

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение высшего профессионального образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

УТВЕРЖДАЮ
Декан ЭФФ

_____ Г.С. Евтушенко
« __ » _____ 2010 г.

В.В. Шестаков

Введение в язык «Си»

Методические указания к выполнению лабораторных работ
по курсу «Компьютерные технологии в приборостроении»
для студентов II курса, обучающихся по направлению 200100
«Приборостроение»

Издательство
Томского политехнического университета
2010

УДК 681.3.06:8.82(076.5)

ББК 32.973-018.1.я73

Ш514

Шестаков В.В.

- Ш514 Введение в язык «Си»: методические указания к выполнению лабораторных работ по курсу «Компьютерные технологии в приборостроении» для студентов II курса, обучающихся по направлению 200100 «Приборостроение» / В.В. Шестаков; Национальный исследовательский Томский политехнический университет. – Томск: Изд-во Томского политехнического университета, 2010. – 57 с.

УДК 681.3.06:8.82(076.5)

ББК 32.973-018.1.я73

Методические указания рассмотрены и рекомендованы
к изданию методическим семинаром кафедры
физических методов и приборов контроля качества ЭФФ
«__»_____ 2010 г.

Зав. кафедрой ФМПК
доктор технических наук

_____ *О.А.Сидуленко*

Председатель учебно-методической
комиссии

_____ *А.Н.Гормаков*

Рецензент

Кандидат технических наук, доцент кафедры информационно
измерительной техники ЭФФ ТПУ

В.В. Ширяев

© ГОУ ВПО «Национальный исследовательский
Томский политехнический университет», 2010
© Шестаков В.В., 2010

Лабораторная работа №1: «Использование условных операторов»

Цель работы:

Получение навыков написания простейших программ на языке Си.

Знакомство с операторами ввода вывода и условными операторами.

Создание блок – схем алгоритмов, содержащих ветвящиеся структуры.

Краткие сведения по языку СИ

Структура простейшей программы:

Общая структура программы на Си/Си++ следующая:

```
директивы_препроцессора
определение_функции_1
определение_функции_2
...
определение_функции_N
директивы_препроцессора
void main()
{ определения_объектов;
  исполняемые_операторы;
}
```

Пример 1. Дано: a , b , c — стороны треугольника. Вычислить S — площадь треугольника. По формуле Герона:

$$S = \sqrt{p(p-a)(p-b)(p-c)}, \quad \text{где } p \text{ — полупериметр треугольника.}$$

```
#include <stdio.h>
#include <math.h>
void main () {
    float a,b,c,p,s;
    printf("\na="); scanf("%f",&a);
    printf("\nb="); scanf("%f",&b);
    printf("\nc="); scanf("%f",&c);
    p=(a+b+c)/2;
    s=sqrt (p * (p-a) * (p-b) * (p-c) ) ;
    printf("\nПлощадь треугольника=%f", s) ; }
```

Программа состоит из одной главной функции со стандартным именем main. Слово void обозначает отсутствие какого-либо

возвращаемого этой функцией результата, а пустые скобки — отсутствие у нее аргументов. Все, что следует после заголовка функции и заключено в фигурные скобки, можно назвать телом функции. Первая строка — объявление используемых переменных. Все они имеют тип `double`. Обратите внимание на то, что объявление переменных заканчивается точкой с запятой.

Дальнейшая часть программы — исполняемые операторы. Среди них операторы вывода на экран, ввода данных с клавиатуры, операторы присваивания.

В рассматриваемой программе присутствуют два оператора присваивания: вычисления полупериметра (p) и вычисления площади треугольника (S).

В выражении для вычисления площади используется библиотечная функция `sqrt()` — квадратный корень. Данная функция относится к *библиотеке математических функций*. Для подключения этой библиотеки к нашей программе используется директива препроцессора `#include <math.h>`. Здесь `math.h` — имя заголовочного файла этой библиотеки. Описания математических функций можно найти в справочной системе

В рассматриваемой программе операторы `printf (...)`; и `scanf (...)` ; реализуют соответственно вывод на экран и ввод исходных данных с клавиатуры. Они осуществляют обращение к соответствующим функциям стандартной библиотеки ввода-вывода Си, заголовочный файл которой имеет имя `stdio.h`.

Форматированный вывод на экран.

Оператор вызова функции `printf()` имеет следующую структуру:

```
printf(форматная строка, список_аргументов);
```

Форматная строка ограничена двойными кавычками (т.е. является текстовой константой) и может включать в себя *произвольный текст*, *управляющие символы* и *спецификаторы формата*. Список аргументов может отсутствовать или же состоять из выражений, значения которых выводятся на экран (в частном случае из констант и переменных).

В примере 1 оператор `printf ("\\na=") ;` содержит текст (`"a="`) и управляющие символы (`"\\n"`). Текст выводится на экран в том виде, в котором он записан. Управляющие символы влияют на расположение на экране выводимых знаков. В результате выполнения этого оператора на экран с новой строки выведутся символы `a=`.

Признаком управляющего символа является значок `\\`. Ниже

приводится их список:

- \n – перевод строки
- \t – горизонтальная табуляция
- \r – возврат курсора к началу новой строки
- \a – звуковой сигнал
- \b – возврат на один символ влево
- \f – перевод страницы
- \v – вертикальная табуляция

Оператор:

```
printf ("\nПлощадь треугольника=%f", s);
```

содержит все виды параметров функции printf. Список аргументов состоит из одной переменной s. Ее значение выводится на экран. Пара символов %f является спецификацией формата выводимого значения переменной s. Значок % — признак формата, а буква f указывает на то, что выводимое число имеет вещественный (плавающий) тип и выводится на экран в форме с фиксированной точкой. Например, если в результате вычислений переменная s получит значение 32,435621, то на экран выведется:

Площадь треугольника=32.435621

Спецификатор формата определяет форму внешнего представления выводимой величины. Вот некоторые спецификаторы формата:

- %c – символ
- %s – строка
- %d – целое десятичное число
- %u – целое десятичное число без знака
- %f – вещественное число
- %e – вещественное число с плавающей точкой
(мантисса и порядок)

Например, после выполнения следующих операторов

```
float m,p;  
int k;  
m=84.3; k=-12; p=32.15;  
printf("\nm=%f \t k=%d \t p=%e",m,p,k);
```

на экран выведется строка:

m=84.299999 k=-12 p=3.21500e+01

Форматированный ввод с клавиатуры.

Оператор `scanf()` имеет следующую структуру:

```
scanf(форматная_строка, список_аргументов);
```

Данная функция осуществляет чтение символов, вводимых с клавиатуры, и преобразование их во внутреннее представление в соответствии с типом величин. В функции `scanf()` форматная строка и список аргументов присутствуют обязательно. В программе из примера 1 имеется оператор:

```
scanf("%f", &a);
```

Здесь `"%f"` - форматная строка; `&a` — список аргументов, состоящий из одного элемента. Этот оператор производит ввод числового значения в переменную `a`.

Символьную последовательность, вводимую с клавиатуры и воспринимаемую функцией `scanf()`, принято называть *входным потоком*. Функция `scanf()` разделяет этот поток на отдельные вводимые величины, интерпретирует их в соответствии с указанным типом и форматом и присваивает переменным, содержащимся в списке аргументов.

Список аргументов — это перечень вводимых переменных, причем перед именем каждой переменной ставится значок `&`. Это знак операции «взятие адреса переменной». Подробнее смысл этого действия будет объяснен позже, а пока примем это правило формально.

Форматная строка заключается в кавычки (как и для `printf`) и состоит из списка спецификаций. Каждая спецификация начинается со знака `%`, после которого могут следовать

*ширина поля модификатор спецификатор

Из них обязательным элементом является лишь спецификатор. Для ввода числовых данных используются следующие спецификаторы:

- `d` — для целых десятичных чисел (тип `int`);
- `u` — для целых десятичных чисел без знака (тип `unsigned int`);
- `f` — для вещественных чисел (тип `float`) в форме с фиксированной точкой;
- `e` — для вещественных чисел (тип `float`) в форме с плавающей точкой.

Звездочка в спецификации позволяет пропустить во входном потоке определенное количество символов. *Ширина поля* — целое положительное число, позволяющее определить число символов из входного потока, принадлежащих значению соответствующей вводимой переменной. Как и в спецификациях вывода для функции `printf()`, в спецификациях ввода функции `scanf()` допустимо использование *модификаторов* `h`, `l`, `L`. Они применяются при вводе значений модифицированных типов:

`hd` — для ввода значений типа `short int`;
`ld` — для ввода значений типа `long int`;
`lf`, `le` — для ввода значений типа `double` в форме с фиксированной и плавающей точкой;
`Lf`, `Le` — для ввода значений типа `long double` в форме с фиксированной и плавающей точкой.

В программе из примера 1 все три величины *a*, *b*, *c* можно ввести одним оператором:

```
scanf ("%f%f%f", &a, &b, &c);
```

Если последовательность ввода будет такой:

```
5 3.2 2.4 <Enter>
```

то переменные получат следующие значения: *a* = 5,0, *b* = 3,2, *c* = 2,4. Разделителем в потоке ввода между различными значениями может быть любое количество пробелов, а также другие пробельные символы: знак табуляции, конец строки. Только после нажатия на клавишу **Enter** вводимые значения присвоятся соответствующим переменным. До этого входной поток помещается в буфер клавиатуры и может редактироваться.

Программирование ветвлений

Для программирования ветвящихся алгоритмов в языке Си имеется несколько различных средств. К ним относятся рассмотренная выше операция условия `?:`, условный оператор `if` и оператор выбора `switch`.

Условный оператор. Формат условного оператора следующий:

if (выражение) оператор1; **else** оператор2;

Это полная форма оператора, программирующая структуру полного ветвления. Обычно *выражение* — это некоторое условие, содержащее операции отношения и логические операции. Значение выражения приводится к целому и интерпретируется в соответствии с правилом: равно нулю — ложь, не равно нулю — истина. Если выражение истинно, выполняется оператор1, если ложно — оператор2.

