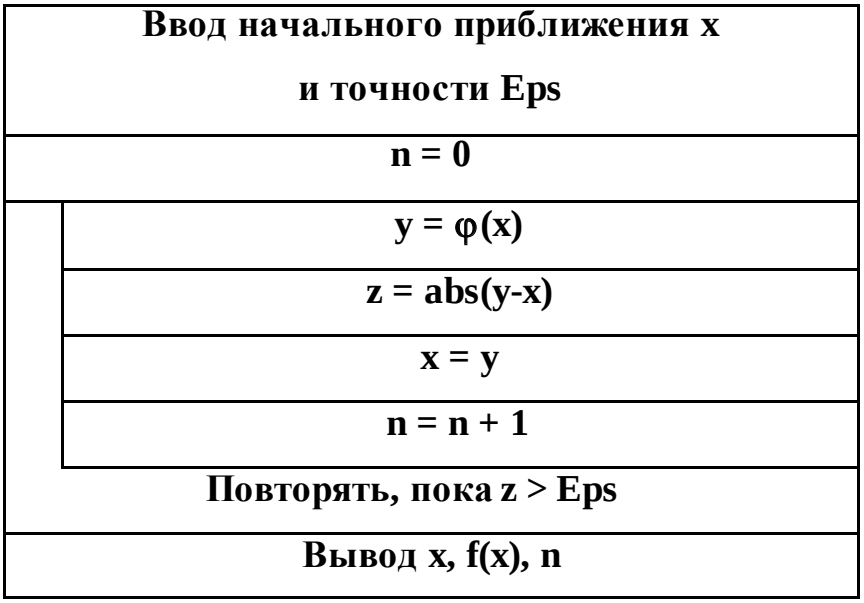

Рис. 3.6. Структурограмма метода итераций

Для выбора начального приближения корня x вычисляют значения первых производных в граничных точках интервала $[a, b]$, содержащего корень, и за исходную величину принимают ту из точек, для которой выполняется условие $|\varphi'(x)| < 1$. Если обе производные по абсолютной величине меньше единицы, то за начальное приближение корня берут ту из граничных точек, для которой значение производной меньше. При соблюдении этого правила процесс итераций сходится быстрее.

3.3. Задания на нахождение корня уравнения

Решить задачу нахождения корня уравнения на указанном интервале. Рекомендуется ввести в алгоритм счётчик циклов и задавать точность $\varepsilon = 10^{-4} \div 10^{-5}$.

№	Уравнение	Интервал
1	$e^x - \ln x - 10x = 0$	[3; 4]
2	$x - 2 + \sin\frac{1}{x} = 0$	[1,2; 2]
3	$\arccos x - \sqrt{1 - 0{,}3x^3} = 0$	[0; 1]
4	$\cos x - e^{-\frac{x^2}{2}} + x - 1 = 0$	[1; 2]
5	$3 \cdot x - 4 \cdot \ln x - 5 = 0$	[2; 4]
6	$\ln x - x + 1{,}8 = 0$	[2; 3]
7	$\cos\frac{2}{x} - 2\sin\frac{1}{x} + \frac{1}{x} = 0$	[1; 2]
8	$x \cdot \operatorname{tg} x - \frac{1}{3} = 0$	[0,2; 1]
9	$x - \frac{1}{3 + \sin 3{,}6x} = 0$	[0; 0,85]
10	$0{,}6 \cdot 3^x - 2{,}3 \cdot x - 3 = 0$	[2; 3]
11	$2x \cdot \sin x - \cos x = 0$	[0,4; 1]
12	$\sqrt{1 - 0{,}4x^2} - \arcsin x = 0$	[0; 1]
13	$x + \sqrt{x} + \sqrt[3]{x} - 2{,}5 = 0$	[0,4; 1]
14	$\operatorname{tg}\frac{x}{2} - \operatorname{ctg}\frac{x}{2} + x = 0$	[1; 2]

15	$3 \sin\sqrt{x} + 0{,}35x - 3{,}8 = 0$	[2; 3]
16	$\sin(\ln x) - \cos(\ln x) + 2 \cdot \ln x = 0$	[1; 3]
17	$3 \cdot \ln^2 x + 6 \cdot \ln x - 5 = 0$	[1; 3]
18	$0{,}4 + \operatorname{arctg}\sqrt{x} - x = 0$	[1; 2]
19	$e^x + \sqrt{1 + e^{2x}} - 2 = 0$	[–1; 0]
20	$3 \cdot x - 14 + e^x - e^{-x} = 0$	[1; 3]

4. ВЫЧИСЛЕНИЕ ОПРЕДЕЛЕННОГО ИНТЕГРАЛА

Величина определенного интеграла $I = \int_a^b f(x)dx$ численно равна площади S геометрической фигуры, образованной графиком подынтегральной функции $f(x)$, осью абсцисс Ox и перпендикулярами, восстановленными в точках $x = a$ и $x = b$ на оси Ox до пересечения с кривой $f(x)$ (рис. 4.1). Напомним, что точное значение определенного интеграла вычисляется на основе суммирования бесконечного ряда вида:

$$I = \lim_{\Delta x \to 0} \sum_i f(x_i) \cdot \Delta x .$$

Однако производить бесконечные вычисления с бесконечной точностью не позволит ни один реальный компьютер. Поэтому при численном подсчете интеграла необходимо ограничиться конечным (а не бесконечно малым) значением Δx.

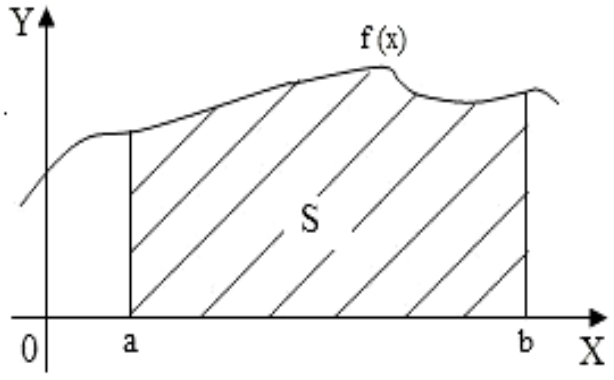

Рис. 4.1. К геометрическому смыслу определенного интеграла

Для нахождения S разбиваем отрезок $[a,b]$ на n равных частей, из точек разбиения восстанавливаем перпендикуляры до пересечения с $f(x)$. Тем самым получаем n криволинейных трапеций, суммарная площадь S которых $S_1 + S_2 + \ldots + S_n$ приближается к искомой величине I. С ростом n (в некотором диапазоне) величина S становится все ближе к значению I, но совпасть «мешает» фиксированная разрядность чисел, используемых в компьютере.

Предлагаемые ниже методы численного решения интеграла основаны на “аппроксимации” криволинейной трапеции S_i такой геометрической фигурой, нахождение площади которой не представляет больших трудностей.

Метод прямоугольников

Метод основан на замене криволинейной трапеции прямоугольником, высота которого вычисляется в определённой точке интервала интегрирования $[x_i, x_i + \Delta x]$, например, в левой. В этом случае значение интеграла:

$$I \approx S = [f(a) + f(a + \Delta x) + f(a + 2\cdot\Delta x) + \ldots + f(a + (n-1)\cdot\Delta x)]\cdot\Delta x.$$

Метод трапеций

Соединяя на каждом отрезке интегрирования точки $f(x_i)$ и $f(x_i+\Delta x)$ отрезком прямой, получаем прямоугольную трапецию. В результате такой аппроксимации приближённое значение интеграла I можно рассчитать по формуле:

$$I \approx S = [0{,}5\cdot(f(a) + f(b)) + f(a + \Delta x) + f(a + 2\cdot\Delta x) + \ldots + f(a + (n-1)\cdot\Delta x)]\cdot\Delta x\ .$$

Метод парабол (Симпсона)

Проводя через три следующих подряд значения функции параболу, можно приближённо представить интеграл I как сумму площадей параболических трапеций:

$$I \approx S = [(f(a) + f(b) + 4(f(a + \Delta x) + 2(f(a + 2\cdot\Delta x) + 4(f(a + 3\cdot\Delta x) + \ldots$$

$$+ 4(f(a + (n-1)\cdot\Delta x)]\cdot\Delta x / 3 = \left[f(a) + f(b) + \sum_{i=1}^{n-1} (3 + C)\cdot f(a + i\cdot\Delta x) \right]\cdot\frac{\Delta x}{3},$$

где: $$C = \begin{cases} +1, \textit{если}\ \text{i} - \text{нечётное}, \\ -1, \textit{если}\ \text{i} - \text{чётное}. \end{cases}$$

Количество разбиений n обязательно должно быть чётным, поскольку число параболических трапеций в 2 раза меньше.

