**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

федеральное государственное бюджетное образовательное учреждение высшего образования «Казанский национальный исследовательский технический
университет им. А.Н. Туполева-КАИ»
(КНИТУ-КАИ)
Чистопольский филиал «Восток»

# МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

по дисциплине

# ОСНОВЫ ПРОГРАММИРОВАНИЯ

Индекс по учебному плану: **Б1.О.12.01**
Направление подготовки: **09.03.01 Информатика и вычислительная техника**
Квалификация: **Бакалавр**
Профиль подготовки: **Вычислительные машины, комплексы, системы и сети**
Типы задач профессиональной деятельности: **проектный, производственно-технологический**

Рекомендовано УМК ЧФ КНИТУ-КАИ

Чистополь
2023 г.

**Лабораторная работа 1**

**Программирование линейных алгоритмов**

1. Изучить работу со средой CodeBlocks.
2. Написать программу, вычисляющую выражение по варианту.

Вариант 1

1) $\frac{x^2-7x+10}{x^2-8x+12}$ 2) $\frac{1-\cos 2x+\sin 2x}{1+\cos 2x+\sin 2x}$

3) Заданы координаты трех вершин треугольника ($x_1$,$y_1$), ($x_2$,$y_2$), ($x_3$,$y_3$).Найти его периметр и площадь.

Вариант 2

1) $\frac{5x-10y}{2y}:\frac{6y-3x}{8y^2}$ 2) $\frac{\sin(a+b)-\cos a\cdot\sin b}{\cos(a-b)-\sin a\cdot\sin b}$

3) Дана сторона равностороннего треугольника. Найти площадь этого треугольника, его высоты, радиусы вписанной и описанной окружностей.

Вариант 3

1) $\frac{5b-10}{4-4b-b^2};$ 2) $\ln(x^3+4\sqrt{x}+5)$

Найти сумму членов арифметической прогрессии, если известны ее первый член, знаменатель и число членов прогрессии.

Вариант 4

1) $xy-\frac{x^3-y^3}{x^2+2xy+y^2}$ 2) $x^3\ln x+\ln(4\sqrt{x}+5)$

3) Вычислить высоты треугольника со сторонами a, b, c.

Вариант 5

1) $x-\frac{x^3}{3}+\frac{x^5}{5}$ 2) $\frac{3+e^{y-1}}{1+x^2|y-tgz|}$

3)Даны два числа. Найти среднее арифметическое кубов этих чисел и среднее геометрическое модулей этих чисел.

Вариант 6

1) $\frac{a+ab}{a^2+ab}+\frac{a^2+b^2}{a+b};$ 2) $\frac{1+\sin 2x}{(\sin x-\cos x)^2}$

3)Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.

Вариант 7

1) $\frac{x^2-xy}{9x^2-18xy+9y^2}+\frac{a^4-a}{3a^2+3a+3};$ 2) $\ln(\sin 3x+2^x)$

3) Составить программу перевода радианной меры угла в градусы, минуты и секунды.

Вариант 8

1) $\frac{3y-2y^2}{4y^2-12y+9};$ 2) $\ln\frac{2-x}{3x+5}$

1. Найти площадь равнобедренной трапеции с $\alpha$ основаниями a и b и углом при большем основании a.

Вариант 9

1) $\frac{x^2-y^2}{(6y-6x)^2};$ 2) $e^{\sin x}+\ln(\sin x)$

3)Дана длина ребра куба. Найти площадь грани, площадь полной поверхности и объем этого куба.

Вариант 10

1) $\frac{9-(b+2)^2}{b^2+10b+25^3}$ 2) $\frac{\ln x^2}{1+\ln^2 x}+\frac{x}{\ln x-1}$

3) Три сопротивления соединены параллельно. Найдите сопротивление соединения.

Вариант 11

1) $\frac{a(b-x)+x(a+b)}{5a+5x};$ 2) $\frac{\sin x}{\cos x}+\frac{\cos x}{1+\sin x}$

3) Найти площадь треугольника, две стороны которого равны a и b, а угол $\gamma$ между этими сторонами равен

Вариант 12

1) $\frac{x+y}{y+1}-\frac{xy-12}{34+x}$

2) $e^x \cdot \sin x + \ln(\sin x)$

3) Дано a. Не используя никаких функций и никаких операций, кроме умножения, получить $a^8$ за три операции и $a^{10}$ за четыре операции.

Вариант 13

1) $\frac{a^2-ab}{a^2-b^2}-\frac{3x-6y}{10y-5x};$

2) $e^{9x}+3e^{-2x}$

3)Вычислить расстояние между двумя точками с данными координатами $x_1$, $y_1$ и $x_2$, $y_2$.

Вариант 14

1) $\frac{x^2-4x+4}{(x+5)^2-49};$

2) $\frac{\sin x+\cos y}{\cos x-\sin y}\cdot tgxy$

3) Вычислить периметр и площадь прямоугольного треугольника по данным длинам двух катетов a и в.

Вариант 15

1) $\frac{10a^2}{a^2-3ab}:\frac{15a^3b}{a-3b};$

2) $\ln\left|\left(y-\sqrt{|x|}\right)\cdot\left(x-\frac{y}{z+\frac{x^2}{4}}\right)\right|$

3)Треугольник задан величинами углов и радиусом описанной окружности. Найти стороны треугольника.

Вариант 16

1) $\frac{a}{c}\cdot\frac{b}{d}-\frac{ab-c}{cd}$

2) $\frac{\ln^2 x}{x}+\frac{\ln x^2}{1+\ln^2 x}$

3)Найти площадь кольца, внутренний радиус которого равен r, а внешний - заданному числу R (R>r).

Вариант 17

1) $\dfrac{(4z-4)^2}{4-8z+8z^2}$ 2) $\dfrac{\ln x^2}{1+\ln^2 x}$

3)Дано x. Получить значения $-2x+3x^2-4x^3$ и $1+2x+3x^2+4x^3$. Позаботиться об экономии операций.

Вариант 18

1) $\dfrac{ac-ab}{b^2-c^2}+\dfrac{b^2+ab}{3a^2-3ab}$; 2) $\sin^3(2x)e^{-x}$

3) Полторы кошки за полтора часа съедают полторы мышки. Сколько мышек съедят X кошек за Y часов.

Вариант 19

1) $\dfrac{x(1-x)+x(x^2-1)}{8x^2}$; 2) $\ln\sqrt{x-5}+\ln\sqrt{2x-3}$

3) Написать программу, которая выводит на экран первые $\pi$ четыре степени числа

Вариант 20

1) $\dfrac{a(a-b)+2ab}{(a+b)c}$; 2) $x\ln^2(x)+\dfrac{x}{\ln x-1}$

3) Вычислить длину окружности и площадь круга одного и того же заданного радиуса R.

Вариант 21

1) $\dfrac{ab}{a^2-b^2}-\dfrac{b}{2a-3b}$ 2) $\dfrac{2^x}{\cos x}+3^x$

3) Составить программу вычисления объема цилиндра и конуса, которые имеют одинаковую высоту H и одинаковый радиус основания R.

Вариант 22

1) $\dfrac{cy-2c+2(y+c)}{7(c+2)}$; 2) $\dfrac{\sin x}{\ln(7x)}+3\ln(5x)$

3)Даны два действительных числа x и y. Вычислить их сумму, разность, произведение и частное.

Вариант 23

1) $\frac{2a-4b}{3a+6b}-\frac{x^3-1}{x^2-1};$ 2) $\ln^3 x+3\ln x$

3)Вычислить корни квадратного уравнения $ax^2+bx+c=0$, заданного коэффициентами a,b и c (предполагается, что a не равно нулю и что дискриминант уравнения неотрицателен).

Вариант 24

1) $\frac{x+y}{x^2-xy}-\frac{3x+y}{y^2-x^2}$ 2) $\sqrt{\frac{1-\cos a}{1+\cos a}}+\sqrt{\frac{1+\cos a}{1-\cos a}}$

3)Дано действительное число x. Не пользуясь никакими другими арифметическими операциями, кроме умножения, сложения и вычитания, вычислить за минимальное число операций $2x^4-3x^3+4x^2-5x+6$.

Вариант 25

1) $\frac{(a-4)^2-1}{a^2-6a+9};$ 2) $\frac{\ln|\cos x|}{\ln|1+x^2|}-\sin(x+y-1)$

3) Найти (в градусах) все углы треугольника со сторонами a, b, c.

## Лабораторная работа 2

## Программирование простейших циклов на языке Си.

Структура программы

Любая программа на языке Си состоит из одной или более "функций", являющихся основными модулями программы. Одна из функций, с которой начинается выполнение программы, называется главной и всегда носит имя ***main***. Остальные функции – это подпрограммы, которые могут вызываться либо из главной функции, либо из других подпрограмм. Простая программа, состоящая только из функции main, имеет следующую структуру:

```
Директивы препроцессора
main ()
{  Описания переменных
   Операторы
}
```

Заголовок функции - main(). Круглые скобки после имени main как раз и указывают, что это функция. Тело функции заключается в фигурные скобки и состоит из описаний переменных и операторов, описывающих процесс обработки данных.

В программу можно включать комментарии, начинающиеся с пары символов **/*** и заканчивающиеся парой ***/** ( они могут быть везде, где могут быть пробелы ).

Пример простой программы:

```
/* программа сложения двух целых чисел */
   #include <stdio.h>
   main ()
   {  int a, b;          /* описание целочисленных переменных a и b */
      printf ("Задайте два числа: ");       /* вывод сообщения       */
      scanf ("%d %d", &a, &b);              /* ввод значений a и b   */
      printf ("%d + %d = %d\n", a, b, a+b); /* вывод результата      */
```

}

При выполнении этой программы на экране появится сообщение:

***Задайте два числа:***

и затем программа будет ждать, пока вы не введете числа (ввести нужно в той же строке, разделяя числа пробелом). Например:

***Задайте два числа: 328 54***

Затем появится результат в виде:

***328 + 54 = 382***

В этой программе директива препроцессора ***#include <stdio.h>*** служит для включения в программу библиотечного файла stdio.h, содержащего объявления стандартных функций ввода/вывода, таких как ***printf, scanf***. Тело функции main содержит три оператора вызова функций printf и scanf.

Обратите внимание, что все ключевые слова в языке Си пишутся строчными буквами, директивы препроцессора начинаются с первой позиции строки, а операторы можно размещать с любой позиции. Для наглядности принята ступенчатая форма записи программы.

**Описания переменных и основные типы данных**

При описании переменных указываются имена переменных и типы значений этих переменных:

тип_1 имя_1;

тип_2 имя_2;

**Имя (идентификатор)** - это последовательность латинских букв и цифр, начинающаяся с буквы. Если несколько переменных имеют один и тот же тип, то их можно описать вместе, перечислив имена через запятую:

тип имя_1, имя_2, ... ;

К основным типам данных относятся целые числа (***int, short, long, unsigned***), символы (***char***) и вещественные числа или числа с плавающей точкой (***float, double***).