Необходимо обратить внимание на следующие особенности синтаксиса условного оператора:

- выражение записывается в круглых скобках;
- точка с запятой после оператора 1 ставится обязательно; (правило Паскаля — не ставить точку с запятой перед **else** — здесь не работает.)

Возможно использование неполной формы условного оператора

if (выражение) оператор;

Вот пример использования полной формы условного оператора для нахождения большего значения из двух переменных *a* и *b*:

if (a>b) max=a; **else** max=b;

Та же самая задача может быть решена с использованием неполного ветвления следующим образом:

max=a; **if** (b>a) max=b;

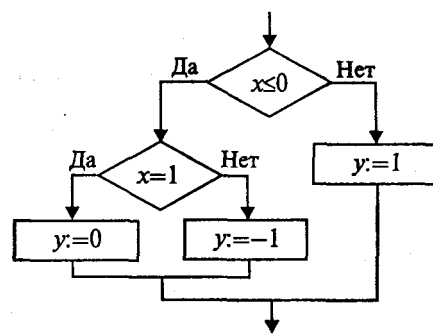
Условие может быть **сложным**. Например, логическое выражение, соответствующее системе неравенств $0 < x < 1$ в программе на Си запишется в виде следующего логического выражения:

(x>0 && x<1)

Теперь рассмотрим примеры программирования вложенных ветвящихся структур. Требуется вычислить функцию $\text{sign}(x)$ — знак *x*, которая определена следующим образом:

$$\text{sign}(x) = \begin{cases} -1, & \text{если } x < 0, \\ 0, & \text{если } x = 0, \\ 1, & \text{если } x > 0. \end{cases}$$

Пример 1. Алгоритм с полными вложенными ветвлениями:

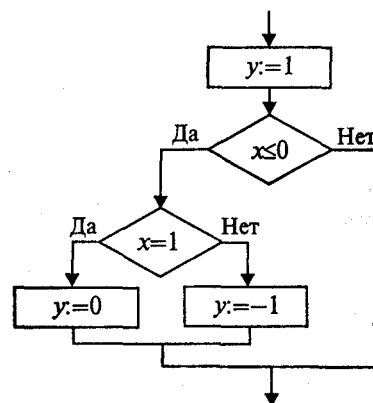


```

if (x ≤ 0)
  if (x == 0) y = 0;
  else y = -1;
else y = 1;
  
```

Рис. 1

Пример 2. Алгоритм с неполным ветвлением:



```

y = 1;
if (x ≤ 0)
  if (x == 0) y = 0;
  else y = -1;
  
```

Рис. 2

Приложение 1:

Блок-схемы программ

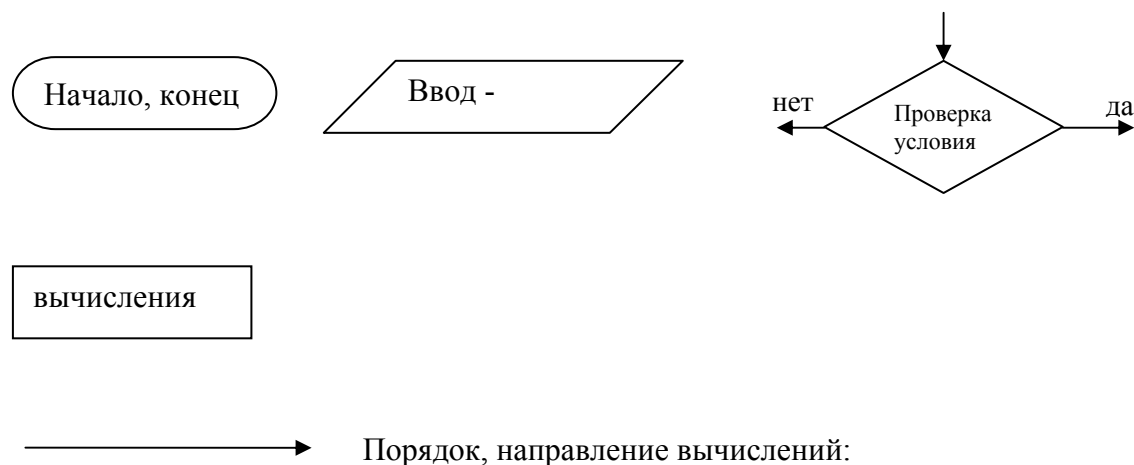


Рис. 3 Основные элементы блок схем

Основные правила составления блок – схем:

- разветвление алгоритма на два потока возможно только при помощи элемента «Проверка условия»
- слияние потоков – без ограничений
- соединительные линии рисуются под углом кратным 90° .

Примеры простейших алгоритмов:

1.) Определение максимального из 2-х чисел. (Рис. 4)

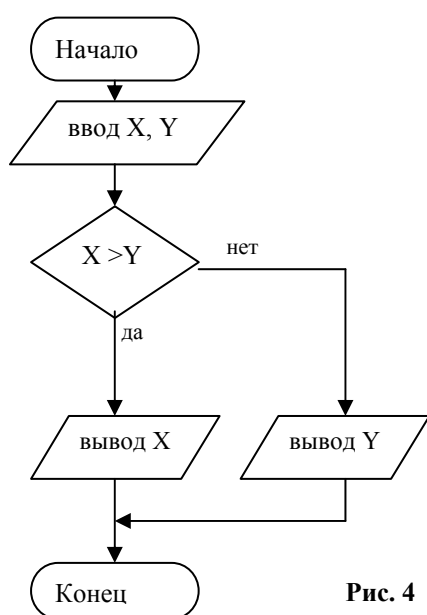


Рис. 4

2.) Определение максимального из трех чисел (Рис. 5)

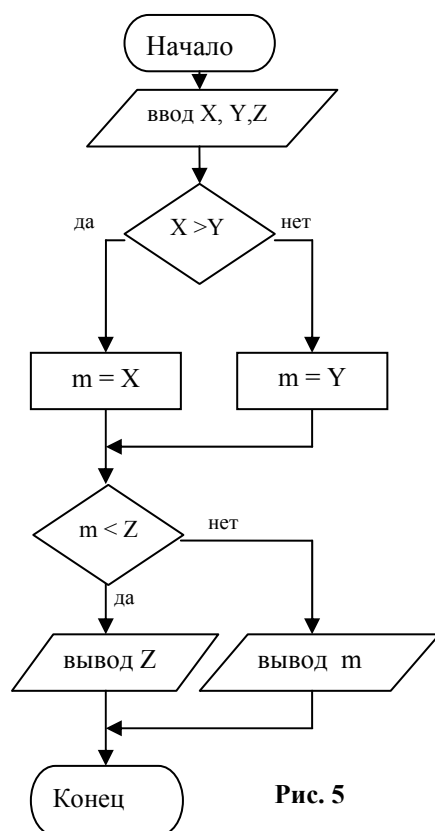


Рис. 5

Контрольные вопросы:

1. Найдите ошибку в следующем фрагменте:
`if (a > b); c=d else d=c;`
2. Каковы правила создания сложных условий, например проверка двойных неравенств вида: $(a < X < b)$?
3. Является ли слово **else** обязательной частью оператора **if**?
4. Без использования условного оператора присвоить переменной **b** значение 1 если числа **x** и **y** равны, и значение 0 иначе.
5. Является ли верным следующий оператор:
`if (a) printf("a=%d", a) else prntf("0");`
если **a** целочисленная переменная.
6. Допустимо ли создавать вложенные конструкции из условных операторов
7. Какие части программы на языке СИ являются обязательными?
Напишите самую короткую программу без действующих операторов.

Индивидуальные задания по теме: «Условные операторы»

1. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы.

$$a = \begin{cases} (x+y)^2 - \sqrt{x*y}; & \text{при } x*y > 0 \\ (x+y)^2 + \sqrt{x*y}; & \text{при } x*y < 0 \\ (x+y)^2 + 1; & \text{при } x*y = 0 \end{cases} \quad \text{Исходные данные : } x, y$$

2. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы.

$$a = \begin{cases} \ln(x/y) + (x^2 + y)^3; & \text{при } x/y > 0 \\ \ln(|x/y|) + (x^2 + y)^3; & \text{при } x/y < 0 \\ (x^2 + y)^3; & \text{при } x = 0 \\ 0; & \text{при } y = 0 \end{cases}$$

Исходные данные : x, y

3. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы.

$$a = \begin{cases} (x+y)^2 - \sin(x); & \text{при } x-y = 0 \\ (x-y)^2 + \cos(x); & \text{при } x-y > 0 \\ (x-y)^2 + \operatorname{tg}(x); & \text{при } x-y < 0 \end{cases}$$

Исходные данные : x, y

4. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы

$$d = \begin{cases} (x-y)^3 + \operatorname{arctg}(x); & \text{при } x > y \\ (y-x)^3 + \operatorname{arctg}(x); & \text{при } y > x \\ (x+y)^3 + \operatorname{tg}(x); & \text{при } x = y \end{cases} \quad \text{Исходные данные : } x, y$$

5. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы.

$$d = \begin{cases} e^2 + 12\sqrt{a*b}; & \text{при } 0.5 < a*b < 10 \\ \sqrt{|a*b|}; & \text{при } 0.1 < a*b < 0.5 \\ 2*b^2; & \text{иначе} \end{cases}$$

Исходные данные : a, b.

6. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы.

$$d = \begin{cases} \arctg(x + |y|); & \text{при } x < y \\ \arctg(|x| + y); & \text{при } x > y \\ (x + y)^2; & \text{при } x = y \end{cases}$$

Исходные данные : x, y

7. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы.

$$a = \begin{cases} i * \sqrt{a}; & \text{при } a > 0, i - \text{нечетное} \\ i / 2 * \sqrt{|a|}; & \text{при } a < 0, i - \text{четное} \\ \sqrt{i * a}; & \text{иначе} \end{cases}$$

Исходные данные : i, a.

8. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы.

$$a = \begin{cases} \sin(5*k + 3*m*|k|); & \text{при } 1 < k < m \\ \cos(5*k + 3*m*|k|); & \text{при } k > m \\ k^3; & \text{при } k = m \end{cases}$$

Исходные данные : k, m.

9. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы.

$$a = \begin{cases} \sin(5 * k + 3 * m * |k|); & \text{при } -1 < k < m \\ \cos(5 * k + 3 * m * |k|); & \text{при } 10 > k > m \\ k^3; & \text{при } k = m \end{cases}$$

Исходные данные : k, m .

10. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы.

$$a = \begin{cases} i * \sqrt{a} + \sqrt[3]{a}; & \text{при } a > 0, i - \text{нечетное} \\ i / 2 * \sqrt{|a|}; & \text{при } a < 0, i - \text{четное} \\ \sqrt{i * |a|}; & \text{иначе} \end{cases}$$

Исходные данные : i, a

11. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы.

$$d = \begin{cases} e^2 + 12\sqrt{a * b}; & \text{при } 0.5 < a * b < 10 \\ \sqrt{|a * b|}; & \text{при } 2 < a * b < 3 \\ 2 * b^2; & \text{иначе} \end{cases}$$

Исходные данные : a, b .

12. Разработать программу вычисления выражения и вывода полученного результата на экран. Соответствующие исходные данные ввести с клавиатуры. Составить блок-схему программы.

$$a = \begin{cases} i * \sqrt{(a + i) / 2}; & i - \text{нечетное} \\ i / 2 * \sqrt{|a|}; & i - \text{четное} \\ \sqrt{i^4 * \sqrt[3]{a}}; & \text{иначе} \end{cases}$$

Исходные данные : i, a

Лабораторная работа №2: «Программирование циклических алгоритмов»

Цель работы:

Изучение правил применения операторов цикла. Научиться применять циклические алгоритмы для различных классов вычислительных задач. Получение практических навыков отладки программ.

Общие сведения

В Си, существуют три типа операторов цикла: цикл с предусловием, цикл с постусловием и цикл с параметром.

Цикл с предусловием.

Формат оператора цикла с предусловием:

```
while (выражение) оператор;
```

Цикл повторяет свое выполнение, пока значение выражения отлично от нуля, т. е. заключенное в нем условие цикла истинно.

В качестве примера использования оператора цикла рассмотрим программу вычисления факториала целого положительного числа $N!$.

// Программа вычисления факториала

```
#include <iostream.h>
void main() {
    long int F; int i,N;
    cout<<"N="; cin>>N;
    F=i=1;
    while(i<=N) F*=i++;
    cout<<"\n"<<N<<"!="<<F;
}
```

Другой пример:

```
while(1);
```

Это бесконечный пустой цикл. Использование в качестве выражения константы 1 приводит к тому, что условие повторения цикла все время остается истинным и работа цикла никогда не заканчивается. Тело в этом цикле представляет собой *пустой оператор*. При исполнении такого оператора программа будет «топтаться на месте».

Цикл с постусловием.

Формат оператора цикла с постусловием:

do оператор while (выражение);

Цикл выполняется до тех пор, пока выражение отлично от нуля, т.е. заключенное в нем условие цикла истинно. Выход из цикла происходит после того, как значение выражения станет ложным, иными словами равным нулю. **Пример:**

```
#include <iostream.h>
void main() {
long int F; int i,N;
cout<<"N="; cin>>N;
F=i=1;
do F*=i++; while(i<=N);
cout<<"\n"<<N<<"!="<<F;
}
```

Цикл с параметром.

Формат оператора цикла с параметром:

```
for      (выражение_1;      выражение_2;      выражение_3)
оператор;
```

Выражение 1 - выполняется только один раз в начале цикла. Обычно оно определяет начальное значение параметра цикла (инициализирует параметр цикла).

Выражение 2 - условие выполнения цикла,

Выражение 3 - обычно определяет изменение параметра цикла,

оператор — тело цикла, которое может быть простым или составным. В последнем случае используются фигурные скобки.

Алгоритм выполнения цикла **for** представлен на блок-схеме на Рис. 6

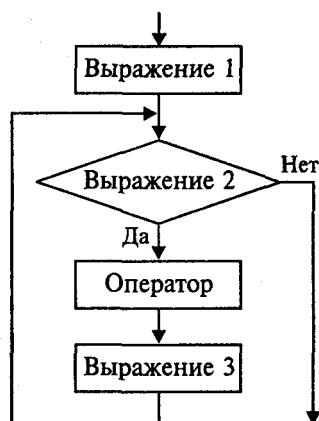


Рис. 6

Обратите внимание на то, что после вычисления выражения 3 происходит возврат к вычислению выражения 2 — проверке условия повторения цикла.

С помощью цикла **for** нахождение $N!$ можно организовать следующим образом:

```
F=1;
for (i=1; i<=N; i++) F*=i;
```

Некоторых элементов в операторе **for** может не быть, однако разделяющие их точки с запятой обязательно должны присутствовать. В следующем примере инициализирующая часть вынесена из оператора **for**:

```
F=1;
i=1;
for (; i<=N; i++) F=F*i;
```

Ниже показан еще один вариант вычисления $N!$. В нем на месте тела цикла находится пустой оператор, а вычислительная часть внесена в выражение 3.

```
for (F=1, i=1; i<=N; F=F*i, i++) ;
```

Этот же оператор можно записать в следующей форме:

```
for (F=1, i=1; i<=N; F*=i++) ;
```

В языке Си оператор **for** является достаточно универсальным средством для организации циклов. С его помощью можно программировать даже итерационные циклы, что невозможно в Паскале. Вот пример вычисления суммы элементов гармонического ряда, превышающих заданную величину ϵ :

```
for (n=1, S=0; 1.0/n>eps && n<INT_MAX; n++) S+=1.0/n;
```

И наконец, эта же самая задача с пустым телом цикла:

```
for (n=1, S=0; 1.0/n>eps && n<INT_MAX; S+=1.0/n++) ;
```

Следующий фрагмент программы на Си++ содержит два вложенных цикла **for**. В нем запрограммировано получение на экране таблицы умножения.

```
for (x=2; x<=9; x++)  
    for (y=2; y<=9; y++)  
        cout<<"\n"<<x<<"*"<<y<<"="<<x*y;
```

На экране будет получен следующий результат:

```
2*2=4  
2*3=6  
.  
9*8=72  
9*9=81
```

Оператор continue. Если выполнение очередного шага цикла требуется завершить до того, как будет достигнут конец тела цикла, используется оператор **continue**. Следующий фрагмент программы обеспечивает вывод на экран всех четных чисел в диапазоне от 1 до 100:

```
for (i=1; i<=100; i++) {  
    if (i%2) continue; cout<<"\t"<<i;  
}
```

Для нечетных значений переменной *i* остаток от деления на 2 будет равен единице, этот результат воспринимается как значение «истина» в условии ветвления, и выполняется оператор **continue**. Он завершит очередной шаг цикла, выполнение цикла перейдет к следующему шагу.

Оператор break. Используется для досрочного прерывания циклов и выполняет переход к оператору, следующему за телом цикла.

Отладка программ

В процессе создания программ неизбежно появляются ошибки. Если ошибки связаны с нарушением синтаксиса языка программирования, их обнаруживает компилятор. Например, начинающие программисты часто забывают ставить *точку с запятой* в конце инструкций программы. Более сложные ошибки связаны с неверной логикой работы, что приводит к неверным результатам или незапланированному поведению программы.

Поскольку ошибки бывают всегда, системы разработки программ имеют средства, помогающие обнаруживать ошибки. Познакомимся с

такими средствами, имеющимися в Turbo C++ на примере простейшей программы, в которой есть синтаксическая и логическая ошибка.

Программа 1. Деление чисел

Загрузим среду программирования, выполним команду **File, New**. В появившемся пустом окне введем следующую программу:

```
#include <iostream.h>
int main()
{
    int a, b, c;           //Определение переменных
    a = 1;                 //Присваивание значений
    b = 0;                 //переменным
    c = a / b              //Деление чисел
    cout << "c = " << c;  //Вывод частного
    return 0;
}
```

Здесь определяются три переменные целого типа **a**, **b** и **c**. Сначала пишется тип величин **int**, а затем перечисляются через запятую имена создаваемых переменных.

Начальные значения переменные получают с помощью оператора присваивания, который обозначается знаком **=**.

Далее вычисляется частное величин **a** и **b** с помощью оператора деления **/**, результат присваивается **c** и выводится.

В тексте программы сознательно допущена ошибка, которую будем сейчас искать.

Выполним команду **File, Save** и сохраним программу в файле **Demoerr.cpp**.

Синтаксические ошибки

Выполним компиляцию, нажав **Alt+F9**. Компилятор выдаст информационное окно, Рис. 7.

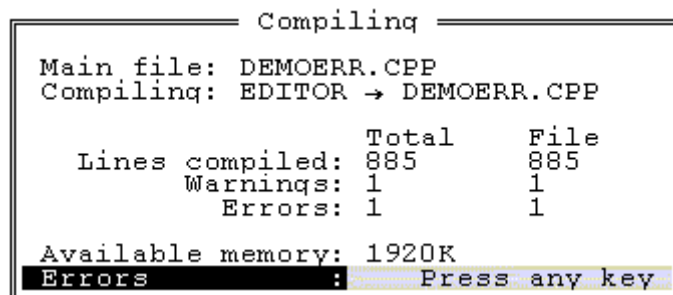


Рис. 7. Информация о результатах компиляции

В информационном окне указано имя компилируемого файла, количество откомпилированных строк (**Lines compiled**), количество предупреждений (**Warnings**) и ошибок (**Errors**). В самой программе

только 10 строк, но в их общее число 885 включаются и строки файла **iostream.h**.