Вычисление интеграла с заданной точностью здесь основано на том, что формально точность должна возрастать с увеличением числа разбиений n интервала интегрирования. Пусть S_n – значение интеграла, полученное при разбиении интервала интегрирования на n частей, и S_{2n} – значение, вычисленное при удвоенном числе разбиений. Тогда можно использовать следующие критерии достижения заданной точности ε:

$$| S_{2n} - S_n| < \varepsilon \text{ при } S_{2n} < 1 \quad \text{и} \quad |(S_{2n} - S_n) / S_{2n} | < \varepsilon \text{ при } S_{2n} \geq 1.$$

На рис. 4.2 приведён алгоритм численного решения интеграла с заданной точностью. Непосредственное нахождение значения интеграла по какому-либо из предложенных выше методов предлагается оформить в виде подпрограммы.

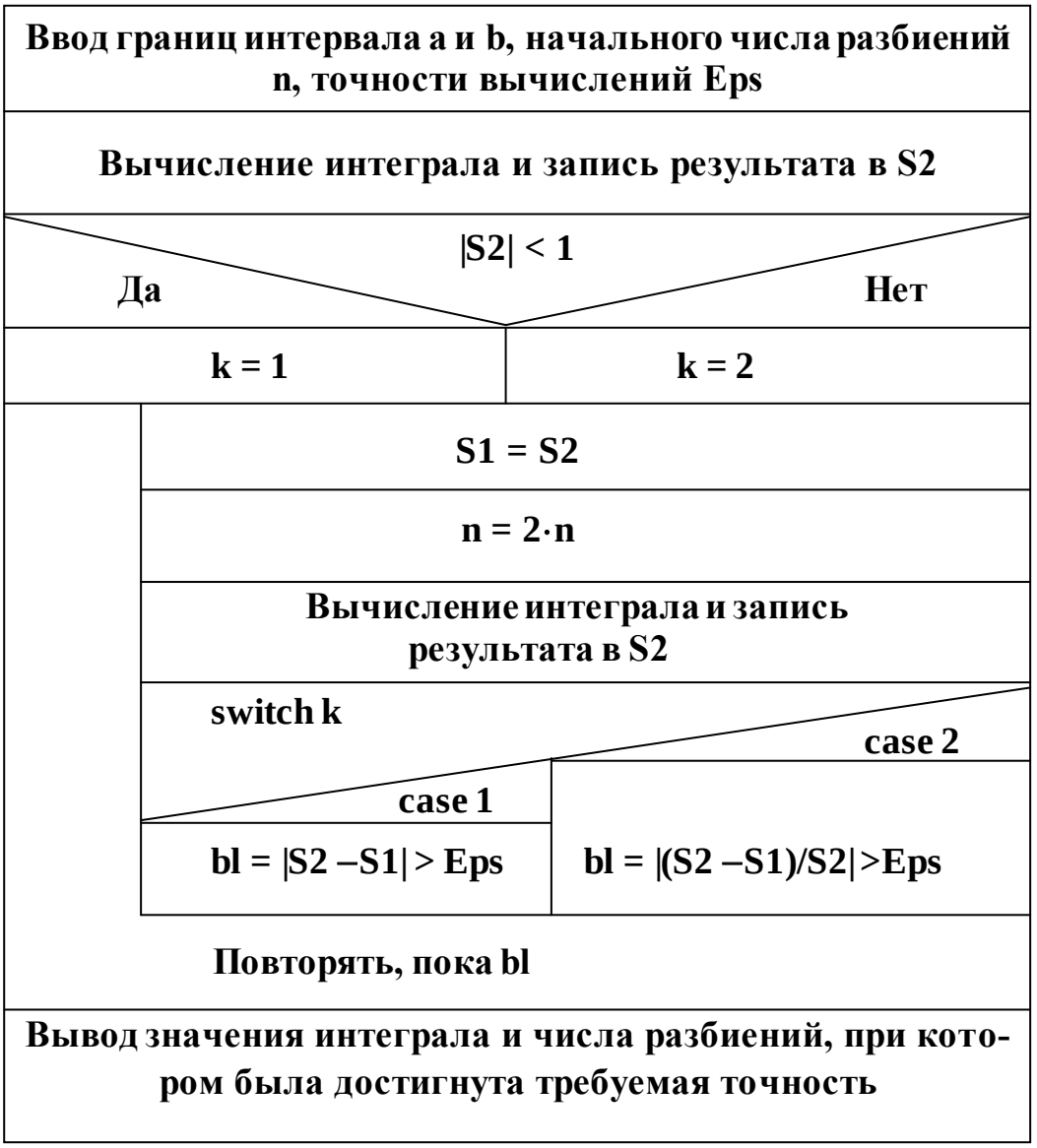

Рис. 4.2. Структурограмма алгоритма решения определённого интеграла

4.1. Задания на вычисление определённого интеграла

№	Подынтегральная функция	Пределы интегрирования
1	e^{-x^2}	0; 1
2	$\cos x^2$	0; 1
3	$e^x \cdot \sin x$	0; π
4	$x \cdot \operatorname{arctg} x$	0; √3
5	$(x^2 - 1) \cdot 10^{-2x}$	0; 1
6	$\sqrt{e^x - 1}$	0,1; 2
7	$x^2 \cdot (1 - x)^2$	0; 1
8	$x^3 \cdot e^{2x}$	0; 1
9	$\sqrt{2 + \sin x}$	0; π/2
10	$x \cdot \ln (1 + x)$	0; 0,5
11	$\cos^3 x \cdot \sin^2 x$	0; π/3
12	$\sqrt{2 - x^3}$	0; 1
13	$x \cdot \cos x$	0; π
14	$x \cdot \ln (1 + x^3)$	0; 0,5
15	$\cos^2 6x$	0; π/2
16	$\sqrt{1 + x^2}$	0; 0,5
17	$\sqrt{4 - x^3}$	0; 1
18	$x \cdot \sin x^2$	0; 1
19	$\operatorname{arctg} x^2$	0; 0,5
20	$\sqrt{2 - \cos x}$	0; π

5. ВЫЧИСЛЕНИЕ КОНЕЧНЫХ СУММ

Часто для вычисления значений некоторой функции используют разложение этой функции в бесконечный ряд. Так, например, функцию $y = e^x$ можно представить в виде ряда Маклорена:

$$e^x = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + R_n,$$

где $\frac{x^n}{n!}$ – общий член ряда, а R_n – остаточный член ряда. Так как ряд сходится, т.е. остаточный член при увеличении значения n стремится к нулю, то функцию $y = e^x$ можно аппроксимировать конечной суммой вида:

$$S = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} \approx e^x, \text{ если } R_n \le \varepsilon,$$

где ε – заранее заданная точность аппроксимации.

Вычисление конечных сумм для различных значений аргумента функции x и для различного количества членов ряда n представляет собой задачу с двумя вложенными циклами. Внутренний цикл – вычисление суммы n членов ряда для фиксированного значения x; внешний цикл – изменение аргумента x в заданном интервале.

Алгоритм вычисления конечной суммы зависит от вида общего члена ряда.

Ряды, содержащие факториалы и степени высоких порядков

$$S = 1 + \ln 3 \cdot x + \frac{\ln^2 3}{2!}x^2 + \ldots + \frac{\ln^n 3}{n!}x^n$$

$$S = 1 - x^2 + \frac{x^4}{2!} - \frac{x^6}{3!} + \ldots + (-1)^n \cdot \frac{x^{2n}}{n!},$$

требуют вычисления факториала и возведения аргумента в степени высоких порядков для каждого члена ряда. Прямое использование формулы общего члена ряда приводит к увеличению объема вычислительной работы и, при больших n, к уменьшению точности.