Примеры описаний переменных:

```
float   x,y,z;       /* вещественные числа                  */
double  x1,x2;    /* вещ. числа двойной точности          */
char    simv;      /* символ                               */

int     i,j;          /* целые числа                       */
long    summa;    /* длинное целое                        */
short   k1,k2;      /* короткие целые                     */
unsigned count;   /* беззнаковое целое (неотрицательное) число */
```

Объем памяти, занимаемой данными различных типов, зависит от типа ЭВМ и конкретной реализации языка Си.

При описании переменной можно инициализировать переменную, например:

```
int  k=0;   /* k присваивается начальное значение 0 */
```

Типы используемых в программе констант определяются по их виду, например:

123  -65  - целые константы;

-34.6  3.14159  .12E-5  7e4  - константы с плавающей точкой

(.12E-5=.0000012  7e4=70000.);

'A'  'a'  '2'  '%'  - символьные константы.

Рассмотренные типы являются простыми. Более сложные структурированные типы данных, а также описание нестандартных типов данных будут рассмотрены позднее.

## Определение символических констант

Часто возникает необходимость использовать в программе именованные константы. Использование символических имен вместо значений делает программу более понятной. Для определения символических констант служит директива препроцессора ***#define***. В начало программы до или после директив #include для каждой константы нужно добавить строку вида:

***#define имя значение***

Например:

```
#define  PI          3.14159
#define  RADIUS  16.75
```

Обратите внимание на прописные буквы в именах констант. По традиции символические константы пишутся прописными буквами в отличие от имен переменных. Конечно, вы можете написать константы и строчными буквами, но при этом вы должны чувствовать свою вину, поскольку нарушаете традицию.

## Операторы

### Оператор присваивания

Оператор служит для присвоения переменной значения и имеет формат:

*переменная = выражение;*

При выполнении оператора вычисляется значение выражения и присваивается переменной.

Примеры:

```
x=0.1;
i=i+1;
y=(sin(x)-10)*x;
k=n % 3;
```

Выражение может состоять из операндов - переменных, констант и вызовов функций, круглых скобок и знаков операций **+** (сложение), **-** (вычитание), ***** (умножение), **/** (деление), **%** (вычисление остатка от целочисленного деления), **++** (увеличение на 1), **--** (уменьшение на 1) и некоторых других.

Операции ***, /, %** имеют более высокий приоритет, чем **+** и **-**. Операции с одинаковым приоритетом выполняются слева направо, если нет скобок.

Операндами операции % должны быть значения целого типа , результат имеет тот же тип.

Некоторые математические стандартные функции:

**abs(x),fabs(x)** - вычисляется абсолютное значение x;

**atan(x)** - вычисляется арктангенс x;

**tan(x)** - вычисляется тангенс x; x задается в радианах;

**acos(x)** - вычисляется арккосинус x;

**cos(x)** - вычисляется косинус x; x задается в радианах;

**asin(x)** - вычисляется арксинус x;

**sin(x)** - вычисляется синус x; x задается в радианах;

**exp(x)** – число ***e*** (≈2.7) возводится в степень x;

**log(x)** - вычисляется натуральный логарифм x;

**log10(x)** - вычисляется десятичный логарифм x;

**sqrt(x)** - вычисляется √x .

Функция abs возвращает целое значение типа int, аргумент также должен быть целым. Остальные функции возвращают вещественное (double) значение при вещественном аргументе.

При использовании указанных функций в программу нужно включить директиву ***#include <math.h>*.**

## Оператор-выражение

В языке Си любое выражение, заканчивающееся точкой с запятой (;), является оператором.

Примеры:

```
i++;  /* увеличение значения i на 1, эквивалентно оператору i=i+1; */
i--;  /* уменьшение i на 1 */
a+2;  /* смысла не имеет, хотя синтаксически верно */
```

**Примечание.** Рассмотренный выше оператор присваивания является частным случаем оператора-выражения, поскольку в выражениях можно использовать операцию присваивания (=) наравне с другими.

## Оператор вызова функции

Оператор вызова функции имеет вид:

***имя_функции (аргумент1, ... , аргументN);***

Он тоже является частным случаем оператора-выражения.

Примерами операторов вызова функции являются уже знакомые вам операторы вызова функций форматированного ввода/вывода printf и scanf (см. стр. 9). Рассмотрим эти функции детальнее.

### Использование функции *printf*

Функция printf служит для вывода на экран монитора сообщений, данных, результатов вычислений. Число аргументов - один или более. Первый аргумент функции - это форматная строка, которая может содержать тексты, подлежащие выводу на экран, управляющие символы, форматы вывода значений переменных или выражений. Остальные аргументы - это переменные или выражения. Вернемся к примеру программы на стр.9.

В операторе

*printf ("Задайте два числа: ");*

аргумент только один - форматная строка, содержащая текст. В операторе

*printf ("%d + %d = %d\n", a, b, a+b);*

четыре аргумента. Первый аргумент - форматная строка (строка символов в кавычках) показывает, как должны быть напечатаны значения остальных аргументов (a,b,a+b). Каждому из аргументов a,b и a+b соответствует одна спецификация преобразования (формат) %d. Это спецификация вывода целого числа. Кроме форматов, форматная строка содержит последовательности символов " + ", " = ", которые нужно вывести, и управляющий символ '\n' (перевод строки), чтобы после вывода результата курсор переместился в начало следующей строки.

Функция printf выводит на экран то, что указано в форматной строке, подставляя вместо каждого формата значение очередного аргумента из списка. Число форматов должно быть равно числу аргументов после форматной строки. Не забудьте это основное требование!

Ниже приведены некоторые форматы:

%d - для вывода целого числа со знаком (типов int, short);

%ld - для вывода целого числа со знаком (типа long);

%u - для вывода целого числа без знака (типа unsigned);

%f - для вывода вещественного числа (типов float, double) в формате числа с фиксированной точкой (с точностью по умолчанию 6 цифр после точки);

%e - для вывода вещественного числа в экспоненциальном формате: [-]d.dddddde{ }dd (здесь d - десятичная цифра);

%c - для вывода символа;

%s - для вывода строки символов.

Например, если число 123.68 вывести в формате %f, то будет напечатано 123.680000, если же указать формат %e, то результатом будет 1.236800e+02. В форматах %f и %e можно указать точность, например:

%.1f 123.7

%.4e 1.2368e+02

**Использование функции *scanf***

Функция scanf предназначена для ввода значений переменных с клавиатуры во время выполнения программы.

Список аргументов этой функции почти такой же, как у функции printf. Первый аргумент - это форматная строка, содержащая форматы ввода значений переменных. Сами переменные, точнее указатели на переменные, записываются в списке аргументов после форматной строки.

Примеры:

```
int a,b;
float x,y;
double z;
scanf ("%d %d", &a, &b);
scanf ("%f %e %lf", &x, &y, &z);
```

При выполнении первого оператора scanf будут прочитаны вводимые пользователем с клавиатуры два целых числа и присвоены переменным *a* и *b*. При втором вызове функции scanf будут введены три вещественных числа

и присвоены соответственно переменным x, y и z.

Числа во входном потоке могут разделяться либо пробелами, либо символами "новой строки", либо символами табуляции. Например, входной поток может выглядеть так:

-52 1374

0.5 -17.472

345678.7654

Тогда результат выполнения операторов scanf будет такой:

a=-52; b=1374; x=0.5; y=-17.472; z=345678.7654.

Функция scanf использует практически тот же набор форматов, что и функция printf. Основные отличия в случае функции scanf следующие:

1. Формат %hd служит для ввода коротких целых чисел (типа short).

2. Форматы %f и %e эквивалентны и используются для ввода чисел типа float. Обе спецификации допускают наличие (или отсутствие) знака, строки цифр с десятичной точкой или без нее и поля показателя степени.

3. Форматы %lf и %le определяют тип вводимых данных как double.

**Составной оператор**

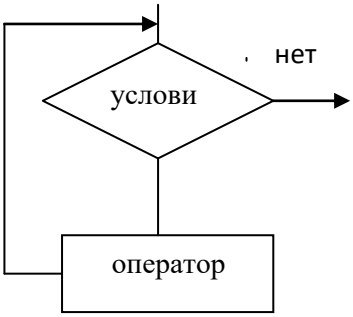
Составной оператор состоит из одного или большего числа операторов любого типа, заключенных в фигурные скобки.

Пример:

{ x=1; y=2; z=3; }

**Оператор цикла while**

Оператор цикла служит для многократного выполнения одних и тех же действий. Оператор while используется при программировании циклов с предусловием:



**Формат оператора while:**

***while ( условие)***

**оператор**

Оператор while выполняется так, как изображено на схеме. Проверяется условие; если оно истинно, то выполняется оператор, входящий в состав while (так называемое "тело цикла"). Затем снова проверяется условие... Тело цикла будет выполняться до тех пор, пока условие не станет ложным. Затем управление передается следующему оператору программы.

Обратите внимание, что тело цикла - это один оператор: либо простой, либо составной.

Условие - это выражение, которое кроме арифметических операций может содержать операции отношения:

> больше,

>= больше или равно,

< меньше,

<= меньше или равно,

== равно,

**!=** не равно.

Пример оператора:

```
while (i<=n)
  {  s=s+i/(i+1); i++; }
```

**Пример**

***Задача***. Дано действительное число x. Вычислить значение sinx с помощью ряда

$$y = \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} + \ldots$$

с точностью $10^{-5}$ (т.е. учитывая только те члены ряда, которые по

абсолютной величине больше либо равны $10^{-5}$ ). Для проверки результата вычислить sinx с помощью стандартной функции.

Обозначим очередной член данного ряда через $a_n$ .

$$a_0 = x$$

$$a_1 = -\frac{x^3}{3!} = -a_0 * \frac{x^2}{2*3}$$

$$a_2 = +\frac{x^5}{5!} = -a_1 * \frac{x^2}{4*5}$$

. . .

$$a_n = -a_{n-1} * \frac{x^2}{2n(2n+1)}$$

**Программа**

```
  /*  Приближенное вычисление  sin x  */
#include <stdio.h>
#include <math.h>
#define   E  1e-5       /* точность вычисления  */
main()
{
   float  x,          /* аргумент функции    */
          y;          /* сумма ряда          */
   float  a;          /* очередной член ряда */
   int    n;          /* номер итерации      */
   printf ("x=");
   scanf ("%f",&x);
   y=0; a=x; n=0;
   while ( fabs(a) >= E )
```

```
    { y=y+a;
      n++;          /* n=n+1; */
      a=-a*x*x/(2*n*(2*n+1));  /* выч-е очередного члена ряда
                                  через предыдущий */
    }
  printf ("y=%f\n",y);
  printf ("sinx=%f\n",sin(x));
}
```

**Примеры результатов выполнения программы:**

x=3.14159
y=0.000002
sinx=0.000003

x=0
y=0.000000
sinx=0.000000

x=1.5708
y=1.000004
sinx=1.000000

x=-1.5708
y=-1.000004
sinx=-1.000000

## Ввод и отладка программы в TURBO C

1. Запустите систему TURBO C.
2. Если окно *Edit* не пустое, выберите команды меню **File | New** для создания нового файла.
3. В окне *Edit* введите текст своей программы, соблюдая ступенчатую

запись.После ввода каждой строки нажимайте [Enter]. Если заметите ошибки, их легко исправить, вернув курсор в нужную позицию с помощью клавиш управления движением курсора.