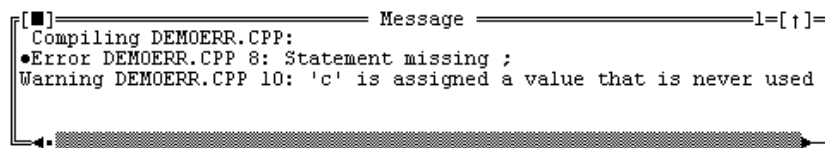


Рис. 8. Окно с сообщениями об ошибках

После нажатия любой клавиши (Press any key), информационное окно закрывается и становится активным окно сообщений, Рис. 8 В нем перечислены найденные ошибки и предупреждения. Сообщение об ошибке состоит из имени файла, номера строки в файле, где обнаружена ошибка, и краткого ее описания. Если выбрать какое-либо сообщение об ошибке и нажать клавишу **F1**, будет показана более подробная справка об ошибке. При нажатии **Enter** происходит переход к строке кода с ошибкой или к следующей строке.

Подробное описание ошибок имеется в системе помощи, которое выводится командой **Help, Contents, Error Messages**.

В рассматриваемом примере имеется одна ошибка **Statement missing ;**, смысл которой в том, что потеряна точка с запятой в выражении.

Обращаясь к программе видим, что ; отсутствует в 7-й строке, хотя в сообщении об ошибке говорится о 8-й.

Смысл предупреждения состоит в том, что переменной **c** присваивается значение, которое нигде не используется.

Поставим символ ; в 7-й строке, записав ее в виде

```
c = a / b; //Деление чисел
```

и откомпилируем программу. Ошибки и предупреждения исчезнут.

Отсюда понятно, что предупреждения было следствием ошибки.

Ошибки в процессе работы программы

Запустим программу на выполнение, командой **Run, Run**, или нажав **Ctrl+F9**. На экране пользователя, нажав **Alt+F5** или выполнив команду **Window, User screen**, увидим

```
Divide error
```

Это уже сообщение об ошибке времени выполнения, или Run-time error. Из текста сообщения понимаем, что ошибка связана с делением.

Иногда программа аварийно завершается из-за невозможности дальнейших вычислений, например, при делении на нуль, при попытке извлечь корень из отрицательного числа, в некоторых других случаях.

Подобные ошибки обычно несложно найти, так как достаточно посмотреть ограниченное число мест, где критические ситуации возможны. В рассматриваемом примере есть единственная операция деления, которая и привела к ошибке.

Более трудно бывает найти ошибку, когда программа завершается нормально, но результаты неправильные.

Достаточно часто возникают ошибки, приводящие к бесконечному повторению какого-либо цикла. В этом случае говорят, что программа *зациклилась*. Если зациклилась программа, запущенная из интегрированной среды ТС, то часто удается прервать бесконечный цикл, нажав комбинацию клавиш **Ctrl+Break**.

Во всех случаях обнаружить ошибки времени выполнения поможет построчное выполнение или трассировка.

Трассировка программ

Построчное выполнение программы осуществляется путем нажатия клавиши **F7** (команда меню **Run, Trace into**). Каждое нажатие **F7** приводит к выполнению одной строки кода, которая при этом выделяется подсветкой. Выполняя по шагам программу, обнаруживаем, что ее выполнение прерывается при достижении строки

```
c = a / b;
```

При нажатии **F7** происходит «заход» в код функций, если в выполняемой строке есть обращение к функции.

Команда **Run, Step over** или **F8** также осуществляет построчное выполнение программы, но без захода в функции.

По команде **Run, Go to cursor** или **F4** программа выполняется до точки расположения курсора и останавливается. Дальнейшее выполнение программы можно делать построчно, нажимая **F7** или **F8**. Ускорить отладку можно с использованием «Точек останова» (BreakPoint) установить или снять которые можно используя клавиши **Ctrl+F8**.

Выход из режима отладки производится командой **Run, Program reset** или нажатием комбинации клавиш **Ctrl+F2**.

Можно прервать отладку и продолжить дальнейшее выполнение программы, выполнив команду **Run, Run** или **Ctrl+F9**.

Просмотр текущих значений переменных и выражений

Разберемся в причине возникновения ошибки при выполнении деления.

В этом поможет просмотр текущих значений переменных в окне просмотра *Watch*, которое выводится на экран командой **Window, Watch**.

Новые выражения добавляются в окно просмотра командой **Debug, Watches, Add watch** или нажатием **Ctrl+F7**.

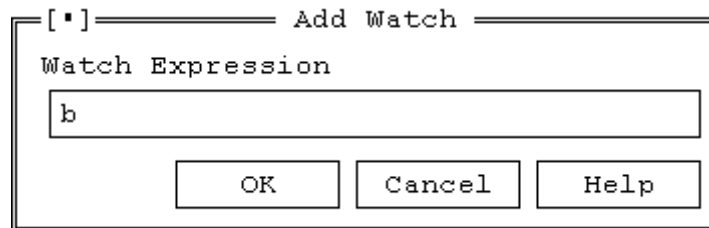


Рис. 9. Добавление выражений в окно просмотра

Нажав четыре раза **F7**, дойдем до строки с ошибкой. Нажмем в этот момент **Ctrl+F7**, появится окно, Рис. 9, в поле которого **Watch Expression**, можно ввести переменную или выражение для просмотра. Введем сначала имя переменной **b**, а затем выражение **2 * 2**. В окне просмотра увидим значение переменной и выражения, Рис. 10

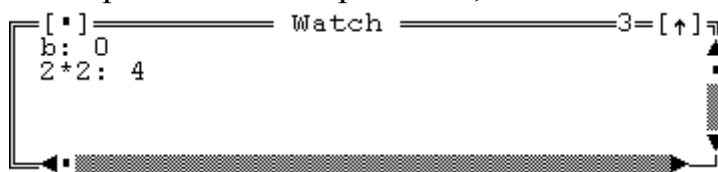


Рис. 10. Окно просмотра значений переменных

Теперь очевидна причина ошибки – деление на нуль.

Пример с выражением **2*2** приведен, чтобы показать, что в окне просмотра могут вычисляться выражения и отображаться их значения.

Индивидуальные задания по теме: «Циклические алгоритмы»

№Вар.	Задание
1.	Вычислить число сочетаний из n по m по формуле: $C = \frac{n!}{m!(n-m)!}; \text{ где } n! = 1*2*3...(n-1)*n,$ целые числа n , m ($n \geq m > 0$) ввести с клавиатуры.
2.	Вычислить значение выражения: $b = (1 + 1/2 + 1/2^2 + 1/2^3 + \dots + 1/2^n)^2$; Значение n ввести с клавиатуры.
3.	Вычислить число размещений из n по m по формуле: $A = n*(n-1)...(n-m+1),$ Целые числа n и m ввести с клавиатуры.
4.	Вычислить приближенное значение бесконечной суммы с точностью до $\epsilon = 0.001$; $S = 1 - n/(2*3) + (n/(3*4))^2 - (n/(4*5))^3 + \dots + (-1)^k (n/((k+1)(k+2)))^{k+1} \dots$ Значение n и точность расчетов ввести с клавиатуры. <i>Примечание: Считать, что требуемая точность достигнута, если очередное слагаемое оказалось по модулю $< \epsilon$</i>
5.	Вычислить приближенное значение бесконечной суммы с точностью до $\epsilon = 0.0005$; $S = 1 - n / (2*3*4)^2 + (n/(3*4*5))^4 - (n/(4*5*6))^6 + \dots + (-1)^k (n / ((k+1) * (k+2) * (k+3)))^{2k+1} \dots$ Значение n и точность расчетов ввести с клавиатуры. <i>Примечание: Считать, что требуемая точность достигнута, если очередное слагаемое оказалось по модулю $< \epsilon$</i>
6.	Вычислить бесконечную сумму $(-1)^i / (i+1)!$ с точностью $\epsilon = 0.0001$ <i>Примечание: Считать, что требуемая точность достигнута, если очередное слагаемое оказалось по модулю $< \epsilon$</i>
7.	Дано натуральное n , определить количество цифр в числе n и сумму всех его цифр. Значение n ввести с клавиатуры.
8.	Найти наибольшее значение функции $y = ax^3 + bx - c$, при изменении x от x_{нач} до x_{кон} с шагом h . Исходные данные: $a = 2,14$; $c = 3,25$; $b = -4,21$; $x_{нач} = -45$; $x_{кон} = -33,5$; $h = -0,5$.