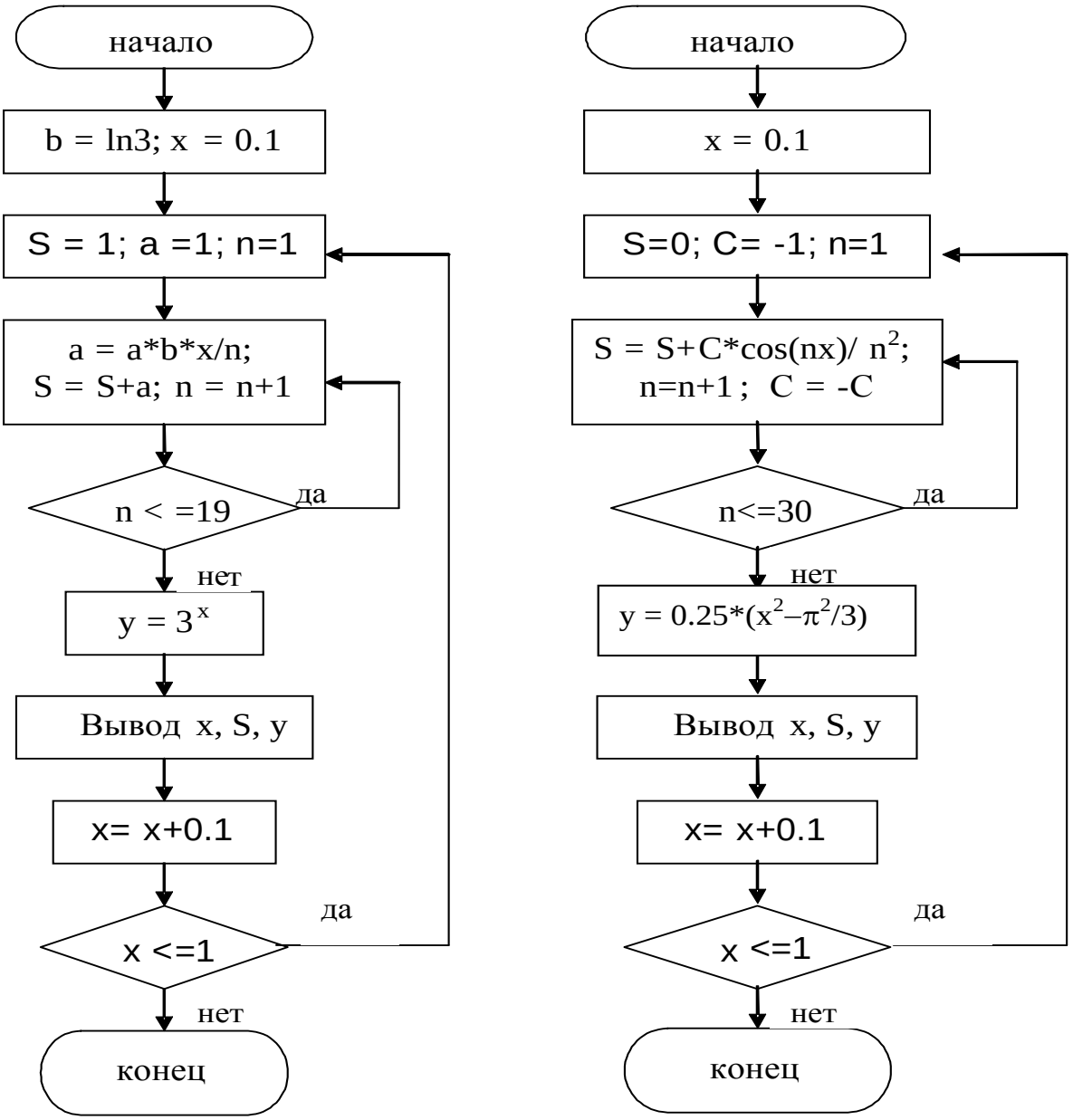

Рис. 5.1. Блок-схемы вычислений сумм двух рядов различного типа

а) $y = 3^x$ б) $y = \frac{1}{4}\left(x^2 - \frac{\pi^2}{3}\right)$

В этих случаях для вычисления очередного члена суммы используют рекуррентные соотношения, т.е. выражают последующий член ряда через предыдущий. Например, необходимо вычислить сумму 20 членов ряда:

$$S = 1 + \ln 3 \cdot x + \frac{\ln^2 3}{2!}x^2 + \ldots + \frac{\ln^n 3}{n!}x^n,$$

где аргумент x изменяется в интервале [0;1] с шагом 0,1.

Эта сумма является частичной суммой ряда для функции $y = 3^x$. Составим рекуррентную формулу, основываясь на формулах n-го и n+1-го членов ряда:

$$a_n = \frac{\ln^n 3}{n!}x^n; \quad a_{n+1} = \frac{\ln^{n+1} 3}{(n+1)!}x^{n+1} = \frac{\ln^n 3}{n!}x^n \cdot \frac{\ln 3}{n+1}x$$

то есть $a_{n+1} = a_n \cdot \frac{\ln 3}{n+1}x$.

Полагая $a_n = 1$ при $n = 0$, по этой формуле можно определить все члены ряда, исключая первый.

Суммирование проводится методом накопления. Присвоим переменной S начальное значение, равное первому члену суммы, в данном случае $S = 1$. Затем, вычисляя очередные члены ряда по рекуррентной формуле, прибавляем каждое из этих значений к накопителю: $S = S + a_n$. При составлении алгоритма (рис. 5.1(а)) учтем, что нам необходимо вычислить 19 следующих членов ряда, так как мы начали накопление с первого члена.

Параллельно, для контроля правильности алгоритма, вычисляем и значение исходной функции $y = 3^x$.

Ряды, не содержащие факториалов и степеней высоких порядков

В качестве примера рассмотрим нахождение частичной суммы разложения в ряд функции: $y = \frac{1}{4}\left(x^2 - \frac{\pi^2}{3}\right)$

$$S = -\cos x + \frac{\cos 2x}{2^2} - \frac{\cos 3x}{3^2} + \ldots + (-1)^n \cdot \frac{\cos nx}{n^2}.$$

Для рядов подобного типа нецелесообразно прибегать к составлению рекуррентных формул. Каждый очередной член определяется подстановкой зна-

чения n в формулу общего члена. Так, при n = 1 запишем a_1 = -cos(x), при n = 2 получаем $a_2 = \frac{\cos 2x}{2^2}$ и т.д.

Составим алгоритм вычисления суммы этого знакопеременного ряда. Пусть количество слагаемых n = 30, диапазон изменения аргумента $0{,}1 \le x \le 1$ и его шаг Δx = 0,1. Положим S = 0, n = 1, C = –1 и вычислим первый член суммы по формуле общего члена ряда. Изменив знак C на обратный, в следующем выполнении цикла имеем второй член суммы и т. д. Полностью алгоритм нахождения суммы рассматриваемого ряда приведён на рис. 5.1(б).

Ряды смешанного типа

$$S = 1 + \frac{\cos x}{1!} + \frac{\cos 2x}{2!} + \ldots + \cdot \frac{\cos nx}{n!}$$

$$S = \frac{x^3}{3} - \frac{x^5}{15} + \frac{x^7}{35} + \ldots + (-1)^{n+1} \cdot \frac{x^{2n+1}}{4n^2 - 1}.$$

В подобных рядах целесообразно для сомножителей, содержащих степени высших порядков и факториалы, составлять рекуррентные соотношения, а остальные сомножители вычислять непосредственной подстановкой значения n. В качестве примера рассмотрим вычисление значений ряда

$$S = \frac{x^3}{3} - \frac{x^5}{15} + \frac{x^7}{35} + \ldots + (-1)^{n+1} \cdot \frac{x^{2n+1}}{4n^2 - 1}$$

для n = 30, $0{,}1 \le x \le 1$, Δx = 0,1. Сумма членов этого ряда является частичной суммой разложения функции $y = \frac{1 + x^2}{2} arctgx - \frac{x}{2}$. Для каждого члена ряда сомножители $(-1)^{n+1}$ и x^{2n+1} будем вычислять рекуррентно, а $1/(4n^2-1)$ непосредственно. Ниже приведён листинг программы решения этой задачи.

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

```
{
    double x,a,S,y;
    int n;
    for (x=0.1; x<=1; x+=0.1)
    {
        a=-x;
        S=0;
        for (n=1;n<=30;n++)
        {
            a=-a*x*x;
            S=S+a/(4*n*n-1);
        }
        y=0.5*((x*x+1)*atan(x)-x);
        printf("x=%3.1f  S=%8.5f  y=%8.5f \n",x,S,y);
    }
    getchar();          /* ожидание нажатия клавиши */
    return 0;
}
```

Так как алгоритмы вычисления конечных сумм очень индивидуальны, рекомендуется составленную программу проверить подстановкой для первых двух-трех членов суммы. Для контроля правильности составления алгоритма в наших задачах параллельно с вычислением суммы значение функции определяется и непосредственно.

5.1. Задания на нахождение конечной суммы ряда

а) Решить задачу вычисления конечной суммы ряда для заданного числа n при изменении аргумента x в заданном диапазоне, количество шагов изменения x равно 10,

б) Определить число членов конечной суммы n, позволяющее аппроксимировать функцию с заданной точностью ε (например, $\varepsilon = 10^{-5}$), исходя из критерия $|S_n - S_{n-1}| = |a_n| < \varepsilon$.