4. Выберите команды меню **Compile | Compile** для компиляции программы.

При компиляции могут быть обнаружены синтаксические ошибки в программе. В таком случае в верхней части окна редактирования появится первое сообщение об ошибке, а курсор укажет на предполагаемое местоположение ошибки в программе. Исправьте ошибку и повторите компиляцию.

5. После исправления всех синтаксических ошибок сохраните программу в файле на диске, выполнив команды меню **File | Save** (или нажав клавишу [F2]). Если на экране появится запрос на изменение имени файла, сотрите *NONAME.C* (это имя дается вначале всем новым файлам) и введите любое другое имя (идентификатор длиной до 8 символов). Расширение *.c* добавляется к имени автоматически.

Замечание. При многократном сохранении программы сохраняется обычно две последние версии программы: последняя - в файле с расширением *.c* , а предпоследняя - в файле с расширением *.bak* .

6. Запустите программу на выполнение, выбрав команды меню **Run| Run** (или нажав одновременно клавиши [Ctrl] и [F9]).

По запросу программы введите тестовые исходные данные. Проверьте результат. Вернуться к экрану с результатами можно, выбрав команды меню Run | User Screen (или нажав одновременно клавиши [Alt] и [F5]).

Для возврата к экрану системы нажмите любую клавишу.

Если результаты оказались неверными, проверьте программу. После редактирования программы повторите действия с пункта 4.

Если программа зациклилась, попытайтесь прервать ее выполнение с помощью одновременного нажатия клавиш [Ctrl] и [Break]. Если это сделать не удалось, перезагрузите систему. Если в окне редактирования появится не

Ваша программа, выполните команды меню **File | Load** (можно с помощью клавиши [F3]). На запрос имени файла введите то имя, которое дали файлу при его сохранении. Проверьте и отредактируйте программу и повторите действия с пункта 4.

## Порядок выполнения работы

1. Познакомьтесь с описанием языка Си и примером программы.

2. Получите у преподавателя индивидуальное задание.

3. Составьте блок-схему и программу на языке Си и подберите тесты для проверки программы на компьютере.

4. Отладьте программу на компьютере и покажите результаты тестирования преподавателю.

## Задания

1. Дано действительное число x. Вычислить значение y с помощью стандартной функции и с помощью ряда с точностью 0,0001:

а) $y = \cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!} + \ldots$

б) $y = e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots$

в) $y = \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \ldots + (-1)^{n-1} \frac{x^n}{n} + \ldots$

где $|x|<1$.

2. Дано натуральное число n. Проверить справедливость равенства:

а) $1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$

б) $1^2 + 2^2 + 3^2 + \ldots + n^2 = \frac{n(n+1)(2n+1)}{}$

в) $1 + 2 + 2^2 + ... + 2^{n-1} = 2^n - 1$

3. Дано натуральное число n.

а) Определить количество цифр в числе n.

б) Определить сумму его цифр.

в) Определить первую цифру числа n.

## Лабораторная работа 3

## Обработка числовых последовательностей

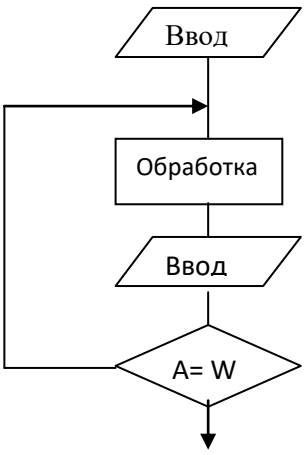
Существует круг задач, в которых необходимо как-то обработать заданную числовую последовательность, причем для получения результата достаточно просмотреть последовательность один раз. Например, чтобы вычислить среднее арифметическое заданной последовательности чисел, можно суммирование чисел и подсчет их количества совместить с вводом. Тогда не нужно будет хранить всю последовательность в памяти компьютера (в виде массива), достаточно иметь одну скалярную переменную целого или вещественного типа и поочередно присваивать ей вводимые значения.

Числовая последовательность может задаваться с указанием количества чисел или иметь какой-то признак конца.

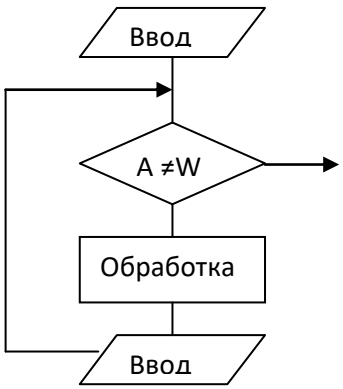
Пусть последовательность задается в виде $\mathbf{A_1 , A_2 , ... , A_n , W}$

(n заранее неизвестно, W - признак конца последовательности), тогда процесс обработки в общем виде можно представить одной из двух схем:

**а) цикл с постусловием**




**б) цикл с предусловием**




Фрагменты программ на Си, соответствующие этим схемам:

а)
```
scanf ("%f",&a);
do

{  /* обработка a */
   . . .
  scanf ("%f",&a);
}
```

б)
```
scanf ("%f",&a);
while (a!=W)

{  /* обработка a */
   . . .
  scanf ("%f",&a);
}
```

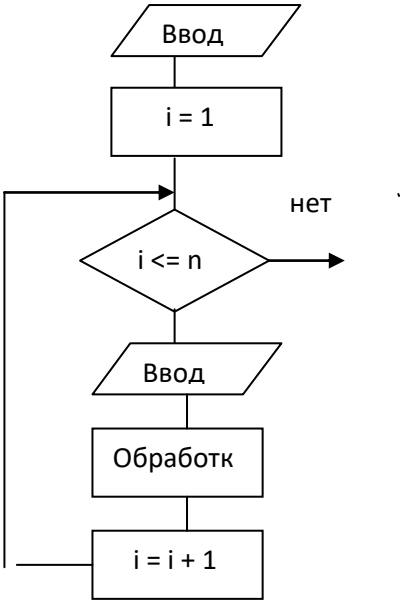
while (a!=W);

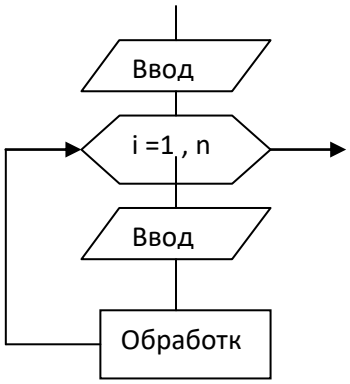
В этих фрагментах предполагается, что переменная ***a*** вещественного типа (***float***), поэтому указан формат ***%f***. Если же последовательность состоит из целых чисел типа int, то следует выбрать формат ***%d***. *W* - символическая константа, которая должна быть определена с помощью директивы ***#define***.

Числовая последовательность может быть задана и с указанием количества вводимых чисел:

**n, $A_1$ , $A_2$ , ..., $A_n$ .**

Тогда процесс обработки можно представить в виде:




а) б) более короткая форма

На языке Си этот процесс можно записать с помощью оператора цикла while или лучше оператора цикла for:

```
а) scanf  ("%d",&n); i=1;
   while (i<=n)
    {
     scanf ("%f",&a);
     /* обработка a */
     i++;
    }
```

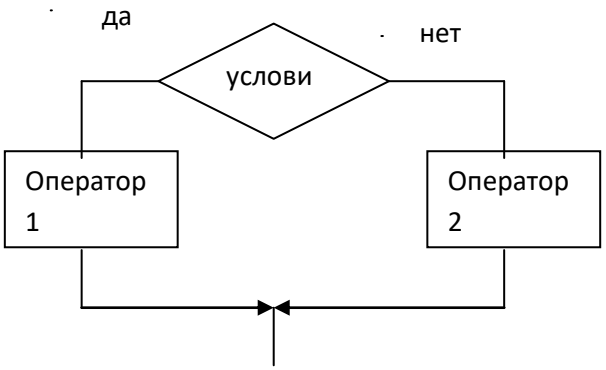
```
б)  scanf ("%d",&n);
    for (i=1; i<=n; i++)
     {
       scanf ("%f",&a);
       /* обработка a */
     }
```

Процесс обработки очередного числа может иметь линейную структуру (состоять из нескольких последовательно выполняемых операторов) или

структуру ветвления, которая программируется с помощью оператора ***if***:

**if (условие) оператор1**

**else оператор2**




В операторе if *оператор1* и *оператор2* могут быть составными операторами, конструкция "*else оператор2*" может отсутствовать, например:

```
if  (a % 2 != 0)
  { s=s+a;
    k++;
  }
```

В операторах ***if, while, do-while*** можно использовать не только условные выражения, но и вообще любые. Это связано с тем, что в Си значение "ложь" - это 0, а "истина" - любое ненулевое число. Поэтому предыдущий оператор if можно записать иначе:

```
if  (a % 2)  { s=s+a;  k++; }
```

## Пример решения задачи

***Задача.*** Даны целые числа n, $A_1$ ,$A_2$ , ... ,$A_n$ . Вычислить сумму тех чисел последовательности, которые удовлетворяют условию $|A_i| < i^2$.

## Программа:

```
#include <stdio.h>
#include <math.h>
main()
{
   int  n;          /* количество чисел  */
   int  a,          /* очередное число   */
        s=0,        /* сумма             */
        i;          /* порядковый номер числа в посл-ти */
   printf ("\nВведите количество чисел: ");
   scanf ("%d",&n);
   printf ("Введите числовую последовательноть:\n");
   for (i=1; i<=n; i++)
     {
      scanf ("%d",&a);
      if (abs(a) < i*i)  s=s+a;
     }
    printf ("сумма=%d\n",s);
}
```

## Тесты для проверки программы:

| № п/п | n | Исходная последовательность | Ожидаемый результат |
|---|---|---|---|
| 1 | 6 | 1 -2 3 16 -5 40 | сумма=-4 |
| 2 | 4 | -1 5 10 -20 | сумма=0 |

## Результаты тестирования программы:

Введите количество чисел: 6

Введите числовую последовательноть:

1 -2 3 16 -5 40

сумма=-4

Введите количество чисел: 4

Введите числовую последовательноть:

-1 5 10 -20

сумма=0

## Порядок выполнения работы

1. Получить задание у преподавателя.