№Вар.	Задание
9.	<p>Составить программу вычисления и вывода на экран таблицы сумм</p> $S = \sum_{k=1}^n \sum_{x=a}^b \frac{\cos(kx)}{k};$ <p>где x изменяется в пределах $a < x \leq b$; с шагом $h = (b-a)/10$. Исходные данные: $a=0.1$; $b=13$; $n=12$.</p>
10.	<p>Числа Фибоначчи f_n определяются формулами:</p> $f_0 = f_1 = 1; f_n = f_{n-1} + f_{n-2} \text{ при } n = 2, 3, \dots$ <p>Напишите программу, выводящую число Фибоначчи с заданным номером $n > 1$.</p>
11.	<p>Числа Фибоначчи f_n определяются формулами:</p> $f_0 = f_1 = 1; f_n = f_{n-1} + f_{n-2} \text{ при } n = 2, 3, \dots$ <p>Напишите программу, печатающую таблицу чисел Фибоначчи, номера которых не превышают заданного значения n.</p>
12.	<p>Числа Фибоначчи f_n определяются формулами:</p> $f_0 = f_1 = 1; f_n = f_{n-1} + f_{n-2} \text{ при } n = 2, 3, \dots$ <p>Напишите программу, выводящую все числа Фибоначчи, значения которых не превышают некоторого числа N.</p>
13.	<p>Числа Фибоначчи f_n определяются формулами:</p> $f_0 = f_1 = 1; f_n = f_{n-1} + f_{n-2} \text{ при } n = 2, 3, \dots$ <p>Напишите программу, выводящую число Фибоначчи с заданным номером $n > 1$.</p>
14.	<p>Числа Фибоначчи f_n определяются формулами:</p> $f_0 = f_1 = 1; f_n = f_{n-1} + f_{n-2} \text{ при } n = 2, 3, \dots$ <p>Вычислить S – сумму всех чисел Фибоначчи, которые не превосходят 1000.</p>
15.	<p>Дано натуральное число $n \leq 255$. Напечатать все числа от 1 до n в десятичной и двоичной системах счисления.</p>
16.	<p>Напишите программу, вводящую положительные числа x_1, x_2, \dots. Признаком окончания ввода является ввод нуля или отрицательного числа. Вычислить $y = x_1 + x_1x_2 + x_1x_2x_3 + \dots + x_1x_2 \dots x_m$, где m – число введенных положительных чисел</p>
17.	<p>Дано вещественное число x и вещественное число $\varepsilon > 0$. Вычислить с точностью ε значение следующей функции:</p> $e^x = 1 + x/1! + x^2/2! + \dots + x^n/n! + \dots$ <p>Вычисления прекратить, когда очередной член суммы $x^n/n!$ станет по модулю меньше ε.</p>

№Вар.	Задание
18.	Вычислите с заданной точностью $\varepsilon > 0$ приближенное значение синуса, используя ряд: $\sin x = x - x^3/3! + x^5/5! - \dots + (-1)^n x^{2n+1}/(2n+1)! + \dots$
19.	Не используя стандартные функции, за исключением <i>abs()</i> , вычислить с заданной точностью <i>eps</i> >0 значение: $y = \arctg x$ $= x - x^3/3 + x^5/5 + \dots + (-1)^n x^{2n+1}/(2n+1) + \dots \quad (x < 1)$
20.	Известно, что $\pi/4 \approx 1 - 1/3 + 1/5 - 1/7 + 1/9 \dots$ Вычислить приближенное значение числа π с заданной точностью.
21.	Дано целое $n > 0$. Напечатать все простые числа из диапазона $[2, n]$. Простым называется число, которое не имеет делителей, кроме 1 и самого себя

Лабораторная работа №3: Массивы и строки.

Цель работы:

Изучение особенностей использования массивов в языке Си, оптимального применения операторов цикла для доступа к элементам массивов.

Общие сведения о массивах

Массив — это структура однотипных элементов, занимающих непрерывную область памяти. С массивом связаны следующие его свойства: имя, тип, размерность, размер.

Формат описания массива следующий:

```
тип элементов имя [константное_выражение]
```

Константное выражение определяет размер массива, т. е. число элементов этого массива. Например, согласно описанию

```
int A[10];
```

объявлен массив с именем A, содержащий 10 элементов целого типа. Элементы массива обозначаются индексированными именами. Нижнее значение индекса равно 0:

```
A[0], A[1], A[2], A[3], A[4], A[5], A[6], A[7],  
A[8], A[9]
```

Размер массива может явно не указываться, если при его объявлении производится инициализация значений элементов. Например:

```
int p[]={2, 4, 6, 10, 1};
```

В этом случае создается массив из пяти элементов со следующими значениями:

```
p[0]=2, p[1]=4, p[2]=6, p[3]=10, p[4]=1
```

Пример 1. Ввод с клавиатуры и вывод на экран одномерного массива.

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

```
#include <conio.h>
void main() {
int i, A[5];
clrscr();
for(i=0;i<5;i++)
{cout<<"A["<i<<"]=";  cin>>A[i];}
for(i=0; i<5; i++)
cout<<"A["<i<<"]="<A[i]<<"  "; }
```

Пример 2. Ввод вещественного массива и вычисление среднего значения.

```
//Среднее значение массива
#include <iostream.h>
#include <conio.h>
void main() {
const n=10;
int i; double A[n], SA;
clrscr() ;
for(i=0; i<n; i++) {cout<<"A["<i<<"]=";
    cin >> A[i];}
SA=0;
for(i=0; i<n;i++) SA=SA+A[i];
SA=SA/n;
cout<<"\Среднее значение="<SA; }
```

В этой программе обратите внимание на определение размера массива через константу.

Пример 3. Сортировка массива «методом пузырька».

```
//Сортировка массива
#include <conio.h>
void main()
{ int X[]={6,4,9,3,2,1,5,7,8,10};
int i,j,n,A;
clrscr () ;
n=sizeof(X)/sizeof(X[0]);
for(i=0; i<n-1; i++)
for(j=0; j<n-1-i; j++)
if(X[j]>X[j+1]) {A=X[j]; X[j]=X[j+1]; X[j+1]=A;}
for(i=0; i<n; i++) cout<<X[i]<<"  "; }
```

Чтобы сделать программу универсальной по отношению к размеру

массива, значение размера вычисляется автоматически и заносится в переменную `n`. Для этого используется операция `sizeof()` — определение размера в байтах. Результат `sizeof (X)` равен размеру в памяти всего массива `X` — 20 байтам. Результат `sizeof (X[0])` равен размеру одного элемента массива — 2 байтам. Отношение этих величин равно 10 — числу элементов массива. Внимательно проанализируйте организацию перебора значений параметров вложенных циклов — `i`, `j`.

В результате выполнения этой программы на экран выведется упорядоченная числовая последовательность

```
1 2 3 4 5 6 7 8 9 10
```

Многомерные массивы.

Двумерный массив трактуется как одномерный массив, элементами которого является массив с указанным в описании типом элементов. Например, оператор

```
float R[5][10];
```

объявляет массив из пяти элементов, каждый из которых есть массив из десяти вещественных чисел. Отдельные величины этого массива обозначаются именами с двумя индексами: `R[0][0]`, `R[0][1]`, ..., `R[4][9]`.

Работа с символьными строками

Символьные строки организуются как массивы символов, последним из которых является символ `\0`, внутренний код которого равен нулю. На длину символьного массива в Си нет ограничения.

Строка описывается как символьный массив. Например:

```
char STR[20] ;
```

Одновременно с описанием строки может инициализироваться. Возможны два способа инициализации строки — с помощью строковой константы и в виде списка символов:

```
char S[10]="строка";  
char S []="строка";  
char S[10]={'с','т','р','о','к','а','\0'};
```

По результату первого описания под строку `S` будет выделено 10 байт памяти, из них первые 7 получают значения при инициализации (седьмой — нулевой символ). Второе описание сформирует строку из семи символов. Третье описание по результату равнозначно первому. Конечно, можно определить символьный массив и так:

```
char S[10]={'с','т','р','о','к','а'};
```

т. е. без нулевого символа в конце. Но это приведет к проблемам с обработкой такой строки, так как будет отсутствовать ориентир на его окончание.

Из вышеприведенных примеров видно, что символьные константы в языке Си задаются при помощи апострофов. (одинарные кавычки), например.

```
char a ='x';    //объявление символьной переменной a с присвоением  
                // начального значения (символ 'x')  
//Вывод на экран алфавита в два столбца в виде символов и кодов  
for (a='a'; a<='z';a++){  
    printf("Символ:%c,\t Код:%d",a,a);  
}
```

Отдельные символы строки идентифицируются индексированными именами. Например, в описанной выше строке `S [0] =' с', S[5]=' а'.`

Обработка строк обычно связана с перебором всех символов от начала до конца. Признаком конца такого перебора является обнаружение нулевого символа. В следующей программе производится последовательная замена всех символов строки на звездочки и подсчет

длины строки.

Пример 1.

```
//Замена символов строки на звездочки
#include <stdio.h>
#include <conio.h>
void main() {
    char s[]="fh5j";
    int i=0;
    clrscr ();
    puts(s);
    while(s[i]){
        s[i++]='*'; puts(s);
    }
    printf("\n,Длина строки=%d",i);
}
```

В результате выполнения программы на экране получим:

```
fh5j
*h5j
**5j
***J
****
Длина строки=4
```

В этой программе цикл повторяет свое выполнение, пока `S[i]` не получит значение нулевого символа.

Для вывода строки на экран в стандартной библиотеке `stdio` имеется функция `puts()`. Аргументом этой функции указывается имя строки. В этой же библиотеке есть функция ввода строки с клавиатуры с именем `gets()`. В качестве аргумента указывается имя строки, в которую производится ввод.

Среди стандартных библиотек Си/Си++ существует библиотека функций для обработки строк. Ее заголовочный файл — `string.h`. В следующем примере используется функция определения длины строки из этой библиотеки. Имя функции — `strlen()`. В качестве аргумента указывается имя строки.

Пример 2. Ввести символьную строку. Перевернуть (обратить) эту строку. Например, если ввели строку «abcdef», то в результате в ней должны получить «fedcba».

```
//Обращение строки
```

```

#include <stdio.h>
#include <string.h>
#include <conio.h>
void main() {
char C,S[10]; int i;
clrscr ();
printf("Введите строку: ");
gets(S) ;
for(i=0; i<=(strlen(S)-1)/2;i++)
{C=S[i]; S[i]=S[strlen(S)-i-1]; S[strlen(S)-i-1]=C;}
printf("\nПеревернутая строка:");
puts (S);
}

```

Идея алгоритма состоит в перестановке символов, расположенных на одинаковом расстоянии от начала и конца строки. Перебор элементов строки доходит до ее середины. Составляя подобные программы, не надо забывать, что индекс первого символа строки — 0, а индекс последнего на единицу меньше длины строки.

Строка как параметр функции. Использование строк в качестве параметра функции аналогично рассмотренному выше использованию массивов других типов. Необходимо помнить, что имя массива есть указатель на его начало. Однако для строк имеется одно существенное отличие от массивов других типов: *имя строки является указателем-переменной*, и, следовательно, его значение может подвергаться изменению. Стандарт Си рекомендует в заголовках функций, работающих со строками, явно использовать обозначение символьного указателя.