№	Сумма	Диапазон изменения аргумента	n	Функция y(x)
1	$S = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$	0,1 ÷ 1	10	sin(x)
2	$S = \cos(0.8) - x \cdot \sin(0.8) - \frac{x^2 \cdot \cos(0.8)}{2!} + \frac{x^3 \cdot \sin(0.8)}{3!} + \ldots + \frac{x^n \cdot \cos(0.8 + \frac{n\pi}{2})}{n!}$	0,1 ÷ 1	15	cos(x + 0,8)
3	$S = 1 + x + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}$	1 ÷ 2	15	e^x
4	$S = \frac{\cos(2x)}{3} + \frac{\cos(4x)}{15} + \frac{\cos(6x)}{35} + \ldots + \frac{\cos(2nx)}{4n^2 - 1}$	0,1 ÷ 0,8	50	$0.5 - \frac{\pi}{4} \cdot \lvert\sin(x)\rvert$
5	$S = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots + (-1)^{n+1} \frac{x^{2n+1}}{2n+1}$	0,14 ÷ 0,5	40	arctg(x)
6	$S = 1 + \frac{\cos(x)}{1!} + \frac{\cos(2x)}{2} + \ldots + \frac{\cos(nx)}{n!}$	0,1 ÷ 1	20	$e^{\cos(x)} \cdot \cos(\sin(x))$
7	$S = 1 - x^2 + \frac{x^4}{2!} - \frac{x^6}{3!} + \ldots + (-1)^{n+1} \cdot \frac{x^{2n}}{n!}$	0,01 ÷ 1	10	e^{-x^2}
8	$S = -\frac{(2x)^2}{2!} + \frac{(2x)^4}{4!} - \frac{(2x)^6}{6!} + \ldots + (-1)^n \frac{(2x)^{2n}}{(2n)!}$	0,1 ÷ 1	10	$2 \cdot \cos^2(x - 1)$
9	$S = \frac{x^3}{3} - \frac{x^5}{15} + \frac{x^7}{35} + \ldots + (-1)^{n+1} \frac{x^{2n+1}}{4n^2 - 1}$	0,2 ÷ 1	20	$\frac{1 + x^2}{2} \operatorname{arctg}\left(-\frac{x}{2}\right)$
10	$S = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots + (-1)^n \frac{x^{2n}}{(2n)!}$	0,1 ÷ 1	10	cos(x)

11	$S = 1 + \frac{3x}{1!} + \frac{(3x)^2}{2!} + \ldots + \frac{(3x)^n}{n!}$	0,l ÷ l	15	e^{3x}
12	$S = \sin(0.5) + x \cdot \cos(0.5) - \frac{x^2 \cdot \sin(0.5)}{2!} - \frac{x^3 \cdot \cos(0.5)}{3!} + \ldots + \frac{x^n \cdot \cos(0.5 + \frac{n\pi}{2})}{n!}$	0,1 ÷ l	10	sin(x+0,5)
13	$S = 1 + \frac{\ln 5 \cdot x}{1!} + \frac{\ln^2 5}{2!} \cdot x^2 + \ldots + \frac{\ln^n 5}{n!} \cdot x^n$	0,l ÷ l	10	$e^x \cdot \ln 5$
14	$S = \frac{(x-1)}{x} + \frac{(x-1)^2}{2x^2} + \frac{(x-1)^3}{3x^3} + \ldots + \frac{(x-1)^n}{nx^n}$	0,75 ÷ 2	15	ln(x)
15	$S = 1 + \frac{3x^2}{1!} + \frac{5x^4}{2!} + \frac{7x^6}{3!} + \ldots + \frac{(2n+1) \cdot x^{2n}}{n!}$	0,l ÷ l	10	$(1 + 2x^2)e^{x^2}$
16	$S = -\left(x + \frac{x^2}{2} + \frac{x^3}{3} \ldots + \frac{x^n}{n}\right)$	0,l ÷ 0,9	60	ln(1 - x)
17	$S = 2 \cdot \left(x + \frac{x^3}{3} + \frac{x^5}{5} + \ldots + \frac{x^{2n+1}}{2n+1}\right)$	0,l ÷ 0,9	35	$\ln\left(\frac{1+x}{1-x}\right)$
18	$S = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \ldots + \frac{x^{2n}}{(2n)!}$	l ÷ 3	10	ch(x)
19	$S = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} + \ldots + (-1)^n \frac{x^{2n}}{(2n+1)!}$	0,l ÷ 1	10	$\frac{\sin x}{x}$
20	$S = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \ldots + \frac{x^{2n+1}}{(2n+1)!}$	l ÷ 3	20	sh(x)

6. ИНДЕКСИРОВАННЫЕ ПЕРЕМЕННЫЕ

6.1. Одномерный массив

Упорядоченное множество данных может быть представлено в виде так называемых регулярных типов или массивов. Для задания массива требуется указать его имя (идентификатор), тип входящих в него элементов, размер, равный количеству элементов в массиве, а также размерность (вектор, матрица, трехмерный массив и т. д.), равную количеству индексов, необходимых для доступа к отдельным элементам.

Объявление массива сводится к указанию типа его элементов и количества элементов. Общий вид оператора объявления индексированных переменных имеет вид:

```
идентификатор_типа имя_массива [размер][размер];
```

Например, объявление массива целых чисел размером в 10 элементов имеет следующий вид:

```
int x[10];
```

Размер массива задается целой константой. Тип элемента массива может быть любым.

Размер массива, определяемый количеством его элементов, в Си фиксируется при его объявлении в программе и не может быть изменён в процессе её выполнения. Преимущества использования в программах массивов, к элементам которых обращаются с помощью индексированных переменных, наиболее явно проявляются в том случае, когда необходимо производить одинаковые операции над большим количеством чисел. Так как элементы массива обозна-

чаются одним именем и различаются лишь местоположением (индексом), алгоритм решения таких задач основывается на циклическом повторении некоторой операции, причем параметром цикла, как правило, является индекс. В тексте программы для обозначения элементов массива используется имя массива, а далее в квадратных скобках указывается индекс элемента массива. Индексация элементов массива в программах, написанных на языке Си, всегда начинается с 0.

Пример подобного алгоритма – ввод и вывод одномерных массивов с известным количеством элементов. Ниже представлен листинг программы подсчета итоговой суммы заработной платы некоторого подразделения:

```
                /* Ввод и вывод массива */
#include <stdio.h>

int main()
{
  const int NumOfMen=30;

  int n,s,i;
  int Salary[NumOfMen];
  printf("Количество сотрудников?\n");
  scanf( "%d", &n);
  if ((n==0) || (n>NumOfMen))
  {
     printf("Ошибка! Недопустимое количество сотрудников!\n");
     exit(4);
    }
    printf("Вводите значения зарплат \n ");
    s=0;
```

```
       /* Цикл ввода и суммирования */
      for(i=0;i<n;i++)
      {
          printf(" %d -я зарплата: ",i+1);
          scanf("%d", &Salary[i]);
          s = s+Salary[i];
       }
       /* Цикл вывода в строку */
    for(i = 0; i< n; i++)
      printf( "%d \t \n",Salary[i]); /*При выводе разде-
ляем числа символами табуляции */
    printf("Сумма зарплат равна %d: \n", s);
    getchar();     /* Задержка экрана пользователя */
  }
```

Достаточно часто приходится сталкиваться с задачей упорядочения числовых массивов, т. е. расположения элементов исходного набора данных в порядке возрастания или убывания. Эта задача, поставленная в общем виде, называется задачей сортировки.

Сортировка – перегруппировка заданного множества объектов в некотором определенном порядке. Цель сортировки – облегчить поиск отдельных элементов в таком отсортированном множестве (телефонная книга, оглавление книги, библиотеки, словари, склады и т. д.).

Основным условием алгоритма сортировки обычно является экономное использование доступной памяти: перестановки должны выполняться на том же месте, где расположен исходный массив, и количество дополнительно используемых ячеек памяти должно быть минимально. В качестве примера приведена программа сортировки методом “пузырька” или, по-другому, методом прямого обмена.

Метод основан на поочерёдном сравнении стоящих рядом элементов массива; если порядок в паре неправилен, элементы меняются местами. Для выполнения такой перестановки требуется одна дополнительная ячейка памяти.