2. Составить схему и программу на языке Си, подобрать тесты для проверки программы на компьютере.

3. Отладить программу на компьютере и показать результаты тестирования преподавателю.

## Задания

1. Даны натуральные числа n, $A_1$, $A_2$, ..., $A_n$. Определить количество членов $A_k$ последовательности $A_1$, ..., $A_n$:

а) кратных 3 и не кратных 5;

б) имеющих четные порядковые номера и являющихся нечетными числами;

в) удовлетворяющих условию $2^k < A_k < k!$.

2. Даны натуральное число n, действительные числа $A_1,A_2,...,A_n$.Получить:

а) удвоенную сумму всех положительных членов последовательности $A_1$, $A_2$, ..., $A_n$

б) сумму отрицательных и количество положительных членов последотельности $A_1$, $A_2$, ..., $A_n$

в) $\min(|A_1|, ..., |A_n|)$;

г) $\min(A_1, A_3, ...) + \max(A_2, A_4, ...)$.

3. Даны натуральные числа $A_1$, $A_2$, ... . Признак конца последовательности 0. Получить

а) количество и сумму тех членов последовательности, которые делятся на 5 и не делятся на 7;

б) наибольший член последовательности ;

в) $\min(A_1 + A_2 , A_2 + A_3 , ...)$.

4. Даны целые числа $A_1 , A_2 , ...$ .Признак конца последовательности число 9999. Выяснить, составляют ли числа возрастающую последовательность.

## Лабораторная работа 4

## Последовательная обработка символов

Если для решения задачи достаточно просмотреть исходный текст один раз, то обычно текст вводится и обрабатывается посимвольно и не хранится целиком в памяти (в виде массива). В программе используется переменная типа ***char***, которой поочередно присваиваются значения символов исходного текста. Ввод и обработка символов происходит до тех пор, пока не встретится признак конца текста или количество введенных символов не достигнет заданной длины текста.

Процессы обработки последовательностей символов можно представить такими же схемами, как и процессы обработки числовых последовательностей (см. описание предыдущей лабораторной работы). Только переменной будет присваиваться при вводе не число, а очередной символ.

### Функции *getchar* и *putchar*

Функции ***getchar*** и ***putchar*** служат соответственно для ввода и вывода одного символа. Для посимвольного ввода/вывода текстов лучше использовать эти функции, нежели scanf и printf.

Функция ***getchar()*** не имеет аргументов. Она получает очередной поступающий с клавиатуры символ и сама возвращает его значение выполняемой программе.

Пример вызова функции getchar:

```
char ch;
ch=getchar();
```

Функция ***putchar*** имеет один аргумент - это символ, который требуется вывести на экран.

Примеры вызова функции putchar:

```
putchar (ch);     /* ch - переменная типа char */
putchar ('S');
putchar ('\n');    /* перевод строки */
```

Определения функций getchar и putchar содержатся в файле stdio.h.

**Пример решения задачи**

***Задача.*** Дан текст произвольной длины, оканчивающийся точкой. Проверить, есть ли в тексте сочетания "BA".

**Программа:**

```
#include <stdio.h>
main()
{
  char  s;              /* текущий символ текста */
  char  prs;            /* предыдущий символ     */
  short net=1;          /* признак, имеется ли "BA" в тексте */
                        /* net=1, если "BA" нет            */
                        /* net=0, если "BA" есть           */
  printf  ("\nВведите текст.\n");
  s=getchar();          /* чтение первого символа */
  if (s!='.')
   {  do
       { prs=s; s=getchar();
         if  (prs=='B' && s=='A')   net=0;
       }
      while (s!='.');
   }
  if (net)  printf ("В тексте нет 'BA'.\n");
  else printf ("В тексте есть 'BA'.\n");
}
```

**Тесты для проверки программы**

| N теста | Исходный текст | Ожидаемый результат |
|---|---|---|
| 1 | МОСКВА, БЕРЛИН, ВАРНА . | В тексте есть 'BA'. |
| 2 | ПЭВМ IBM PC . | В тексте нет 'BA'. |
| 3 | . | В тексте нет 'BA'. |

## Порядок выполнения работы

1. Получить у преподавателя задание.

2. Составить схему и программу на Си и подобрать тесты для проверки программы на компьютере.

3. Отладить программу и показать результаты тестирования преподавателю.

## Задания

1. Дан текст произвольной длины, оканчивающийся точкой с запятой. Подсчитать количество цифр в тексте.

2. Дана последовательность символов и количество символов в этой последовательности. Проверить, есть ли в тексте латинские буквы.

3. Дан текст произвольной длины, оканчивающийся символом ";". Проверить, есть ли в тексте скобки.

4. Дан текст заданной длины. Подсчитать количество сочетаний ":=".

5. Дан текст произвольной длины, оканчивающийся точкой. Текст состоит из слов, разделенных пробелами. Подсчитать

а) количество слов в тексте;

б) количество слов, начинающихся с буквы К;

в) количество слов, заканчивающихся буквой А.

6. Дано скобочное выражение, оканчивающееся точкой с запятой.

а) Проверить правильность расстановки скобок в выражении.

б) Подсчитать количество уровней вложенности скобок в выражении.

7. Дана строка символов. Признак конца - символ '\n' ( перевод строки).

а) Удалить лишние пробелы, т.е. если подряд следует несколько пробелов, оставить только один.

б) Удалить последовательности символов, заключенные в фигурные скобки.

## Лабораторная работа 5

## Обработка массивов

Массив используется, когда дана упорядоченная совокупность однотипных данных (чисел, символов, строк символов и т.д.) ограниченным числом элементов.

Примеры описаний массивов:

```
char text[10];        /* массив из 10 символов          */
int a[50];            /* массив из 50 целых чисел       */
float matr[5][10];    /* матрица вещ.чисел разм.5x10 */
```

Для обращения к элементу массива указываются имя массива и индексы элемента в квадратных скобках, например, text[0], a[i+1], matr[i][j]. Индексация начинается с 0 (в приведенном примере text[0] - первый элемент массива, последний элемент имеет индекс 9).

Ввод/вывод числовых массивов осуществляется в цикле поэлементно.

### Пример программы:

### Первый вариант:

```
/*  Задача. Входная строка содержит последовательность   */
/*  слов, разделенных пробелами. Признак конца строки -  */
/*  символ '\n' (перевод строки). Вывести на экран слова     */
/*  длиной до пяти символов.                                 */
   #include <stdio.h>
   #define  DLSL  80    /* макс. длина слова */
   main()
   {   char s;              /* тек. символ */
       char sl[DLSL];  /* тек. слово  */
       int i,j;             /* индексы тек. символа в слове */
      int psl=1;          /* признак, что слово длиной до 5 симв. первое */
      printf ("\n\nВведите строку символов\n");
      s=getchar();
      while (s!='\n')
```

```
        {
         if (s==' ')  s=getchar();
         else
          { i=0;
            do
             { sl[i++]=s;
               s=getchar();
             }
            while ((s!=' ') && (s!='\n'));
             if (i<5)
               { if (psl)  /* если слово первое */
                  { printf ("Слова длиной до 5 символов:\n");
                    psl=0;
                  }
                 for (j=0; j<i; j++)
                    putchar(sl[j]);
                 putchar(' ');
                }
          }
        }
      if (psl)  printf ("Слов длиной до 5 символов нет");
  }
```

**Пример результатов тестирования программы:**

Введите строку символов

май  апрель март   весна лето

Слова длиной до 5 символов:

май март лето

Введите строку символов

декабрь  январь  февраль

Слов длиной до 5 символов нет

**Второй вариант программы:**

```
   /*  Задача. Входная строка содержит последовательность      */
   /*  слов, разделенных пробелами. Признак конца строки -     */
   /*  символ '\n' (перевод строки). Вывести на экран слова    */
   /*  длиной до пяти символов.                                */
#include <stdio.h>
#define  DLSTR  80     /* макс.длина строки */
main()
 {  char str[DLSTR];   /* тек. строка */
    int i,j;           /* индексы тек. символа в строке */
    int n,k;           /* индексы перв. и посл. символов тек. слова
                          в строке */
    int net_sl=1;      /* признак, что слов длиной до 5 симв. нет */
    printf ("\n\nВведите строку символов\n");
    gets(str);     /* ввод строки в массив str с заменой символа '\n'
                      на признак конца строки '\0'  */
    printf ("Результат:\n");
    i=0;
    while (str[i]!='\0')
     {
       if (str[i]==' ')  i++;
       else
         { n=i;
           do  i++;  while ((str[i]!=' ') && (str[i]!='\0'));
           k=i;
           if ( k-n < 5 )
             { for (j=n; j<k; j++)
                  putchar(str[j]);
               putchar(' ');
               net_sl=0;
```

```
            }
        }
    }
    if (net_sl)  printf ("Слов длиной до 5 символов нет.");
    printf ("\nДля завершения нажмите любую клавишу");
    getch();   /* чтение символа без отображения его на экране */
}
```

**Пример результатов тестирования программы:**

Введите строку символов

весна лето осень зима

Результат:

лето зима

Для завершения нажмите любую клавишу

Введите строку символов

декабрь январь февраль

Результат:

Слов длиной до 5 символов нет.

Для завершения нажмите любую клавишу

**Порядок выполнения работы**

1. Получить задание у преподавателя.

2. Составить схему и программу на Си и подобрать тесты для проверки программы на компьютере.

3. Отладить программу на компьютере.

**Задания**

1. Дана строка символов. Признак конца строки - символ '\n' (перевод строки). Строка состоит из слов, разделенных пробелами.

а) Вывести слова, у которых первая и последняя буквы одинаковые, и количество таких слов.

б) Вывести слова, заканчивающиеся буквой 'а', с порядковыми номерами этих слов в исходной строке.

в) Вывести слова, заканчивающиеся слогом 'ва', и длину каждого из этих слов.

г) Вывести самое короткое слово и его длину.

д) Вывести самое длинное слово и его порядковый номер в исходной строке.

2. Дан массив, состоящий из 10 целых чисел. Упорядочить массив по убыванию элементов методом последовательного нахождения максимума.

3. Дан массив из 10 вещественных чисел. Упорядочить массив по возрастанию элементов методом последовательного нахождения минимума.

4. Дан массив из n целых чисел (n<=10). Упорядочить массив по убыванию элементов методом "пузырька".

5. Дана строка длиной не более 80 символов, оканчивающаяся точкой.

а) Подчеркнуть все гласные буквы в строке.

б) Определить, сколько раз каждая цифра встречается в данной последовательности.

в) Определить, сколько раз каждая буква латинского алфавита встречается в данной последовательности.

г) Определить символ, встречающийся в тексте с максимальной частотой.

## Лабораторная работа 6

## Функции

Вы уже знакомы с некоторыми библиотечными функциями, такими как printf(), scanf(), getchar(), putchar(), gets(), sin(), cos(), ... . Теперь нужно знать, как создавать свои собственные функции.