Пример 1. Запишем определение функции вычисления длины строки (аналог стандартной функции `strlen()`).

```

int length(char *s)
{ int k;
  for(k=0; *s++!='\0'; k++);
  return k;
}

```

Здесь функция использует явный механизм работы с указателем. Изменение значения указателя `s` допустимо благодаря тому, что он является переменной. Еще раз напомним, что для числовых массивов этого делать нельзя! Если соблюдать данное ограничение и для строк, то условное выражение в операторе `for` следовало бы писать так:

* (s+k) != '\0' или s[k] != '\0' .

Пример 2. Оформим программу обращения строки в виде функции и напомним основную программу, использующую ее. Алгоритм обращения реализуем иначе, чем в рассмотренной выше программе. Для определения длины строки не будем пользоваться стандартной функцией. Для вывода строки на экран применим функцию `printf()` со спецификатором `%s` (работает аналогично функции `puts()`).

```
//Обращение строки
#include <stdio.h>
#include <conio.h>
//Прототипы функций
int length(char * str);
void invers(char * str);
// Основная программа
void main () {
    char S[]="123456789";
    clrscr ();
    invers(S);    //Вызов функции обращения строки
    printf("\n%s",S);
}
//Функция вычисления длины строки
int length(char *s){
    int k;
    for(k=0; *s++!='\0'; k++) ;
    return k;
}
//Функция обращения строки
void invers(char *e){
    char c;
    int i,j,m;
    m=length(e); //Вызов функции length
    for(i=0, j=m-1; i<j; i++, j--){
        c=e[i]; e[i]=e[j]; e[j]=c;
    }
}
```

В результате выполнения этой программы на экране получим строку:

9 8 7 6 5 4 3 2 1 0

Пример 3. Описать функцию вставки символа в строку. Параметры

функции: строка, позиция вставки, вставляемый символ. Использовать эту функцию в основной программе, где исходная строка, номер позиции и вставляемый символ задаются вводом.

```
// Вставка символа в строку
#include <stdio.h>
#include <string.h>
#include <conio.h>
void INSERT(char *str, int p, char c)
{ int i;
  for(i=strlen(str); i>=p; i--)
    str[i+1]=str[i];
  str[p]=c; }

void main() {
  char c, S[100]; int n;
  clrscr();
  puts("Введите строку:"); gets(S);
  puts("Введите позицию вставки:"); scanf("%d", &n);
  puts("Введите символ:"); c=getche();
  INSERT(S,n,c);
  puts("\nРезультат:"); puts(S);
}
```

Вот вариант диалога на экране при выполнении

Введите строку:

0123456789

Введите позицию вставки:

4

Введите символ: *

Результат: 0123*456789

В этой программе наряду с рассмотренными ранее функциями для ввода и вывода строки `gets()` и `puts()` используется функция чтения символа с клавиатуры **`getche()`** из библиотеки `stdio.h`. Ее прототип имеет вид: `int getche (void)`. Она возвращает значение символа, введенного с клавиатуры, которое может быть присвоено символьной переменной.

Часто бывает нужно производить изменение содержимого строк, анализируя при этом коды символов. Для определения кодов можно пользоваться оператором `printf` следующего вида:

`printf("%d", 'a'); //вывод кода символа`
или стандартной программой из состава операционной системы
«Пуск→Стандартные→Служебные→Таблица символов.»



Рис. 11. Таблица символов.

При этом необходимо выбрать шрифт с названием «terminal» для выбора кодировки DOS.

Индивидуальные задания по теме: Строки

1. Дана строка, заканчивающаяся точкой. Подсчитать, сколько слов в строке.
2. Дана строка, содержащая английский текст. Найти количество слов, начинающихся с буквы b.
3. Дана строка. Подсчитать, сколько в ней букв r, k, t.
4. Дана строка. Определить, сколько в ней символов *, ;, :.
5. Дана строка, содержащая текст. Найти длину самого короткого слова и самого длинного слова.
6. Дана строка символов, среди которых есть двоеточие (:). Определить, сколько символов ему предшествует.
7. Дана строка, содержащая текст, заканчивающийся точкой. Вывести на экран слова, содержащие три буквы.
8. Дана строка. Преобразовать ее, удалив каждый символ * и повторив каждый символ, отличный от *.
9. Дана строка. Определить, сколько раз входит в нее группа букв abc.
10. Дана строка. Подсчитать количество букв k в последнем ее слове.
11. Дана строка. Подсчитать, сколько различных символов встречается в ней. Вывести их на экран.
12. Дана строка. Подсчитать самую длинную последовательность подряд идущих букв a.
13. Дана строка символов, среди которых есть одна открывающаяся и одна закрывающаяся скобка. Вывести на экран все символы, расположенные внутри этих скобок.
14. Имеется строка, содержащая буквы латинского алфавита и цифры. Вывести на экран длину наибольшей последовательности цифр, идущих подряд.
15. Дан набор слов, разделенных точкой с запятой (;). Набор заканчивается двоеточием (:). Определить, сколько в нем слов, заканчивающихся буквой a.
16. Дана строка. Указать те слова, которые содержат хотя бы одну букву k.
17. Дана строка. Найти в ней те слова, которые начинаются и оканчиваются одной и той же буквой.
18. В строке заменить все двоеточия (:) точкой с запятой (;). Подсчитать количество замен.
19. В строке удалить символ «двоеточие» (:) и подсчитать количество удаленных символов.
20. В строке между словами вставить вместо пробела запятую и пробел.

21. Удалить часть символьной строки, заключенной в скобки (вместе со скобками).
22. Определить, сколько раз в строке встречается заданное слово.
23. В строке имеется одна точка с запятой (;). Подсчитать количество символов до точки с запятой и после нее.
24. Дана строка длиной n символов. Преобразовать ее, заменив точками все двоеточия (:), встречающиеся среди первых $n/2$ символов, и заменив точками все восклицательные знаки, встречающиеся среди символов, стоящих после $n/2$ символов.
25. Строка содержит одно слово. Проверить, будет ли оно читаться одинаково справа налево и слева направо (т.е. является ли оно палиндромом).
26. В записке слова зашифрованы — каждое из них записано наоборот. Расшифровать сообщение.
27. Проверить, одинаковое ли число открывающихся и закрывающихся скобок в данной строке.
28. Строка, содержащая произвольный русский текст, состоит не более чем из 200 символов. Написать, какие буквы и сколько раз встречаются в этом тексте. Ответ должен приводиться в грамматически правильной форме, например а — 25 раз, к — 3 раза и т.д.
29. Упорядочить данный массив английских слов по алфавиту.
30. Даны две строки А и В. Составьте программу, проверяющую, можно ли из букв, входящих в А, составить В (буквы можно использовать не более одного раза и можно переставлять).
Например, А: ИНТЕГРАЛ; В: АГЕНТ — составить можно; В: ГРАФ — составить нельзя.
31. Строка содержит произвольный русский текст. Проверить, каких букв в нем больше: гласных или согласных.
32. Двумерный массив $m \times n$ содержит некоторые буквы русского алфавита, расположенные в произвольном порядке. Написать программу, проверяющую, можно ли из этих букв составить данное слово S. Каждая буква массива используется не более одного раза.
33. Результаты вступительных экзаменов представлены в виде списка из N строк, в каждой строке которого записаны фамилия студента и отметки по каждому из M экзаменов. Определить количество абитуриентов, сдавших вступительные экзамены только на «отлично».
34. Составить программу преобразования натуральных чисел, записанных в римской нумерации, в десятичную систему счисления.
35. Из заданной символьной строки выбрать те символы, которые встречаются в ней только один раз, в том порядке, в котором они встречаются в тексте.

36. В символьном массиве хранятся фамилии и инициалы учеников класса. Требуется напечатать список класса с указанием для каждого ученика количества его однофамильцев.
37. Дано число в двоичной системе счисления. Проверить правильность ввода этого числа (в его записи должны быть только символы 0 и 1). Если число введено неверно, повторить ввод. При правильном вводе перевести число в десятичную систему счисления.
38. В заданной строке удалить все лишние пробелы.
39. Для заданного текста определить длину содержащейся в нем максимальной серии символов, отличных от букв.
40. Расстояние между двумя словами равной длины — это количество позиций, в которых различаются эти слова. В заданном предложении найти пару слов заданной длины с максимальным расстоянием.
41. Отредактировать заданное предложение, удаляя из него те слова, которые встречаются в предложении заданное число раз.
42. Напечатать те слова, которые встречаются в каждом из двух заданных предложений.
43. Отредактировать заданное предложение, удаляя из него все слова с нечетными номерами и переворачивая слова с четными номерами.
44. Шифрация. Один из методов шифрации называется наложением гаммы. Делается это следующим образом: берется некоторое случайное число в диапазоне от 127 до 255 — гамма, и код каждого символа строки заменяется кодом, получающимся в результате операции: $\text{новый код} = \text{старый код} \oplus \text{гамма}$.
- Написать программу, реализующую:
- а) данный метод шифрации;
 - б) дешифрацию строки при заданной гамме. Входные данные: шифруемая строка. Выходные данные:
- гамма;
 - зашифрованная строка.