Ниже приведен листинг программы сортировки по убыванию числового одномерного массива, написанный на языке Си.

```
      /* Программа сортировки методом пузырька */
#include <stdio.h>

int main()
{
    int   i, j, j1, n, n1;
    float t, x[1000];
    printf("Введите длину последовательности: ");
    scanf("%d", &n);
    for(i = 0; i< n; i++)
    {
        printf(" Введите %d -й элемент: ", i);
        scanf("%f", &x[i]);
    }
    printf("Для запуска сортировки нажмите (Enter)");
    getchar();
    n1= n-1;
    for(i = n-1; i>=0; i--)
        for(j = 0; j<i; j++)
        {
            j1 = j+1;
            if (x[j] < x[j1])
```

```
                {   /* перестановка элементов */
                    t = x[j];
                    x[j] = x[j1];
                    x[j1] = t;
                }
            }
        printf("Сортировка закончена. Перед просмотром результата задайте число элементов в строке вывода ");
        scanf("%d", &j);
        for(i = 0; i< n; i++)
        {
            printf("%8.1f", x[i]);
            if ((i+1)% j == 0) printf("\n");
        }
        printf("Для завершения нажмите (Enter)");
        getchar();
        return 0;
    }
```

6.2. Задания на одномерный массив

1. Из последовательности чисел $a_1, a_2, ..., a_n$ выбрать отрицательные элементы, подсчитать их число и переписать подряд в массив x.
2. Вычислить компоненты векторов $\boldsymbol{a}(a_1,a_2,...,a_n)$ и $\boldsymbol{b}(b_1,b_2,...,b_n)$ по формулам $a_i = \operatorname{arctg}\dfrac{i+2}{n}$; $b_i = i + \cos(i)$ и вывести на печать. Из векторов $\boldsymbol{a}$ и $\boldsymbol{b}$ получить вектор $\boldsymbol{c}(a_1, b_1, a_2, b_2, ..., a_n, b_n)$, компоненты которого пронумеровать по порядку от 1 до $2n$.
3. Из последовательности чисел $y_1, y_2, ,..., y_n$ выбрать элементы, делящиеся на 3. Подсчитать их число и вывести их порядковые номера в массиве.

4. Вычислить значения компонент вектора $\boldsymbol{x}(x_1, x_2, \ldots, x_n)$ по формуле

$$x_i = \begin{cases} \operatorname{arctg}\dfrac{\sqrt{i}+2}{n+3}, \text{если } \operatorname{tg}(n \cdot i) < 0, \\ \cos(i) \cdot e^{i \cdot \cos(n)}, \text{если } \operatorname{tg}(n \cdot i) \geq 0. \end{cases}$$

Выбрать из массива x положительные компоненты и переписать их в массив $b(b_1,b_2,...,b_n)$.

5. Из последовательности чисел $a_1, a_2, ..., a_n$ выбрать числа, равные нулю, подсчитать их число, а оставшиеся числа вывести в одну строку.
6. Вычислить компоненты вектора $\boldsymbol{x}(x_1, x_2, \ldots, x_n)$ по формуле $x_i = a \cdot \sin(0{,}5 \cdot i) + b \cdot \cos(i\text{-}1)$, где $a = 2{,}2$, $b = -0{,}4$, и вывести на печать. Далее преобразовать полученный вектор следующим образом: все отрицательные компоненты увеличить на 0,5, а положительные заменить единицей. Преобразованный вектор вывести на экран.
7. Дан ряд чисел x_i ($i = 1, 2, \ldots, n$). Вычислить сумму элементов с нечетными и сумму элементов с четными номерами.
8. Дан ряд чисел $\boldsymbol{x}(x_1, x_2, \ldots, x_n)$. Выбрать из этой совокупности целые числа и для них подсчитать M по формуле $M = \dfrac{\sqrt{\sum_{k=1}^{n} x_k^2}}{N}$.
9. Дан вектор $\boldsymbol{y}(y_1, y_2, ,\ldots, y_n)$. Найти и вывести максимальную компоненту и ее порядковый номер.
10. Дан ряд чисел $b_1, b_2,..., b_n$. Выбрать из них положительные числа и найти среднее арифметическое квадратов этих чисел.
11. Дана числовая последовательность $x_1, x_2, \ldots, x_m$. Разбить ее на последовательность $b_1, b_2, ..., b_k$ отрицательных и последовательность $a_1, a_2, ..., a_n$ неотрицательных чисел ($n = k+m$).
12. Дан ряд чисел y_i ($i = 1, 2, \ldots, n$). Записать в него +1 вместо максимального элемента и –1 вместо минимального элемента массива y.

13. Дан вектор $\boldsymbol{x}(x_1, x_2, \ldots, x_n)$. Пронормировать его по своей длине, т. е. получить новые компоненты по формуле $x'_k = \dfrac{x_k}{\sqrt{\sum_{i=1}^{n} x_i^2}}$.

14. Даны два вектора $\boldsymbol{a}$ и $\boldsymbol{b}$ по n компонент в каждом. Найти максимальную разность соответствующих компонент и порядковый номер этих компонент.

15. Дан вектор $\boldsymbol{c}(c_1, c_2, \ldots, c_n)$. Найти наибольшую и наименьшую из компонент вектора и переставить их местами.

16. Дан произвольный ряд чисел a_k ($k = 1, 2, \ldots, n$). Подсчитать число перемен знаков в этом ряду (числа, равные нулю, пропускать).

17. Вычислить компоненты вектора $\boldsymbol{c}(c_1, c_2, \ldots, c_n)$, каждая компонента которого определяется по формуле $c_i = b_i\, a_i$. Далее переписать вектор $\boldsymbol{c}$ в обратном порядке. Вектора $\boldsymbol{a}$ и $\boldsymbol{b}$ заданы.

18. Дана последовательность чисел $x_1, x_2, \ldots, x_n$. Выбрать из них принадлежащие отрезку $[a, b]$ и записать в виде компонент вектора $\boldsymbol{p}$.

19. Дана последовательность чисел $c_1, c_2, \ldots, c_n$. Записать в виде компонент вектора $\boldsymbol{s}$ те числа последовательности, для которых $|c_k| > 35$, и вывести их число.

20. Даны: n точек, координаты которых заданы массивами $\boldsymbol{x}(x_1, x_2, \ldots, x_n)$ и $\boldsymbol{y}(y_1, y_2, \ldots, y_n)$, и окружность, определяемая формулой $(x-a)^2 + (y-b)^2 = r^2$. Вывести номера точек, лежащих внутри окружности, и подсчитать их количество.

6.3. Двумерный массив

Некоторые замечания по поводу обозначения массивов в Си

Существует большой класс задач, решение которых основано на знании выводов векторной алгебры (математики). Наиболее известной из таких задач является разрешение системы линейных уравнений. Для того, чтобы с наименьшими затратами можно было перенести решения (формулы с индексными переменными) различных задач из векторной алгебры в программу, разработчики языка Си стремились максимально сблизить форму записи математических выражений на этом языке к той, что используется в математике. Так, если в математике для описания элемента матрицы $\boldsymbol{A}$ используется запись типа a_{ij}, то в Си для описания элемента двумерного массива – a[i][j].

Ввод и вывод двумерных массивов

Первое, с чем сталкивается программист при работе с массивами – это их ввод и вывод. В Си для ввода элементов массива через клавиатуру предлагается функция scanf(), а для вывода на экран – функция printf(). Следует, однако, добавить, что среди стандартных функций ввода/вывода существуют и другие способы ввода/вывода массивов. Один из таких способов ввода будет рассмотрен ниже.

Рассмотрим ввод двумерного массива с помощью клавиатуры подробнее. Для этого нам понадобится использовать два цикла, первый из которых для определенности назовем циклом по строкам, а второй цикл, вложенный в первый – циклом по столбцам. Примем, что в обозначении массива a[i][j] (так же, как в математике) значение первой переменной *i* будет соответствовать номеру строки, а второй переменной *j* – номеру столбца. В качестве примера ввода и вывода элементов массива может служить следующая программа:

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

```
int main()
{
    const int Ni = 5, Nj = 10;
    int i,j;
    float a[Ni][Nj];

    printf("вводим массив a\n");
    /* Напоминание: индексация в массивах
    ВСЕГДА начинается с 0 */
    for (i=0; i< Ni; i++)
        for (j=0;j< Nj; j++)
        {
            printf("a[ %d , %d ]=", i,j);
            scanf("%f",&a[i][j]);
        }
    printf("Ввод массива окончен \n");

    printf("Вывод массива\n");
    for (i=0; i< Ni; i++)
        for (j=0;j< Nj; j++)
          printf("a[ %d , %d ]=%f \n", i, j, a[i][j]);
    return 0;
}
```

Несмотря на простоту приведенной программы, отметим, что внутренние или вложенные циклы по переменной *j* выполняются без остановок, тогда как внешние циклы по *i* "дожидаются" очередного окончания своего внутреннего цикла. Иными словами, сначала значение *i* становится равным 1, затем *j* "про-

бегает" значения от 1 до N_j, затем *i* становится равным 2 и вновь *j* "пробегает" значения от 1 до N_j, далее *i* уже равняется 3 и т.д.