Функция - это самостоятельная единица программы, предназначенная для решения определенной задачи. Функции в языке Си играют ту же роль, какую играют функции, подпрограммы и процедуры в других языках программирования.

Программа на Си может состоять из любого количества функций, одна из которых всегда носит имя main. Выполнение программы начинается с функции main(), которая может вызывать другие функции. Те в свою очередь тоже могут вызывать какие-то функции.

Описания функций могут размещаться в одном файле или в нескольких. Рассмотрим случай, когда все функции программы описываются в одном файле. В этом случае расположить в файле описания функций можно в любом порядке. Но проще описание каждой функции поместить перед ее использованием (вызовом) в других функциях.

Описание любой функции имеет вид:

```
Заголовок функции
{  Описания локальных переменных
   Операторы
}
```

Заголовок функции имеет формат:

**тип_функции имя_функции ( список_параметров )**

Через параметры (аргументы) происходит передача данных между функцией и вызывающей программой. В списке параметров перечисляются через запятую описания параметров (перед каждым параметром указывается тип). Имена параметров могут быть любые, т.к. это формальные параметры. Если функция не имеет параметров, то пишутся пустые скобки.

Примеры заголовков функций:

```
main()
float max (float x, float y)
int prov (char mas[], int s)
void line (ink k, int n, char simv, int ps)
```

Тип функции, указанный перед именем функции, - это тип возвращаемого функцией значения. По умолчанию предполагается тип ***int***. Если функция никакое значение не возвращает, то указывается тип ***void.***

Допускается запись заголовка функции в иной форме, когда в списке параметров перечисляются только имена, а описания параметров выносятся за скобки. Например:

```
void line (k,n,simv,ps)
        int k,n;
        char simv;
        int ps;
```

Но такая форма записи уже устарела (и имеет недостатки).

Значение функции передается в вызывающую программу с помощью оператора возврата ***return.***

Пример описания функции:

```
/*  функция определения наибольшего из двух чисел  */
float  max ( float x, float y )
{  if (x>y) return x;
   else return y;
}
/*  2-й вариант  */
float  max ( float x, float y )
{  float z;    /*  z=max(x,y)  */
   z = (x>y) ? x : y ;
   return z;
}
```

Вы можете вызвать свою функцию в любом выражении, указав ее имя и аргументы (фактические параметры):

**имя_функции (аргумент_1, аргумент_2, ...)**

Аргументом может быть идентификатор, константа и выражение. Типы аргумента и соответствующего параметра в заголовке функции должны совпадать. Например, оператор

f = max(a,b) - max(c,d);

содержит два вызова приведенной выше функции max. При первом обращении функции max передаются значения переменных a и b, она возвращает наибольшее из этих чисел, которое подставляется вместо указателя функции max(a,b). При втором вызове функции max формальным параметрам x и y присваиваются соответственно значения фактических параметров c и d. Оператор return возвращает наибольшее из этих значений в точку вызова функции.

Вызов функции без параметров имеет вид: **имя_функции ().**

**Использование указателей при передаче параметров**

Обмен информацией между вызывающей программой и функцией осуществляется благодаря параметрам. Если данные входные и функцией не изменяются, то передавать в списке параметров нужно значения данных. Сложнее вернуть вызывающей программе выходные данные, если их несколько (если результатом выполнения функции является одно единственное значение, то оно обычно, как вы уже видели, в число параметров не включается, а является значением функции). Если функция должна вернуть вызывающей программе нес колько значений, в этом случае передавать ей в качестве фактических параметров нужно **адреса** переменных. Вспомните вызов функции scanf(). Если, например, нужно ввести значения двух переменных x и y типа int, вы должны написать:

scanf ("%d %d",&x,&y);

где &x,&y - адреса переменных x и y. Точно так же при обращении к своей функции для выходных параметров нужно указывать адреса.

В описании функции соответствующие формальные параметры должны быть описаны как **указатели** на переменные соответствующего типа.

Допустим, функция function1() должна изменить значения переменных a и b типа float. В вызывающей программе обращение к функции нужно записать так:

function1 ( &a, &b );

а заголовок функции должен иметь вид:

void function1 ( float *ptr1, float *ptr2 )

Здесь prt1 и ptr2 - это указатели на переменные типа float (названия их могут быть изменены - это формальные параметры). Значениями указателей являются адреса. При вызове функции function1 переменной pt1 присваивается адрес a, а ptr2=&b.

Таким образом функции становятся доступными переменные вызывающей программы. Если нужно этим переменным присвоить новые значения, допустим, с помощью операторов присваивания, то это делается так:

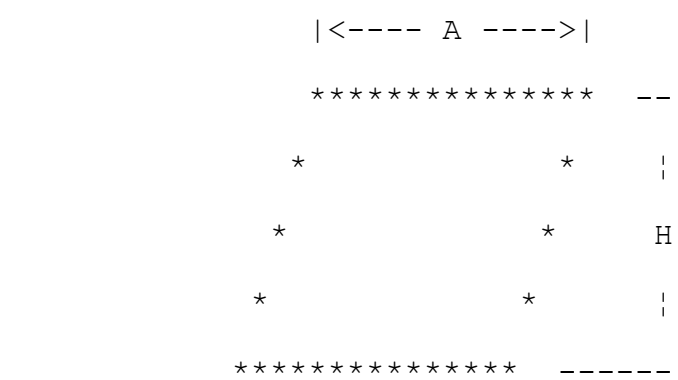
*ptr1 = ... ;

*prt2 = ... ;

Итак, если ptr1 и ptr2 являются указателями на переменные, то чтобы взять значения этих переменных, нужно перед именами указателей использовать операцию *****.

## Пример

**Задача.** Нарисовать параллелограмм вида




при заданных значениях A, H и C, где C - смещение фигуры по горизонтали вправо относительно левой границы экрана.

В приведенной ниже программе параллелограмм вычерчивается в текстовом режиме, хотя можно (и проще) было бы сделать это в графическом режиме, используя стандартные графические процедуры Turbo C.

## Программа

```
#include <stdio.h>
#include <conio.h>
/*    Функция вывода горизонтального отрезка прямой    */
void   line ( int k, int n, char simv, int ps)
        /*  Входные данные:                                          */
        /*       k    - смещение отрезка вправо,                     */
        /*       n    - длина отрезка,                               */
        /*       simv - символ заполнения отрезка,                   */
        /*       ps - признак перевода строки:                       */
        /*            если ps!=0, то нужен перевод строки после вывода*/
        /*            отрезка; если ps=0, то не нужен перевод строки  */
{     int i;     /* параметр цикла */
          /* установление левой границы отрезка */
      for ( i=0; i<k; i++ )   putchar (' ');
          /* вывод отрезка */
```

```
    for ( i=0; i<n; i++ )   putchar (simv);
    if (ps)  putchar ('\n');
}
      /*   О с н о в н а я   п р о г р а м м а          */
main()
{   int  c,    /* смещение нижнего левого угла */
         a,h,  /* основание и высота параллелограмма */
         i,    /* параметр цикла */
         cls;  /* смещение левой боковой стороны */
    printf ("Введите три числа: смещение, основание, высоту.\n");
    scanf ("%d %d %d",&c,&a,&h);
    cls=c+h-1;
    line (cls,a,'*',1);
    for (i=0; i<h-2; i++)
      { line (--cls,1,'*',0);
        line (a-2,1,'*',1);
      }
    line (c,a,'*',1);
    getch();
}
```

**Порядок выполнения работы**

1. Получить задание у преподавателя.

2. Составить схему и программу на Си и подобрать тесты для проверки программы на компьютере.

3. Выполнить тестирование и отладку программы на компьютере, показать результаты преподавателю.

**Задания**

Нарисовать заданную фигуру при заданных значениях A (основание), H (высота) и C (смещение фигуры по горизонтали вправо), используя процедуру вывода горизонтального отрезка прямой, приведенную в

примере.

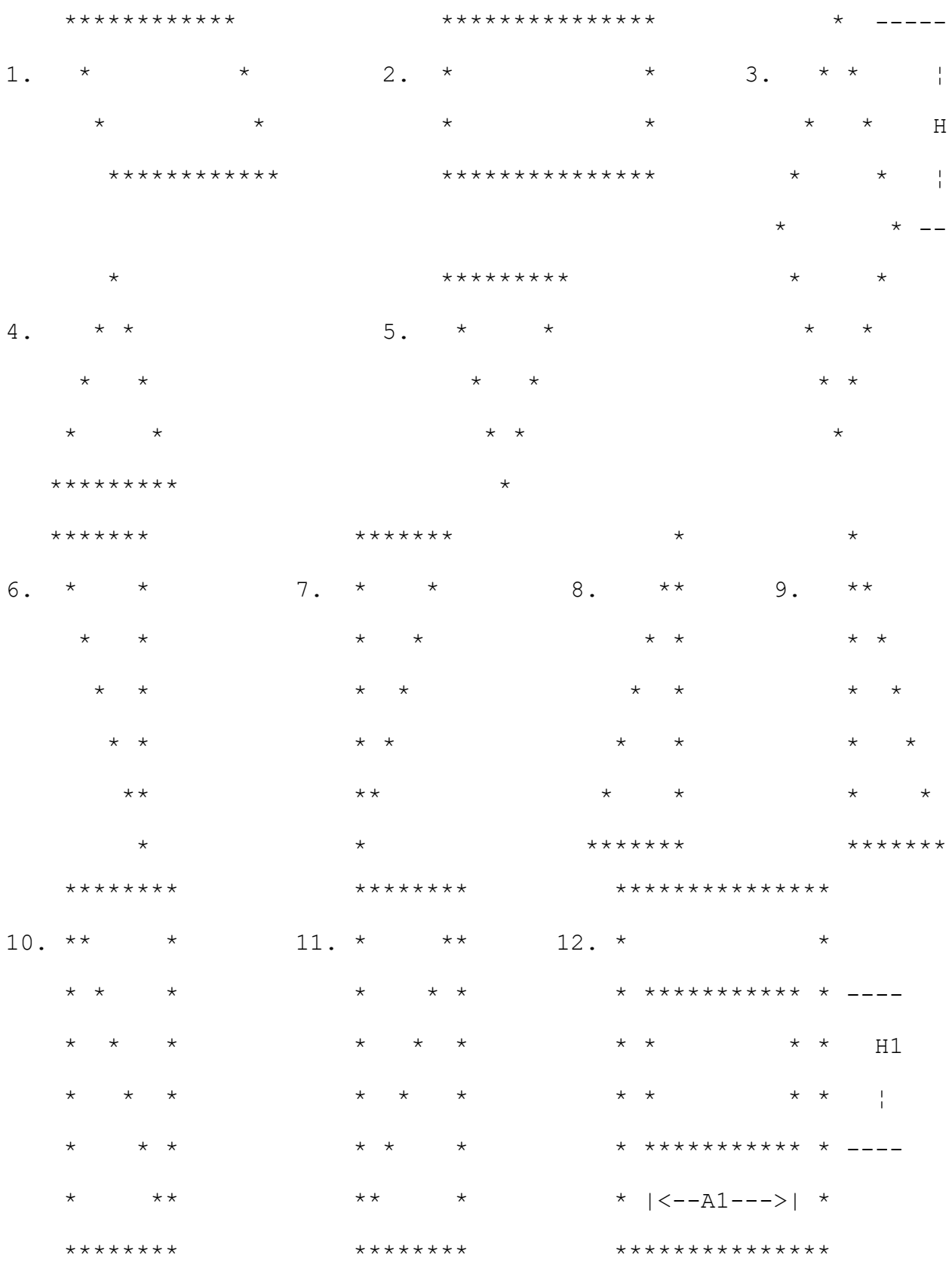
Составить несколько вариантов основной программы:

а) начертить только контур фигуры;

б) заполнить внутреннюю часть фигуры тем же символом, которым изображается контур фигуры;

в) заполнить внутреннюю часть фигуры точками.