Индивидуальные задания по теме: Массивы

1. Задана вещественная матрица A размерности $m \times n$. Получить матрицу $B = A^n$. Степень n матрицы B и размерность уточнить у преподавателя.
2. Написать (под)программу, которая выводит на экран элементы одномерного массива в порядке убывания их значений. Сформировать матрицу, которая содержит все отрицательные элементы исходной последовательности.
3. Заданы два двумерных массива вещественных чисел A и B размерности n . Написать (под)программу сравнения двух массивов. Вывести на экран соответствующее сообщение.
4. Даны натуральное n и квадратная вещественная матрица 5-го порядка. Вычислить n -ю степень этой матрицы ($A^1 = A$, $A^2 = A \cdot A$, $A^3 = A^2 \cdot A$, и т.д.)
5. Дана двумерная квадратная матрица вещественных чисел A размерности n и одномерный массив X той же размерности. Написать подпрограмму для решения следующей задачи:
 - a.) нечетные строки матрицы A заменить на X
 - b.) четные столбцы матрицы A заменить на X
 - c.) первые 6 строк заменить на X
 - d.) поменять местами 1-ю и 2-ю, 3-ю и 4-ю... строки матрицы A
6. Даны описания:

```
typedef float point[2];  
point M [7];
```

Рассматривая элементы массива M как координаты точек на плоскости, найти наибольшее расстояние между этими точками.
7. Сформировать квадратную целочисленную матрицу 10-го порядка и заполнить следующими способами:
 - a.) Заполнить главную диагональ числами от 0 до 9;
 - b.) построчно заполнить матрицу числами от 1 до 100;
 - c.) заполнить область правее главной диагонали возрастающими числами 1..10, 1..9, 1..8, ..1.
8. Задан двумерный массив A размерности $n \times m$. Получить массив B из массива A удалением одной строки и одного столбца.
9. Определить количество различных элементов произвольного двумерного массива. (т.е. повторяющиеся элементы считать один раз)
10. Определить количество “особых” элементов массива $C[1..n, 1..m]$, считая элемент особым если:
 - a.) он больше суммы остальных элементов своего столбца
 - b.) в его строке слева от него находятся элементы меньше его, а справа - большие.

11. Дана вещественная матрица размерности 4×5 . Упорядочить ее строки по неубыванию:
 - a.) их первых элементов
 - b.) суммы их элементов
 - c.) их наибольших элементов
12. Определить, является ли заданная целая квадратная матрица 5-го порядка симметричной относительно главной диагонали.
13. Элемент матрицы назовем седловой точкой, если он является наименьшим в своей строке и одновременно наибольшим в своем столбце или наоборот. Для заданной матрицы размером 7×7 найти индексы всех седловых точек.
14. Дана вещественная матрица размером 7×7 , все элементы которой различны. Найти скалярное произведение строки в которой находится наибольший элемент матрицы на столбец с наименьшим элементом.
15. Определить, является ли заданная целая квадратная матрица 5-го порядка магическим квадратом, т.е. такой, в которой суммы элементов во всех строках и столбцах одинаковы.
16. Преобразовать массив S размерности 7×7 , осуществив поворот элементов вокруг его центра на 90° против часовой стрелки.
17. В двумерном массиве (матрице) 4×4 переставьте местами строки с заданными номерами i и k
18. В двумерном массиве (матрице) 4×4 поменяйте местами первую строку и строку, у которой первый элемент является максимальным в первом столбце
19. Напишите программу, меняющую местами столбцы j и k матрицы размером 4×4 .
20. Написать программу для поиска минимального элемента в двумерном целочисленном массиве. (массив инициализировать в тексте программы)
21. Написать программу для вычисления среднего арифметического ненулевых элементов в двумерном целочисленном массиве. (массив инициализировать в тексте программы)
22. Написать программу, проверяющую, находится ли в двумерном целочисленном массиве введенное с клавиатуры число. (массив инициализировать в тексте программы)
23. Написать программу, проверяющую, образуют ли элементы двумерного целочисленного массива неубывающую последовательность. (массив инициализировать в тексте программы)
24. Написать программу, которая вычисляет, сколько раз введенное с клавиатуры число встречается в двумерном целочисленном массиве. (массив инициализировать в тексте программы)

25. Написать программу, проверяющую, есть ли в двумерном целочисленном массиве одинаковые элементы. (массив инициализировать в тексте программы)
26. Написать программу, вычисляющую сумму строк двумерного целочисленного массива. (массив инициализировать в тексте программы)
27. Заданы два двумерных массива вещественных чисел A и B размерности n . Вычислить:
- a.) $C = A + B$;
 - b.) $y = Ax$;
 - c.) $C = AB$;
 - d.) $B = B^T$; (транспонировать)

Лабораторная работа №4: «Функции»

Цель работы:

Изучение структурного подхода к созданию программ. Применение метода декомпозиции для реализации алгоритма решения поставленной задачи. Получение навыков работы с функциями. Изучение способов передачи параметров в подпрограммы.

Определение функции.

*Функция является основной программной единицей в Си, минимальным исполняемым программным модулем. Всякая программа обязательно включает в себя основную функцию с именем **main**. Если в программе используются и другие функции, то они выполняют роль подпрограмм.*

Формат определения функции:

тип имя_функции (спецификация_параметров) {тело_функции}

Тип — это тип возвращаемого функцией результата. Если функция не возвращает никакого результата, то для нее указывается тип **void**.

Имя — идентификатор, задаваемый программистом или **main** для основной функции.

Спецификации параметров — это либо «пусто», либо список имен формальных параметров функции с указанием типа для каждого из них.

Тело функции — это либо составной оператор, либо *блок*.

Оператором возврата из функции в точку ее вызова является оператор *return*. Он может использоваться в функциях в двух формах:

return;
или
return выражение;

В первом случае функция не возвращает никакого значения в качестве своего результата. Во втором случае результатом функции является значение указанного выражения. Тип этого выражения должен либо совпадать с типом функции, либо относиться к числу типов, допускающих автоматическое преобразование к типу функции.

Вызов функции

Формат вызова следующий:

имя_функции(список_фактических_параметров)

Между формальными и фактическими параметрами при вызове функции должны соблюдаться правила соответствия *по последовательности* и *по типам*. Фактический параметр — это выражение того же типа, что и у соответствующего ему формального параметра. *Функция не может изменить значения переменных, указанных в качестве фактических параметров.*

Рассмотрим пример. Требуется составить программу нахождения наибольшего значения из трех величин — $\max(a, b, c)$. Для ее решения можно использовать вспомогательный алгоритм нахождения максимального значения из двух, поскольку справедливо равенство: $\max(a, b, c) = \max(\max(a, b), c)$.

Вот программа решения этой задачи с использованием вспомогательной функции.

Пример 1.

```
#include <stdio.h>
//Определение вспомогательной функции
int MAX(int x, int y) {
    if (x>y) return x;
    else return y;
}
//Основная функция
void main() {
    int a,b,c,d;
    printf("Введите a,b,c:");
    scanf("%d%d%d", &a, &b, &c);
    d=MAX(MAX(a,b), c);
    printf("\nmax(a,b,c)=%d", d);
}
```

Прототипы функций.

Прототипом называется предварительное описание функции, в котором содержатся все необходимые сведения для правильного обращения к ней: имя и тип функции, типы формальных параметров. Прототипы используются в случае, если определение функции производится в другом файле (например в стандартной библиотеке), или если нужно вызвать функцию до ее определения в тексте программы. Очка с запятой в конце прототипа ставится обязательно!

Вот другой вариант программы, решающей ту же самую задачу.

Пример 2.

```
#include <stdio.h>
//Прототип функции MAX()
int MAX(int , int);
//Основная функция
void main() {
    int a,b,c,d;
    printf("Введите a,b,c:");
    scanf("%d%d%d",&a,&b,&c);
    d=MAX(MAX(a,b),c);
    printf("\nmax(a,b,c)=%d",d);
}
//Определение вспомогательной функции
int MAX(int x, int y) {
    if (x>y) return x;
    else return y;
}
```

Здесь использован *прототип* функции. Можно было бы записать прототип и в теле основной функции наряду с описаниями других программных объектов в ней. В прототипе имена формальных параметров указывать необязательно (как это сделано в примере 2), хотя их указание не является ошибочным. Можно было написать и так, как в заголовке определения функции:

```
int MAX(int x, int y);
```

Следующий пример: программа для вычисления наибольшего общего делителя для суммы, разности и произведения двух чисел:

Пример 4.

```
#include <iostream.h>
#include <math.h>
int NOD2(int,int) ;
void main() {
    int a,b,Rez;
    cout<<"a="; cin>>a;
    cout<<"b="; cin>>b;
    Rez=NOD2(NOD2(a+b, abs(a-b)),a*b);
    cout<<"NOD is: "<<Rez;
```

```

}
int NOD2(int M, int N) {
    while(M!=N) {
        if(M>N) M=M-N; else N=N-M;
    }
    return M;
}

```

Использование библиотечных функций.

Библиотечными называются вспомогательные функции, хранящиеся в отдельных файлах. Стандартные библиотеки входят в стандартный комплект системы программирования на Си/Си++. Кроме того, программист может создавать собственные библиотеки функций. Ранее уже говорилось о том, что для использования стандартных функций необходимо подключать к программе заголовочные файлы соответствующих библиотек. Делается это с помощью директивы претранслятора `#include` с указанием имени заголовочного файла. Например, `#include <stdio.h>`. Все заголовочные файлы имеют расширение `h` (от английского *header*). Теперь должно быть понятно, что эти файлы содержат прототипы функций библиотеки. На стадии претрансляции происходит подстановка прототипов перед основной функцией, после чего компилятор в состоянии контролировать правильность обращения к функциям. ибке.

Рекурсивные определения функций.

Проиллюстрируем определение рекурсивной функции на классическом примере вычисления факториала целого положительного числа.

```

long Factor(int n)
{ if (n<0) return 0;
  if (n==0) return 1;
  return n*Factor(n-1);
}

```

В случае если при вызове функции будет задан отрицательный аргумент, она вернет нулевое значение — признак неверного обращения. Выполнение программы происходит аналогично тому, как это описано в разд. 3.13.

Передача значений через глобальные переменные.

Областью действия описания программного объекта называется часть программы, в пределах которой действует (учитывается) это описание. Если переменная описана внутри некоторого блока, то она локализована в этом блоке и из других блоков, внешних по отношению к данному, «не видна». Если описание переменной находится вне блока и предшествует ему в тексте программы, то это описание действует внутри блока и называется **глобальным**.

Индивидуальные задания по теме: «Функции.»