Здесь следует отметить бросающуюся в глаза "негибкость" такого способа работы с массивами, для которых заранее неизвестно число элементов (их размер или "глубина") по любой из размерностей массива. Иными словами, идентификаторы N_i и N_j не могут быть переменными, значения которых можно вводить с клавиатуры (как, например, значения элементов массива) во время исполнения программы.

Отметим также, что отладка программы, использующей приведенную выше программу, будет, скорее всего, неудобной. При запуске такой программы от вас потребуется не унывать, когда после очередного ввода 50-ти чисел вы обнаружите, что остальная часть программы работает неверно. Поэтому при отладке программ рекомендуется другой способ ввода массива. Он называется начальной инициализацией массива. Для этого в отлаживаемой программе при объявлении массива можно сразу задать начальные значения его элементов:

```
const int Ni = 5, Nj = 10;
float a[Ni][Nj]=
{{0.1, 0.5, 1.1, 1.5, 3.3, 0.1, 0.2, 8.9, 0.9, 1.0},
 {3.3, 6.1, 8.8, 3.7, 4.1, 9.1, 1.1, 5.2, 8.8, 4.1},
 {4.4, 6.5, 1.7, 3.6, 2.6, 5.9, 4.8, 2.7, 8.7, 2.0},
 {8.1, 3.2, 2.1, 9.1, 4.4, 7.7, 5.2, 1.9, 3.6, 5.9},
 {5.8, 8.2, 5.2, 7.4, 9.5, 0.8, 4.2, 2.4, 6.9, 8.0}};
```

В этом случае начальными значениями элементов объявленного массива будут не произвольные числа, а числа, заключенные между фигурными скобками. Например, первое число в первой строчке заносится в элемент `a[0,0]`, второе - в `a[0,1]`, последнее в `a[0,9]`, первое число во второй строке – в `a[1,0]` и т.д.

Примеры программ с использованием двумерных массивов

Задача 1. Дана квадратная матрица ***A***. Написать программу, которая вычисляет сумму всех элементов матрицы ***A*** и сумму ее элементов, расположенных на главной диагонали.

-1	23	14	-4
2	38	65	47
29	53	29	-8
13	54	17	55

```
#include <stdio.h>

int main()
{
    const int N = 4;
    int A[N][N] =   {{ -1, 23, 14,-34},
                     { 52, 38,-65, 47},
                     { 29, 53,-29, -8},
                     { 13,-54, 17, 55}};
    int Diag_Sum, All_Sum, i,j;
    All_Sum= 0;
    Diag_Sum= 0;
    for(i=0; i<N; i++)
        for( j=0; j< N; j++)
        {
            All_Sum = All_Sum + A[i][j];
            if (i==j) Diag_Sum = Diag_Sum + A[i][j];
        }
```

```
        printf("Сумма элементов матрицы = %d \n",All_Sum);
        printf("Сумма элементов главной диагонали = %d \n",
Diag_Sum);
        return 0;
    }
```

Программа получилась довольно простая и не требует дополнительных комментариев.

Задача 2. Для заданного вещественного значения *x* вычислить элементы квадратной матрицы ***A***, имеющей следующий вид:

$$A = \begin{vmatrix} 1 & x^1 & x^2 & x^3 \\ x^1 & x^2 & x^3 & x^4 \\ x^2 & x^3 & x^4 & x^5 \\ x^3 & x^4 & x^5 & x^6 \end{vmatrix}$$

Перед составлением алгоритма отметим характерную особенность матрицы ***A***: степень основания *x* каждого элемента матрицы равна сумме номера строки *i* и столбца *j*, на пересечении которых размещен данный элемент (при условии отсчета индексов от нуля). Например, для элемента, расположенного на пересечении третьей строки и второго столбца, имеем $a(3,2) = x^{3+2}$. В общем виде можно записать представление элемента матрицы ***A*** как $a(i,j) = x^{i+j}$.

Самый простой способ решения поставленной задачи – это непосредственно воспользоваться последней формулой. Однако в некоторых версиях языка Си отсутствует функция возведения в степень. Для возведения числа в произвольную степень можно воспользоваться известной формулой $a^b = e^{b \cdot \ln(a)}$, и тогда программа решения задачи может выглядеть следующим образом:

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

```
int main()
{
int i, j;
float x, a[3][3];
    printf("введите x > ");
    scanf("%f",&x);
    for(i=0; i< 3; i++)
    {
        for(j=0; j< 3; j++)
        {
            a[i][j]= exp( (i+j)*log(x) );
            printf("%10.3f", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Представленный вариант программы не является оптимальным, так как возведение в степень, связанное с вычислением элементарных функций exp(x) и ln(x), займет больше времени и будет менее точным (мало того, значения переменной x не могут быть отрицательными) по сравнению с многократным умножением. Поэтому, если вы дорожите скоростью и точностью вычислений, то вычисления необходимо проводить по рекуррентной формуле. Для этого в новой программе понадобятся еще две новые переменные bi и bj, необходимые для "накопления произведения" во внешнем и внутреннем циклах соответственно.