## Варианты заданий:




**Примечание.** Все значения A,H,C задают число выводимых символов или число позиций. В заданиях 4-5 A=2H-1, в заданиях 6-11 A=H.

**Лабораторная работа 7**
**Символьные строки и функции обработки строк**

Строка символов - это последовательность символов произвольной длины, завершающаяся нуль-символом '\0'(все биты в байте нулевые).

Строковые константы записываются в кавычках, например:

"Как Вас зовут?"

Если в программе встречается строковая константа, компилятор выделяет для нее память объемом, равным длине строки (количеству символов) + 1 (для нуль-символа). Признак конца строки '\0' добавляется автоматически.

Строковые переменные описываются либо как массивы символов, либо как указатели на символы, например:

```
char stroka[81];      /* строка длиной до 80 символов */
char *str;            /* указатель на строку */
```

В первом случае память для строки выделяется (имя массива является адресом его первого символа), во втором случае - нет (выделяется только для указателя).

При выполнении оператора

```
str= "Группа 21101";
```

указателю str присваивается адрес памяти, где размещена строковая константа "Группа 21101". Для массива такой оператор будет неверным, т.к. адрес массива - величина постоянная.

Еще раз обратите внимание, что в Си имя массива отождествляется с адресом его первого элемента. Для описанного выше массива stroka имя массива stroka и &stroka[0] идентичны. Другими словами, имя массива является указателем на первый символ, причем его нельзя изменять.

Если строка будет вводиться с клавиатуры, то лучше описать переменную как массив символов, иначе предварительно придется выделять память для строки, например, с помощью функции ***malloc().***

## Функции gets() и puts()

Для ввода и вывода строк символов служат функции gets() и puts().

Функция gets() вводит с клавиатуры строку, заменяя символ "перевод строки" на нуль-символ, и помещает ее по указанному адресу. Например:

```
gets (stroka);  /* ввод строки в массив строка */
```

Функция puts() выводит указанную строку на экран. Например, оператор

```
puts(str);
```

выведет на экран строку, на которую указывает переменная str, курсор после вывода переместится на новую строку.

### Функции обработки строк

Для работы со строками символов в библиотеке Turbo C имеется ряд функций, например, функция определения длины строки strlen(), копирования строк strcpy(), сцепления строк strcat(), сравнения строк strcmp(), нахождения в строке указанного символа strchr(). Прототипы таких функций находятся в файле ***string.h***, их перечень приведен в следующем разделе.

Примеры обращения к функциям обработки строк:

```
char s1[81], s2[81], s3[81];
char *s;
gets(s1);
printf ("Длина строки = %d\n", strlen(s1));
strcpy(s2,s1);    /* копирование строки s1 в массив s2  */
if ((s=strchr(s2,'a'))!=NULL)    /* есть буква 'a' в строке s2 */
     *s = 'b';     /* замена в строке s2 первой буквы 'a' на 'b' */
gets(s3);
if (strcmp(s1,s3)==0) printf ("Строки одинаковые\n");
```

Рассмотрим одну из библиотечных функций - функцию сцепления двух заданных строк strcat(). Определение функции:

```
char *strcat (char *s1, char *s2);
```

Функция копирует строку s2 (на которую ссылается указатель s2) в конец строки s1 и возвращает значение s1 - ссылку на сцепленную строку.

Работу функции можно описать так:

```
char *strcat (char *s1, char *s2)
   {     char *rs;        /* ссылка на результирующую строку   */
  rs=s1;  /* запоминание адреса начала строки s1 */
  while (*s1!='\0')   s1++;  /* поиск конца строки s1 */
       /* копирование строки s2 в конец s1 */
  while (*s2!='\0')
    { *s1=*s2; s1++; s2++; }
  *s1='\0';
   return rs;
   }
```

Как видите, функция не проверяет, достаточно ли памяти для результирующей строки. Вызывающая программа должна позаботиться об этом.

А теперь посмотрите на более компактную (но менее понятную) запись этой функции:

```
char *strcat (char *s1, char *s2)
   {     char *rs;
  rs=s1;  /* запоминание адреса начала строки s1 */
  while (*s1!='\0') s1++;          /* поиск конца строки s1 */
  while ((*s1++ = *s2++) !='\0');  /* копирование s2 в конец s1,
                                      включая нуль-символ */
  return rs;
   }
```

Даже еще можно сократить текст функции, записав второй оператор **while** короче:

```
      while (*s1++ = *s2++);
```

Каждый раз очередной символ из второй строки копируется в первую,

затем значения указателей s1 и s2 увеличиваются на 1, т.е. происходит продвижение указателей к следующим символам строк. Этот процесс повторяется до тех пор , пока не скопируется нуль-символ (т.к. выход из цикла происходит при нулевом значении выражения в скобках).

Теперь вам предлагается самим **написать одну из функций обработки строк** из списка библиотечных или дополнительных функций, указанную преподавателем. Для проверки работы вашей функции напишите драйвер (программу отладки) в виде функции main().

Пример драйвера для функции сцепления строк strcat():

```
#include <stdio.h>
/*     Тестирование ф-ции strcat    */
/*        Программа-драйвер          */
main()
  { char str1[81],str2[81];
    puts ("Введите две строки");
    gets (str1);
    gets (str2);
    if (strlen(str1)+strlen(str2) < 81)
      { puts ("Результат:");
        puts (strcat(str1,str2));
        printf ("Строки после вызова функции сцепления:\n%s\n%s\n",
                  str1,str2);
      }
    else puts ("Не хватает памяти для результирующей строки");
    getch();
  }
```

Библиотечные функции обработки строк

1. strcmp - сравнить две строки.

Определение: int strcmp (char *s1, char *s2);

0, если строки одинаковые;

Значение функции = разность кодов двух первых несовпадающих символов, если строки разные.

2. strncmp - сравнить первые n символов двух строк.

Определение: int strncmp (char *s1, char *s2, int n);

0, если первые n символов строк совпадают;

Значение функции

разность кодов символов, если первые n символов строк не совпадают.

3. strcpy - копировать строку s2 в s1.

Определение: char *strcpy (char *s1, char *s2);

Значением функции является s1 - ссылка на первую строку.

4. strncpy - копировать не более n символов строки s2 в s1.

Определение: char *strncpy (char *s1, char *s2, int n);

Значением функции является s1 - ссылка на первую строку.

5. strlen - определить длину строки (число символов без завершающего нуль-символа).

Определение: int strlen(char *s);

6. strchr - найти в строке s первое вхождение символа c.

Определение: char *strchr (char *s, char c);

Значение функции - ссылка на первый символ c в строке s или NULL (пустая ссылка), если символа нет в строке.

7. strrchr - найти в строке s последнее вхождение символа c.

Определение: char *strrchr (char *s, char c);

Значение функции - ссылка на последний символ c в строке s или NULL (пустая ссылка), если символа нет в строке.

8. strpbrk - найти в строке s1 любой из множества символов, входящих в строку s2.

Определение: char *strpbrk (char *s1, char *s2);

Значение функции - ссылка на любой символ в строке s1, имеющийся в s2, или NULL (пустая ссылка), если символов из s2 нет в s1.

9. strstr - найти в строке s1 первое вхождение строки s2.

Определение: char *strstr (char *s1, char *s2);

Значение функции - ссылка на первое вхождение s2 в s1 или NULL (пустая ссылка), если подстроки s2 нет в s1.

10. strncat - сцепить две строки s1 и s2, причем из второй строки копировать не более n символов.

Определение: char *strncat (char *s1, char *s2, int n);

Значением функции является s1 - ссылка на результирующую строку (n символов строки s2 копируется в конец строки s1).

**Дополнительные функции обработки строк**

11. strrsub - найти в строке s1 последнее вхождение строки s2.

Определение: char *strrsub (char *s1,char *s2);

Значение функции - ссылка на последнее вхождение s2 в s1 или NULL (пустая ссылка), если подстроки s2 нет в s1.

12. delchr - удалить в строке s первое вхождение символа c.

Определение: char *delchr (char *s,char c);

Значением функции является s - ссылка на строку.

13. delrchr - удалить в строке s последнее вхождение символа c.

Определение: char *delrchr (char *s,char c);

Значением функции является s - ссылка на строку.

14. delnchr - удалить в строке s n первых символов.

Определение: char *delnchr (char *s,int n);

Значением функции является s - ссылка на строку.

15. delchrn - удалить в строке s все символы, кроме n первых .

Определение: char *delchrn (char *s,int n);

Значением функции является s - ссылка на строку.

16. delstr - удалить в строке s1 первое вхождение строки s2.

Определение: char *delstr (char *s1,char *s2);

Значением функции является s1 - ссылка на первую строку.

17. chngchar - заменить в строке s каждый символ c1 на символ c2.

Определение: char *chngchar (char *s,char c1,char c2);

Значением функции является s - ссылка на строку.

18. chngstr - заменить в строке s первое вхождение строки s1 на строку s2.

Определение: char *chngstr (char *s,char *s1,char *s2);

Значением функции является s - ссылка на строку.

Порядок выполнения работы.

1. Получите задание у преподавателя.

2. Составьте функцию и драйвер, подберите тесты для проверки функции на ЭВМ.

3. Если аналогичная функция есть в библиотеке Turbo C, выполните тестирование библиотечной функции с помощью вашего драйвера, добавив директиву #include <string.h> (разумеется, свою функцию пока включать в программу не нужно). Затем выполните тестирование и отладку своей функции (результаты должны быть такими же, как и при выполнении библиотечной функции).

## Лабораторная работа 8

## Структуры

Если массив – это совокупность однотипных элементов, то структура объединяет логически связанные данные разных типов.

При использовании структурных переменных вначале нужно описать тип (или шаблон) структуры, а затем структурные переменные. Тип структуры задает порядок следования отдельных элементов (полей) в структуре и их типы. Описание структурного типа имеет вид:

```
struct  имя_типа
{  Описание элементов
};
```

Точка с запятой в конце описания обязательна. Например:

```
struct  BOOK
{  char author[40];       /* автор книги */
   char  name {50};       /* название */
   int year;              /* год издания */
   int pages;             /* число страниц */
};
```

Здесь *BOOK* - это имя типа структуры, а не переменная. При описании типа память под структуру не выделяется.

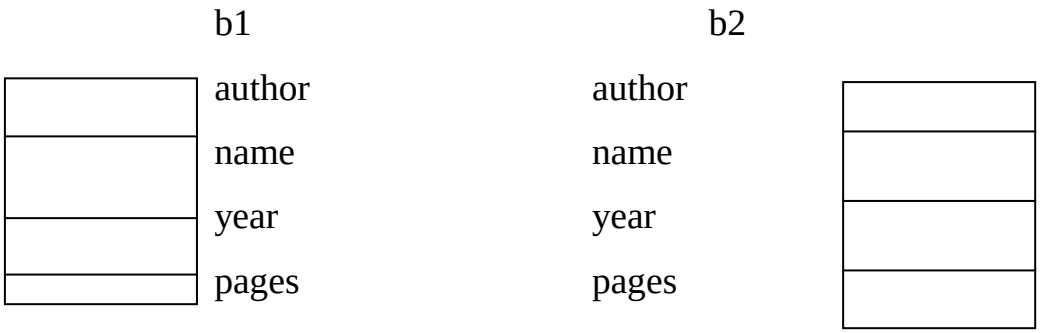
Описание структурной переменной имеет вид:

```
struct   имя_типа    имя_переменной;
```

Например:

```
struct  BOOK     b1,b2;
struct  BOOK     mb[100];   /* массив из 100 структур типа  BOOK  */
struct  BOOK      *p2 = &b2;   /* указатель на структуру типа  BOOK,
                                  ссылающийся на переменную b2      */
```

Каждая структурная переменная будет состоять из полей, перечисленных при описании типа структуры.




При объявлении структурных переменных можно их инициализировать, например:

*struct BOOK b3 = { "Толстой Л.Н.", "Война и мир", 1995, 950};*

Для однотипных структур определена операция присваивания.

Пример:

b1=b3;

mb[0]=b3;

Другие операции целиком над структурами выполнять нельзя. Например, сравнение структур или ввод/вывод нужно выполнять поэлементно.

Для обращения к отдельным элементам (полям) структурных переменных используется операция выбора "." или "->" :

имя_переменной . имя_элемента

имя_указателя -> имя_элемента

Пример 1

```
/* ввод значения структурной переменной  b2 типа BOOK*/
gets(b2.author);
gets (b2.name);
scanf("%d%d", &b2.year, &b2.pages);
```

Пример 2

```
struct BOOK  *p3 = &b3;
printf ("%s, %s, %d г., %d с.\n",  p3->author, p3->name,
p3->year, p3->pages);
```

Результат на экране:

Толстой Л.Н., Война и мир, 1995 г., 650 с.

Если структурные переменные используются только в одной функции программы, то можно совместить описание переменных с описанием типа. При этом имя типа можно не задавать, например:

```
struct
{ char author[40];        /* автор книги */
  char  name {50};        /* название */
  int year;               /* год издания */
  int pages;              /* число страниц */
 }  b1, b2, *p1;
```

Память под структурную переменную может быть выделена динамически, при выполнении программы с помощью функции *malloc()*, например:

```
struct   BOOK  *p4;
p4 = malloc (sizeof (struct  BOOK));
```

Такая структура имени не имеет; обращаются к ней с помощью указателя p4, например:

```
gets (p4->name);
```

## Работа с файлами

Файл – это поименованная область на диске, содержащая какую-либо информацию, например, текст программы, данные для программы, документ.

Файлы бывают текстовые и двоичные (бинарные). Текстовые файлы – это файлы, которые создаются или которые можно просмотреть с помощью текстовых редакторов. В операционной системе MS DOS текстовые файлы представляют собой последовательность символьных строк. Каждый символ занимает один байт. Строка заканчивается двумя символами: «возврат каретки» (с кодом 13) и «перевод строки» (с кодом 10).

Двоичные файлы содержат информацию во внутреннем представлении. Примером двоичного файла является exe-файл, содержащий программу в машинных командах. Прикладная программа тоже может создать двоичный

файл, записав в него данные в том виде, в каком они хранятся в памяти (к примеру, типа int).

При работе с файлами программа на языке Си должна вначале открыть каждый файл с помощью функции ***fopen()***. При этом для каждого файла создается структура типа ***FILE,*** содержащая всю необходимую информацию о файле. Тип такой структуры с именем FILE определен в файле stdio.h. В программе необходимо для каждого файла описать указатель на структуру типа FILE и присвоить ему значение, возвращаемое функцией fopen().

Для чтения информации из файла служат функции:

***fscanf()*** – форматированный ввод,

***fgets()*** – чтение одной строки,

***fgetc()*** – чтение одного символа,

***fread()*** – ввод заданного числа байтов (символов).

Для записи информации в файл используются функции:

***fprintf()*** – форматированный вывод,

***fputs()*** – вывод строки,

***fputc()*** – вывод одного символа,

***fwrite()*** – вывод заданного числа байтов (символов).

Обычно к файлу применяется последовательный доступ: чтение из файла и запись в файл происходят последовательно, например, в цикле по одной строке либо блоками байтов. При этом указатель текущей позиции файла каждый раз автоматически сдвигается на начало следующей строки или следующего блока. Однако возможен и прямой (произвольный) доступ к файлу. Он используется, когда файл состоит из записей одинаковой (фиксированной) длины. Для прямого доступа к файлу служит функция ***fseek()***. Она устанавливает указатель текущей позиции файла в указанное место.

После завершения работы с файлом его следует закрыть с помощью функции ***fclose()***.

**Некоторые функции доступа к файлам**

***fopen*** – открытие файла.

Прототип функции:

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

Первый параметр fname задает имя открываемого файла, второй – режим открытия файла или вид его обработки. Параметр mode может задаваться в виде:

**“r”** – чтение файла,

**“w”** – запись в файл (если файл существует, он стирается),

**“a”** – добавление информации в конец файла,

**“r+”** – чтение и запись.

Если файл открывается для добавления информации в конец файла, то указатель текущей позиции файла устанавливается в конец файла. В остальных случаях указатель устанавливается на начало файла.

Дополнительно во втором параметре можно указать символ ***t*** (текстовый режим) или ***b*** (двоичный режим). Например, “rb+” означает, что файл открывается для чтения и записи в двоичном режиме. Режимы отличаются только обработкой символов перехода на новую строку. В текстовом режиме при чтении строки комбинация символов «возврат каретки» и «перевод строки» заменяется одним символом перевода строки (при записи в файл выполняется обратное преобразование). В двоичном режиме эти преобразования не выполняются. По умолчанию файл открывается в текстовом режиме.

Функция fopen() создает структуру типа FILE с информацией о файле и возвращает адрес этой структуры. При попытке открыть несуществующий файл для чтения или чтения и записи функция возвращает пустую ссылку NULL. Если открывается несуществующий файл для записи или добавления в конец файла, файл создается.

При открытии файла с ним связывается область памяти, называемая буфером ввода/вывода. Чтение информации из файла осуществляется блоками, равными размеру буфера. Функции чтения читают данные из буфера. При выводе информации в файл данные вначале помещаются в буфер и только после заполнения буфера записываются в файл. Буферизация повышает скорость обмена информацией между оперативной памятью и внешними устройствами.

При аварийном завершении программы выходной буфер может быть не выгружен и возможна потеря данных.

**Пример**

```
FILE *fout, *fmod;     /* указатели на выходной и модифицируемый файлы */
char fname[13];        /* имя модифиц. файла */
fout = fopen (“f1.txt”, “w”);
puts(“Введите имя модифицируемого файла”);
gets (fname);
if  ((fmod = fopen(fname, “r+”) ==NULL)
{  puts (“Файл в текущем каталоге не найден”);
   exit (1);
}
```

***fclose*** – закрытие файла.

Функция имеет один параметр – указатель на файл.

**Пример**

```
fclose (fout);
fclose (fmod);
```

При закрытии файла информация из выходного буфера выгружается в файл. Поэтому следует все файлы закрывать явно.

***fgets*** – чтение строки файла.

Прототип функции:

**char * fgets (char *s, int n, FILE *f);**

Функция считывает символы из файла, на который ссылается указатель f, в строку с указателем s. Параметр n задает предельную длину считываемой строки: читается не более (n-1) символов. В конец строки добавляется нулевой байт (‘\0’). В отличие от функции gets() символ перевода строки сохраняется (если он был прочитан).

Функция возвращает адрес строки s или значение NULL при достижении конца файла.

**Пример**

```
char str[81];
FILE *f;
  …
while (fgets(str,81,f))
{   /* обработка строки */
   .
   .
   .
}
```

**Пример решения задачи**

**Задача.** Входной файл **st.txt** содержит сведения о сдаче студентами группы экзаменационной сессии. Каждая запись файла содержит фамилию и инициалы студента (15 символов) и пять оценок (5 символов) и завершается символом ”перевод строки”. Напечатать список студентов с указанием среднего балла каждого студента.

**Программа:**

```
   /*---------------------------------------------------*/
   /*  Печать среднего балла каждого студента */
   /*---------------------------------------------------*/
#include <stdio.h>
struct STUDENT
{ char fio[15];   /* фамилия и.о. */
```

```
  char oc[7];     /* 5 оценок + '\n' + '\0' */
};
void main()
{ FILE *f;           /* указатель на входной файл */
  struct  STUDENT  tz;  /* текущая запись файла */
  int i,
      s;             /* сумма оценок */
  if ((f= fopen("st.txt","r")) == NULL)
  { puts ("Файл st.txt не найден");
    return;
  }
  puts ("\nФамилия и.о.   Ср.балл");
  puts ("-----------------------------");
  while (fgets(&tz,sizeof(struct STUDENT),f)!=NULL)
  { for (i=0,s=0; i<5; i++)
       s+=tz.oc[i]-'0';
    tz.fio[14]='\0';
    printf("%s  %.1f\n", tz.fio, (float)s/5);
  }
  fclose(f);
  getch();
}
```

**Пример входного файла  st.txt:**

Анисимов А.И.  54435

Берхеев П.В.  55445

Вавилова С.Н.  45343

**Задание**

Введите и выполните программу. Создайте файл **st.txt** для проверки программы и снова запустите программу.

Измените файл **st.txt** так, чтобы поле фамилии занимало 20 позиций, а оценок было 4. Соответственно измените программу (для удобства задайте именованные константы). Проверьте результат ее работы. Объясните программу.