```
#include <stdio.h>

int main()
{
    int i, j;
    float x, bi, bj, a[3][3];
    printf("введите x > ");
    scanf("%f",&x);
    bi=1;
    for(i=0; i< 3; i++)
    {
        bj=bi;
        for(j=0; j< 3; j++)
        {
            a[i][j]= bj;
            printf("%10.3f", a[i][j]);
            bj=bj*x;
        }
        bi=bi*x;
        printf("\n");
    }
    return 0;
}
```

Накопление произведения в приведённой программе происходит по формуле типа $b=b * x$, где x – вводимый с клавиатуры множитель, b – накапливающий коэффициент.

Последняя программа с многократным умножением все равно не является оптимальной по скорости выполнения, так как матрица является симметрической (симметричной относительно главной диагонали) и достаточно вычислить элементы выше (или ниже) главной диагонали. Попробуйте усовершенствовать программу так, чтобы в матрице ***A*** считались лишь элементы, расположенные на главной диагонали и выше. Элементы ниже главной диагонали заполнить значениями «симметричных» элементов.

6.4. Задания на двумерный массив

1. Найти наибольший элемент a_{ik} главной диагонали квадратной матрицы ***A*** и вывести на экран всю строку, в которой он находится (для элементов главной диагонали $i = k$).
2. Вычислить сумму элементов b_{ik} квадратной матрицы ***B***, расположенных над главной диагональю (для элементов главной диагонали $i = k$).
3. Из квадратной матрицы ***X*** построить матрицу ***Y***, заменив строки столбцами (транспонирование матрицы). Исходную и полученную матрицу вывести на экран.
4. Определить количество положительных и отрицательных элементов матрицы ***A***.
5. Определить количество положительных элементов x_k каждого столбца матрицы ***X*** и переписать их в одномерный массив (вектор) ***M***.
6. Найти наибольший элемент a_{ik} матрицы ***A***, элементы которой вычисляются по формуле $a_{ik} = x_i * y_k$. Векторы $\boldsymbol{X} = \{x_1, x_2, ..., x_i\}$ и $\boldsymbol{Y} = \{y_1, y_2, ..., y_k\}$ считаются заданными.
7. Найти минимальные элементы x_{ik} в каждой строке квадратной матрицы ***X*** и поместить их на главной диагонали, а диагональные элементы записать на место минимальных (для элементов главной диагонали $i = k$).
8. Найти наименьший элемент x_{ik} матрицы ***X***. Затем в строке и в столбце, где он находился, все элементы заменить нулями. Исходную и полученную матрицу вывести на экран.

9. Вычислить сумму положительных элементов a_{ik} каждой строки матрицы $\boldsymbol{A}$. Найти наибольшую из этих сумм, а также номер этой строки.
10. Выделить положительные и отрицательные элементы x_{ik} главной диагонали квадратной матрицы $\boldsymbol{X}$. Записать их в два одномерных массива (вектора) $\boldsymbol{P} = \{p_1, p_2, ...\}$ и $\boldsymbol{N} = \{n_1, n_2, ...\}$ и вывести их на экран.
11. Найти среднее арифметическое положительных элементов x_{ik} каждого столбца матрицы $\boldsymbol{X}$ при условии, что в каждом столбце есть хотя бы один элемент больше нуля.
12. В матрице $\boldsymbol{A}$ найти столбец a_k, сумма элементов a_{ki} которого минимальна. Вывести номер столбца, его элементы и их сумму.
13. Дана матрица $\boldsymbol{C}$. В каждой строке переставить максимальный c_{ik} и минимальный c_{ij} элементы.
14. В прямоугольной матрице $\boldsymbol{X}$ найти разность между наибольшим x_{ik} и наименьшим x_{jn} элементами.
15. Дана прямоугольная матрица $\boldsymbol{A}$ размерности $m \times n$, получить вектор $\boldsymbol{B} = \{b_1, b_2, ..., b_m\}$ из максимальных элементов строк.
16. Пронормировать матрицу $\boldsymbol{X}$ по ее максимальному элементу, т.е. каждый элемент матрицы x_{ik} разделить на максимальный. Вывести на экран максимум и номера строки и столбца, где он находился. Вывести нормированную матрицу.
17. Дана матрица $\boldsymbol{A}$ размером $m \times n$ и вектор $\boldsymbol{Y} = \{y_1, y_2, ..., y_n\}$. Найти вектор $\boldsymbol{P}$, равный произведению вектора $\boldsymbol{Y}$ на матрицу $\boldsymbol{A}$. Формула векторного умножения выглядит следующим образом:

$$p_i = \sum_{k=1}^{n} a_{ik} \cdot y_k,$$

где $i = 1, 2, ... m$.

18. Даны квадратные матрицы $\boldsymbol{A}$ и $\boldsymbol{B}$ размером $n \times n$. Выполнить их перемножение $\boldsymbol{C}=\boldsymbol{A} \cdot \boldsymbol{B}$ по традиционному правилу – строки первой матрицы поочередно скалярно умножаются на столбцы второй:

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} \cdot b_{k,j}$$

19. Заполнить квадратный массив ***A*** размером (10×10) следующим образом:

а)

$$\begin{vmatrix} 1 & 2 & 3 & \dots & 10 \\ 11 & 12 & 13 & \dots & 20 \\ 21 & 22 & 23 & \dots & 30 \\ \dots & \dots & \dots & \dots & \dots \\ 91 & 92 & 93 & \dots & 100 \end{vmatrix}$$

б)

$$\begin{vmatrix} 1 & 2 & 3 & \dots & 10 \\ 0 & 1 & 2 & \dots & 9 \\ 0 & 0 & 1 & \dots & 8 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{vmatrix}$$

20. Дана квадратная матрица ***A*** размером ($n \times n$), где n – нечетное число. Осуществить несколько поворотов этой матрицы вокруг его центра на 90° против часовой стрелки.

21. Определить, является ли целая квадратная матрица ($n \times n$) ортонормированной, т.е. такой, в которой скалярное произведение каждой пары различных строк равно 0, скалярное произведение каждой строки на себя равно 1.

22. Дана вещественная матрица ***A*** размером (8×5). Переставляя ее строки и столбцы, добиться того, чтобы наибольший элемент оказался в верхнем левом углу.

23. Имеются несколько годовых рядов среднемесячных температур для различных городов (не менее 6). Найти среднегодовую температуру в каждом городе и определить, какой месяц в каждом городе самый холодный.

7. РЕШЕНИЕ СИСТЕМЫ ЛИНЕЙНЫХ АЛГЕБРАИЧЕСКИХ УРАВНЕНИЙ МЕТОДОМ ГАУССА

Система n линейных алгебраических уравнений может быть представлена в виде:

$$\begin{aligned} a_{11}\,x_1 + a_{12}\,x_2 + \ldots + a_{1n}\,x_n &= b_1 = a_{1,n+1}, \\ a_{21}\,x_1 + a_{22}\,x_2 + \ldots + a_{2n}\,x_n &= b_2 = a_{2,n+1}, \\ &\ldots\ldots\ldots\ldots\ldots \\ a_{n1}\,x_1 + a_{n2}\,x_2 + \ldots + a_{nn}\,x_n &= b_n = a_{n,n+1}, \end{aligned} \qquad (6.1)$$

где x_i – неизвестные величины, a_{ij} – коэффициенты системы, b_i – свободные члены, которые здесь можно трактовать как элементы $a_{i,n+1}$ расширенной матрицы коэффициентов.

Решением системы (6.1) называется совокупность чисел $x_1, x_2, \ldots, x_n$, превращающая уравнения системы в тождества. Система имеет единственное решение, если определитель матрицы коэффициентов a_{ij} не равен нулю.