Измените программу, чтобы выводился список только тех студентов, у которых

а) средний балл ниже 4;

б) средний балл выше 4 и нет двоек;

в) только хорошие и отличные оценки;

г) есть двойки;

д) все пятерки;

е) больше одной двойки.

Добавьте определение среднего балла группы.

## Лабораторная работа 9

## Стеки

Стек - это динамическая структура данных, из которой элементы удаляются в порядке, обратном их поступлению ("последним пришел - первым вышел").

Хранить стек можно в виде вектора и указателя на последний записанный в стек элемент (указателя верхушки стека). Если стек пустой, указатель равен -1. При добавлении нового элемента в стек указатель увеличивается на 1, а при удалении элемента из стека -уменьшается на 1.

**Пример**

**Задача.** Дано арифметическое выражение длиной до 80 символов, оканчивающееся пробелом. Выражение содержит целые числа без знака и знаки операций +, -, *, /. Вычислить значение выражения.

Например, входной текст: 130+25*3-160/20*6 .

Результат: 157.

Метод решения задачи основан на использовании стека операндов, операций и приоритетов. Операции * и / имеют одинаковый приоритет, причем более высокий, чем операции + и - . При просмотре выражения происходит выделение очередного операнда (числа), преобразование его из символьной формы в целочисленную и запись в стек полученного числа, следующей за ним операции и присвоенного ей приоритета. Пока в стеке более одного элемента и приоритет последней операции оказывается не выше приоритета предыдущей операции в стеке, происходит выполнение соответствующих операций над двумя последними операндами стека и удаление ненужных элементов из стека. В конце концов, когда просмотр выражения закончится, первый элемент стека и будет искомым результатом.

В программе стек реализован в виде трех параллельных массивов.

**Программа:**

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

```
   /*-------------------------------------------------------*/
   /*   Функция выделения числа и преобразования его из     */
   /*    символьной формы представления в целочисленную  */
   /*-------------------------------------------------------*/
int chislo ( char text [], int *i )
     /* Входные данные:                                     */
     /*      text - символьная строка,                      */
     /*      *i - индекс первой цифры числа в строке text.  */
     /* Выходные данные:                                    */
     /*      *i - индекс следующего после числа символа.  */
     /* Функция возвращает целое число                      */
{   int c=0;   /* возвращаемое значение  */
    while (text[*i]>='0' && text[*i]<='9')
    { c=c*10+(text[*i]-'0');
      (*i)++;
    }
    return c;
 }
      /*-----------------------------------*/
      /*       Главная функция             */
      /*-----------------------------------*/
int  main()
{   char  text [80]; /* вх. текст - арифм. выражение */
    int   i ;            /* индекс массива text */
    float  opd [3];  /* стек операндов   */
    char   opc [3];  /* стек операций    */
    int    pr [3] ;     /* стек приоритетов */
    int    j = -1 ;     /* указатель стека */
    printf ("\nВведите арифметическое выражение.\n");
    gets (text);
```

```
    i=-1;
    do
    {  i++;
        /* запись числа, операции и ее приоритета в стек */
       opd[++j]=chislo(text,&i); opc[j]=text[i];
       switch  (text[i])
       {
          case '+':
          case '-': pr[j]=1; break;
          case '*':
          case '/': pr[j]=2; break;
          case ' ' /* пробел */: pr[j]=0;
       }
       while  (j>0 && pr[j]<=pr[j-1])
       {      /* выполнение соответствующей операции */
          switch  (opc[j-1])
          {  case '+' : opd[j-1]=opd[j-1]+opd[j]; break;
             case '-' : opd[j-1]=opd[j-1]-opd[j]; break;
             case '*' : opd[j-1]=opd[j-1]*opd[j]; break;
             case '/' : opd[j-1]=opd[j-1]/opd[j];
          }
             /* удаление выполненной операции из стека */
          opc[j-1]=opc[j];  pr[j-1]=pr[j];  j=j-1;
       }
    }
    while (text[i]!=' ');
    printf ("Результат:%.2f\n",opd[0]);
    getch();
}
```

**Результаты тестирования**

Введите арифметическое выражение.

9/4-12

Результат:-9.75

Введите арифметическое выражение.

10*2/5+6/3

Результат:6.00

Введите арифметическое выражение.

130+25*3-160/20*6

Результат:157.00

Введите арифметическое выражение.

2345

Результат:2345.00

**Задания**

**1.** Дано арифметическое выражение длиной до 20 символов, оканчивающееся пробелом. Выражение содержит однобуквенные идентификаторы и знаки операций +, -, *, /. Преобразовать выражение в обратную польскую запись, используя стек операций и приоритетов.

Пример.

Вх. текст: a*b+c/d-e

Результат: ab*cd/+e-

**2**. Дано арифметическое выражение в постфиксной (обратной польской) записи длиной до 20 символов, оканчивающееся пробелом. Получить эквивалентную последовательность элементарных присваиваний, содержащих одну арифметическую операцию (используя стек операндов).

**а)** В левой части операторов присваивания использовать вспомогательные переменные R1,R2,R3,.

Пример.

Вх. текст: abc*+def+/-

Результат: R1:=b*c

R2:=a+R1

R3:=e+f

R4:=d/R3

R5:=R2-R4

**б)** В левой части операторов присваивания использовать вспомогательные переменные R1,R2,R3,..., сократив до минимума число этих переменных.

Пример.

Вх. текст: abc*+def+/-

Результат: R1:=b*c

R1:=a+R1

R2:=e+f

R2:=d/R2

R2:=R1-R2

**3.** Дано арифметическое выражение длиной до 30 символов, оканчивающееся знаком равенства. Выражение содержит знаки операций +, -, *, / и однозначные целые числа и представлено в обратной польской записи. Вычислить значение выражения, используя стек операндов.

Пример.

Вх. текст: 345+2*63/-+=19

результат

**4.** Дано арифметическое выражение длиной до 40 символов, оканчивающееся знаком равенства. Выражение содержит знаки операций +, -, / и однозначные целые числа. Вычислить значение выражения, используя стек операндов, операций и приоритетов.

Пример.

Вх. текст: 5-2+8/4-6/2=2

Результат

**5.** Дан ор.граф без циклов в виде количества вершин n<=10 и матрицы смежности.

а) Проверить, существует ли путь от вершины A к вершине B.

б) Найти какой-нибудь путь от начальной вершины к конечной, если такой существует.

**Указание.** Для хранения очередного пути при обходе графа использовать стек.

**6.** Дан орграф в виде количества вершин n<=10 и матрицы смежности.

а) Проверить, существует ли цикл, проходящий через заданную вершину А.

б) Найти какой-нибудь цикл, проходящий через начальную вершину, если такой существует.

Указание. Для хранения очередного пути при обходе графа использовать стек.

**7.** Дано скобочное выражение длиной до 50 символов, оканчивающееся пробелом. Напечатать попарно порядковые номера соответствующих открывающих и закрывающих скобок в выражении.

Пример.

1 4 8 10 14 19 27

Вх. текст: (a+(b+c)/(a-b)*(x+(y+2)**3))/2

Результат: 4 8

10 14

19 23

16 27

1 28

Указание. Для запоминания номеров открывающих скобок использовать стек.

**8**. Дана последовательность чисел, оканчивающаяся нулем.

а) Напечатать только отрицательные числа из этой последовательности, причем, если подряд идет несколько отрицательных чисел, печатать их в обратном порядке.

Пример.

Вх. последовательность: 5 -1 -20 -8 10 14 -3 5 -2 -13 -7 0

Результат: -8 -20 -1 -3 -7 -13 -2

Указание. Последовательность подряд расположенных отрицательных чисел помещать в стек.

б) Напечатать только положительные числа из этой последовательности, причем, если подряд идет несколько положительных чисел, печатать их в обратном порядке.

Пример.

Вх. последовательность: -2 5 10 20 15 -4 -6 2 -10 3 1 4 -5 0

Результат: 15 20 10 5 2 4 1 3

Указание. Последовательность подряд расположенных положительных чисел помещать в стек.

**9.** Дана последовательность из n слов (n<=20) в алфавитном порядке. Длина каждого слова не более 10 букв. Напечатать слова, начинающиеся с одной и той же буквы в обратном порядке, используя стек.

Пример.

| Вх. последовательность: | Результат: |
|---|---|
| Август | апрель |
| Апрель | август |
| Май | море |
| Март | март |
| Море | май |
| Лето | |

## Литература

1. Немцова, Т. И. Программирование на языке высокого уровня. Программирование на языке С++ : учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Те-рентьев ; под ред. Л.Г. Гагариной. — Москва : ФОРУМ : ИНФРА-М, 2023. — 512 с. + Доп. материалы [Электронный ресурс]. — (Среднее профессиональное обра-зование). - ISBN 978-5-8199-0699-6. - Текст : электронный. - URL: https://znanium.com/catalog/product/1916204 (дата обращения: 28.06.2023). – Режим доступа: по подписке.

2. Рейзлин, В. И. Язык С++ и программирование на нём : учебное пособие / В. И. Рейзлин. — 3-е изд., перераб. — Томск : ТПУ, 2021. — 206 с. — ISBN 978-5-4387-0975-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/246239 (дата обращения: 28.06.2023). — Режим доступа: для авториз. пользователей.

3. Ефимова Ю.В. Программирование на языке высокого уровня: Практикум. - Казань: Изд-во Казан. гос. техн. ун-та, 2012. - 32 с.

4. Чукич, И. Функциональное программирование на С++ : учебное пособие / И. Чукич ; перевод с английского В. Ю. Винника, А. Н. Киселева. — Москва : ДМК Пресс, 2020. — 360 с. — ISBN 978-5-97060-781-7. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/140597 (дата обращения: 28.06.2023). — Режим досту-па: для авториз. пользователей.

5. Программирование на языке Си/А.В.Кузин, Е.В.Чумакова - М.: Форум, НИЦ ИНФРА-М, 2015. - 144 с. - Режим доступа: http://znanium.com/bookread2.php?book=505194

6. Язык Си: кратко и ясно: Учебное пособие / Д.В. Парфенов. - М.: Альфа-М: НИЦ ИНФРА-М, 2014. - 320 с. - Режим доступа: https://znanium.com/read?id=356040

7. Программирование графики на С++. Теория и примеры : учеб. пособие / В.И. Корнеев, Л.Г. Гагарина, М.В. Корнеева. — М. : ИД «ФОРУМ» :

ИНФРА-М, 2017. — 517 с. - Режим доступа: http://znanium.com/bookread2.php?book=562914

8. Программирование на языке С++: Учебное пособие / Т.И. Немцова, С.Ю. Голова, А.И. Терентьев; Под ред. Л.Г. Гагариной. - М.: ИД ФОРУМ: ИНФРА-М, 2012. - 512 с. - Режим доступа: http://znanium.com/bookread2.php?book=244875