Метод Гаусса, называемый также методом последовательного исключения неизвестных, является одним из наиболее распространенных методов решения системы линейных алгебраических уравнений. Алгоритм реализации метода можно разделить на два этапа.

Первый этап называется прямым ходом и начинается с того, что первое уравнение системы (6.1) преобразуется к виду:

$$x_1 + a_{12}\,x_2 + a_{13}\,x_3 + \ldots + a_{1n}\,x_n = a_{1,n+1}, \qquad (6.2)$$

т. е. каждый коэффициент первой строки делится на первый коэффициент, являющийся элементом главной диагонали:

$$a_{1j} = a_{1j} / a_{11}. \qquad (6.3)$$

Очевидно, что (6.3) осуществимо, если $a_{11} \neq 0$. Поскольку конкретная система может иметь нулевой коэффициент a_{11}, в модифицированном методе Гаусса («Метод Гаусса с выбором главного элемента») перед выполнением деления в системе (6.1) находится ненулевой и максимальный по абсолютной величине коэффициент a_{i1}. Уравнение, содержащее этот коэффициент, меняется местами с первым уравнением. Это исключает деление на ноль, а также уменьшает возможную погрешность вычислений, связанную с делением на число, близкое к нулю.

Из уравнения (6.2) следует, что

$$x_1 = (a_{1,n+1} - a_{12}\, x_2 - \ldots - a_{1n}\, x_n) / a_{11} \,. \qquad (6.4)$$

Подстановка (6.4) в последующие уравнения системы позволяет исключить из них члены, содержащие x_1. Нетрудно показать, что такая подстановка эквивалентна преобразованию коэффициентов расширенной матрицы коэффициентов по формуле:

$$a_{ij} = a_{ij} - a_{i1}\, a_{1j} \text{ , где } \quad i = 2, 3, \ldots, n; \quad j = 1, 2, \ldots, n+1. \qquad (6.5)$$

Далее описанная выше процедура повторяется для x_2 из второго уравнения и т. д. В результате мы получаем равносильную исходной систему линейных уравнений

$$\begin{aligned}
&x_1 + a_{12}\, x_2 + a_{13}\, x_3 + \ldots + a_{1n}\, x_n = a_{1,n+1} \\
&\qquad x_2 + a_{23}\, x_3 + \ldots + a_{2n}\, x_n = a_{2,n+1} \qquad (6.6) \\
&\qquad\qquad \ldots\ldots\ldots\ldots\ldots \\
&\qquad\qquad\qquad x_n = a_{n,n+1} \,.
\end{aligned}$$

с треугольной матрицей преобразованных коэффициентов.

Второй этап решения системы линейных алгебраических уравнений методом Гаусса, называемый обратным ходом, заключается в последовательном определении неизвестных x_i по формулам (6.6) снизу вверх, начиная с x_n и заканчивая x_1.

На рис. 7.1 приводится структурограмма алгоритма решения системы линейных алгебраических уравнений методом Гаусса с выбором главного элемента:

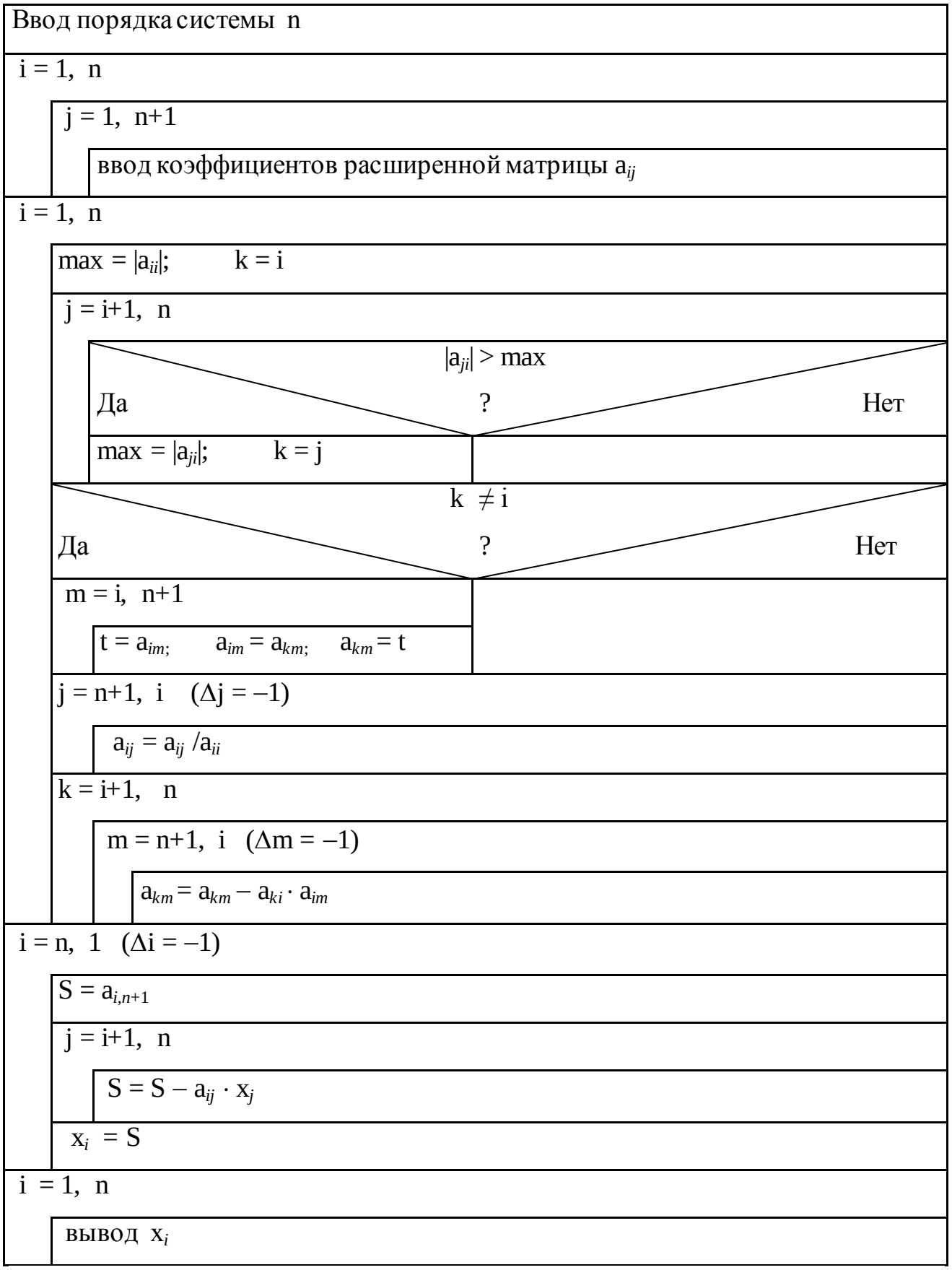

Рис. 7.1. Алгоритм решения линейных алгебраических уравнений методом Гаусса

7.1. Задания на решение системы линейных уравнений методом Гаусса

При выполнении задания необходимо ввести в программу проверку правильности полученного решения путем подстановки полученных значений x_i в наиболее полное уравнение исходной системы.

1. $4x_1 + 2x_2 - 3x_3 = -2,$
   $2x_1 + 8x_2 - x_3 = 8,$
   $9x_1 + x_2 + 8x_3 = 0.$

2. $x_1 + x_2 - 3x_3 + 2x_4 = 6,$
   $x_1 - 2x_2 - x_4 = -6,$
   $x_2 + x_3 + 3x_4 = 16,$
   $2x_1 - 3x_2 + 2x_3 = 6.$

3. $x_1 + 2x_2 - x_3 = 8,$
   $x_2 + 3x_3 + x_4 = 15,$
   $4x_1 + x_3 + x_4 = 11,$
   $x_1 + x_2 + 5x_4 = 23.$

4. $36,47x_1 + 5,28x_2 + 6,34x_3 = 12,26,$
   $7,33x_1 + 28,74x_2 + 5,86x_3 = 15,15,$
   $4,63x_1 + 6,31x_2 + 26,17x_3 = 25,22.$

5. $2x_1 + x_2 + x_3 = 5,$
   $x_1 - 2x_2 + x_3 = -5,$
   $-7x_1 + x_2 - x_3 = 10.$

6. $x_1 + x_2 - x_3 + x_4 = 4,$
   $2x_1 - x_2 + 3x_3 - 2x_4 = 1,$
   $x_1 - x_3 + 2x_4 = 6,$
   $3x_1 - x_2 + x_3 - x_4 = 0.$

7. $3{,}21x_1 + 0{,}71x_2 + 0{,}34x_3 = 6{,}12,$

$0{,}43x_1 + 4{,}11x_2 + 0{,}22x_3 = 5{,}71,$

$0{,}17x_1 + 0{,}16x_2 + 4{,}73x_3 = 7{,}06.$

8. $0{,}04x_1 - 0{,}08x_2 + 4{,}00x_3 = 20,$

$4{,}00x_1 + 0{,}24x_2 - 0{,}08x_3 = 8,$

$0{,}09x_1 + 3{,}00x_2 - 0{,}15x_3 = 9.$

9. $3x_1 - x_2 + x_3 + 2x_5 = 18,$

$2x_1 - 5x_2 + x_4 + x_5 = -7,$

$x_1 - x_4 + 2x_5 = 8,$

$2x_2 + x_3 + x_4 - x_5 = 10,$

$x_1 + x_2 - 3x_3 + x_4 = 1.$

10. $3x_1 + 2x_2 + x_3 = 4,$

$x_1 + x_2 - x_3 = 1,$

$x_1 - 2x_2 + x_3 = 3.$

11. $2x_1 + 2x_2 + x_3 = 14,$

$10x_1 + x_2 + x_3 = 12,$

$2x_1 + 10x_2 + x_3 = 13.$

12. $x_1 - 1{,}3x_2 + 3{,}9x_3 - 3{,}7x_4 = 3{,}1,$

$-0{,}5x_1 + x_2 - 3{,}1x_3 - 4x_4 = -12,$

$2x_1 - 0{,}8x_2 - x_4 = 1,$

$3x_1 + 1{,}5x_2 - x_3 + 2{,}4x_4 = 6.$

13. $x_1 - 10x_2 - x_3 + 2x_4 = 0,$

$2x_1 + 3x_2 + 20x_3 - x_4 = -10,$

$10x_1 - x_2 + 2x_3 - 3x_4 = 0,$

$3x_1 + 2x_2 + x_3 + 20x_4 = 15.$

14. $1{,}76x_1 - 3{,}12x_2 + 9{,}38x_3 = 1{,}93,$

$5{,}92x_1 - 1{,}24x_2 - 1{,}84x_3 = 2{,}44,$

$2{,}72x_1 - 9{,}71x_2 + 2{,}43x_3 = 2{,}4.$

15. $9{,}28x_1 - 79{,}6x_2 - 4{,}92x_3 = -25{,}8,$

$68{,}3x_1 - 2{,}71x_2 - 8{,}14x_3 = -32{,}6,$

$10{,}2x_1 + 6{,}07x_2 - 9{,}1x_3 = -50{,}3.$

ЛИТЕРАТУРА

1. Хуторова О. Г., Стенин Ю. М., Фахртдинов Р. Х., Зыков Е. Ю., Журавлев А. А. Практикум по информатике. Программирование на языке Си. Учебно-методическое пособие. Казань: Казанский государственный ун-т, 2009. – 46 с.

2. Программирование на языке Си / А.В.Кузин, Е.В.Чумакова - М.: Форум, НИЦ ИНФРА-М, 2015. - 144 с.

3. Царев, Р. Ю. Программирование на языке Си [Электронный ресурс] : учеб. пособие / Р. Ю. Царев. – Красноярск : Сиб. федер. ун-т, 2014. – 108 с. - Режим доступа: http://znanium.com/catalog/product/510946

4. Программирование на языке высокого уровня. Программирование на языке С++: учеб. пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев ; под ред. Л.Г. Гагариной. — М. : ИД «ФОРУМ» : ИНФРА-М, 2018. — 512 с

5. Язык Си++ : учебное пособие для студентов высших учебных заведений, обучающихся по направлениям "Прикладная математика" и "Вычислительные машины, комплексы, системы и сети" / В. В. Подбельский . 5-е изд. Москва : Финансы и статистика, 2008 . 559 с.

6. Колдаев В.Д. Численные методы и программирование: Учебное пособие / В.Д. Колдаев; Под ред. Л.Г. Гагариной. - М.: ИД ФОРУМ: НИЦ ИНФРА-М, 2014. - 336 с. - Режим доступа: - http://znanium.com/bookread.php?book=452274

7. Демидович Б.П. Основы вычислительной математики./ Демидович Б.П. , Марон И.А.; – М.: изд. Лань, 2009. – 664 с.

8. Керниган Б. Язык программирования Си./ Керниган Б., Ритчи Д. — 2-е изд. – М.: «Вильямс», 2007. – 304 с.

9. Белецкий Ян. Энциклопедия языка СИ./Пер. с пол. Под ред. Ф.Ф.Пащенко. – М: Мир, 1992. – 686 с.

10. Р. Хазфилд, К. Лоуренс и др. Искусство программирования на С. Фундаментальные алгоритмы, структуры данных и примеры приложений. Энциклопедия программиста: Пер. с англ./Р. Хэзфилд, Л. Кирби и др. – М.: Изд. «Диа-Софт», 2001. – 736 с.