1. Написать подпрограмму, которая выводит на печать элементы одномерного массива в порядке возрастания их значений. В головной программе вызвать эту подпрограмму для нескольких массивов.
2. Написать подпрограмму для вычисления суммы:
$$\sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$
с точностью 0.001, где X передать в качестве параметра.
3. Написать подпрограмму для вычисления суммы:
$$\sum_{k=0}^{\infty} \frac{(-1)^{k+1}}{(2k)!} \left(\frac{x}{3}\right)^{4k}$$
с точностью 0.001, где X передать в качестве параметра.
4. Даны натуральное n и целочисленная матрица $m \times n$. Получить b_1, b_2, \dots, b_n , где b_i - наименьшее из значений элементов $a_{i1}, a_{i2}, \dots, a_{in}$, $i=1, 2, \dots, n$. Для решения поставленной задачи создать функцию с двумя параметрами (исходный и результирующий массив).
5. Создать функцию, которая из произвольной строки, содержащей некоторый текст, выделяет слова и печатает их в алфавитном порядке (по первой букве). Исходную строку ввести в головной программе
6. Задана матрица A размером m на n . Получить матрицу $B=A^{15}$
7. Создать подпрограмму, для нахождения максимального элемента в одномерном массиве и замены всех последующих элементов на значение 0.
8. Даны описания:

```
typedef float point[2];  
point M [7];
```

Рассматривая элементы массива M как координаты точек на плоскости, найти наибольшее расстояние между этими точками. Создать подпрограмму для определения расстояния между двумя точками.
9. Написать подпрограмму, для поиска максимума в двумерном массиве. В головной программе вызвать эту подпрограмму для нескольких массивов.

10. Вывести на экран все числа в диапазоне от A до B , затем повторить задание, изменив порядок цифр в каждом числе на обратный. (Написать подпрограмму, для изменения порядка цифр в числе.)
11. Заменить в произвольной строке все строчные буквы на прописные и наоборот. (Написать подпрограмму, для изменения регистра букв.)
12. Создать функцию, которая в произвольной строке находит наиболее часто повторяющийся символ. В головной программе вызвать эту подпрограмму для различных строк.
13. Создать функцию, которая из произвольной строки, содержащей некоторый текст, удаляет все повторно (подряд) встречающиеся символы.
14. Создать функцию, которая печатает все натуральные числа, меньшие N , являющиеся полиндромом. Число называется полиндромом, если оно читается одинаково как сначала, так и с конца (например 383, 22). Число N передать подпрограмме, как параметр.
15. Создать функцию, которая печатает все числа Фибоначчи, меньшие N . Число N передать подпрограмме, как параметр.
16. Создать функцию, которая печатает все числа Фибоначчи, номер которых меньше N . Число N передать подпрограмме, как параметр.
17. Число, в записи которого n цифр, называется числом Армстронга, если сумма его цифр, возведенная в степень n , равна самому числу. Найти все числа Армстронга от 1 до k .
18. Найти все натуральные n -значные числа, цифры в которых образуют строго возрастающую последовательность (например, 1234, 5789).
19. Написать программу, которая находит и выводит на печать все четырехзначные числа вида $abcd$, для которых выполняется условие: a, b, c, d — разные цифры;
20. Заменить отрицательные элементы линейного массива их модулями, не пользуясь стандартной функцией вычисления модуля. Подсчитать количество произведенных замен.

Лабораторная работа №5: Работа с файлами.

Цель работы:

Изучение правил работы с файлами и способов доступа к информации хранящейся в файлах. Обучение способам создания, модификации и копирования файлов на примере работы с потоками языка Си.

Общие сведения:

При работе с файлами с использованием языка «СИ» можно выделить 4 основных действия:

1.) Вначале необходимо **объявить указатель на поток (файл)**. Формат такого объявления:

FILE *имя указателя;

Например:

FILE *fp;

Слово **FILE** является стандартным именем структурного типа, объявленного в заголовочном файле «stdio.h». В структуре **FILE** содержится информация, с помощью которой ведется работа с потоком, в частности: указатель на буфер, указатель (индикатор) текущей позиции в потоке и т.д.

2.) Следующий шаг — **открытие потока**, которое производится с помощью стандартной функции **fopen ()**. Эта функция возвращает конкретное значение для указателя на поток и поэтому ее значение присваивается объявленному ранее указателю. Соответствующий оператор имеет формат:

имя_указателя= fopen(имя файла, режим_открытия);

Параметры функции **fopen()** являются строками, которые могут быть как константами, так и указателями на символьные массивы.

Например:

fp=fopen ("test.dat" , "r") ;

Здесь «test. dat» — это имя физического файла в текущем каталоге диска, с которым теперь будет связан поток с указателем **fp**. Параметр режима «**r**» означает, что файл открыт для чтения. Что касается терминологии, то допустимо употреблять как выражение «открытие потока», так и выражение «открытие файла».

Существуют следующие режимы открытия потока и соответствующие им параметры:

<i>Параметр</i>	<i>Режим</i>
r	Открыть для чтения данных
w	Создать для записи
a	Открыть для добавления данных

Открытие уже существующего файла для записи ведет к потере прежней информации в нем. Если такой файл еще не существовал, то он создается. Открывать для чтения можно только существующий файл. Поток может быть открыт либо для текстового, либо для двоичного (бинарного) режима обмена.

Текстовый файл: это последовательность символов, которая делится на строки специальными кодами — возврат каретки (код 13) и перевод строки (код 10). Если файл открыт в текстовом режиме, то при чтении из такого файла комбинация символов «возврат каретки — перевод строки» преобразуется в один символ `\n` — переход к новой строке. При записи в файл осуществляется обратное преобразование. При работе с **двоичным файлом** никаких преобразований символов не происходит, т.е. информация переносится без всяких изменений. Указанные выше параметры режимов открывают текстовые файлы. Если требуется указать на двоичный файл, то к параметру добавляется буква **b**. Например: `rb`, или `wb`, или `r+b`.

Если при открытии потока по какой-либо причине возникла ошибка, то функция **fopen()** возвращает значение константы `NULL`. Эта константа также определена в файле `<stdio.h>`. Ошибка может возникнуть из-за отсутствия открываемого файла на диске, нехватки места в памяти и т.д. Поэтому желательно контролировать правильность прохождения процедуры открытия файла. Рекомендуется следующий способ открытия:

```
FILE *fp;
if (fp=fopen("test.dat", "r")==NULL)
{puts("Не могу открыть файл\n");
return;
}
```

3. Запись и чтение символов. Запись символов в поток производится функцией `putc()` с прототипом:

```
int putc(int ch, FILE *fptr);
```

Если операция прошла успешно, то возвращается записанный символ. В случае ошибки возвращается константа **EOF**.

Считывание символа из потока, открытого для чтения, производится функцией `getc()` с прототипом:

```
int getc(FILE *fptr);
```

Функция возвращает значение считываемого из файла символа. Если достигнут конец файла, то возвращается значение EOF. Заметим, что это происходит лишь в результате чтения кода EOF.

Исторически сложилось так, что `getc()` возвращает значение типа `int`. То же можно сказать и про аргумент `ch` в описании функции `puts()`.

Используется же в обоих случаях только младший байт. Поэтому обмен при обращении может происходить и с переменными типа `char`.

4. Закрытие потока (файла)

осуществляет функция **`fclose()`**, прототип которой имеет вид:

```
int fclose(FILE *fptr);
```

Здесь **`fptr`** обозначает формальное имя указателя на закрываемый поток. Функция возвращает ноль, если операция закрытия прошла успешно. Другая величина означает ошибку.

Пример 1. Составим программу записи в файл последовательности символов, вводимых с клавиатуры. Признаком завершения ввода будет символ `*`.

```
//Запись символов в файл
#include <stdio.h>
#include <conio.h>
void main() {
FILE *fp; char c;
    if((fp=fopen("test.dat", "w"))==NULL) {
        puts("Не могу открыть файл!\n"); return;
    }
    puts ("Вводите символы. Приznak конца - *");
    while((c=getchar()) != '*')
        putc(c, fp);
    fclose(fp);
}
```

В результате на диске (в каталоге, определяемом системой) будет создан файл с именем `test.dat`, который заполнится вводимыми символами. Символ `*` в файл не запишется.

Пример 2. Файл, созданный в результате работы предыдущей программы, требуется последовательно (посимвольно) прочитать и содержимое вывести на экран.

```
//Чтение символов из файла
#include <stdio.h>
```

```
#include <conio.h>
void main() {
FILE *fp; char c;
    clrscr();
    if((fp=fopen("test.dat", "r"))==NULL) {
        puts("Не удалось открыть файл!\n"); return;}
    while((c=getc(fp))!=EOF) putchar (c);
    fclose (fp);
}
```

Связь между результатом работы предыдущей программы и данной программой осуществляется через имя физического файла, которое в обоих случаях должно быть одним и тем же.

Другой способ позволяет прочитать содержимое файла **построчно**:

```
#include <stdio.h>
#include <conio.h>
void main() {
    FILE *fp;
    char S[30];
    clrscr();
    fp=fopen("test.dat", "r") ;
    while(fgets(S, 30, fp) !=NULL) {
        fputs(S, stdout);
    }
    fclose(fp);
}
```

Функция **fgets()** считывает из потока **fp** в строку **S** не более 30 символов. Ввод строки заканчивается при достижении конца строки или заданного количества символов (в нашем случае - 30). При достижении конца файла функция **fgets()** возвращает NULL.

Функция **fputs()** выводит строку в поток (в нашем примере на экран)

Форматированный ввод -вывод.

С помощью функций форматного ввода-вывода можно формировать на диске (или считывать с диска) текстовый файл с результатами вычислений, представленными в символьном виде аналогично выводу на экран. В дальнейшем этот файл может быть просмотрен на экране, распечатан на принтере, отредактирован с помощью текстового редактора. Общий вид функции **форматного вывода**:

```
int fprintf(указатель_на_поток, форматная_строка,
            список_переменных);
```

Используемая нами ранее функция `printf()` для организации вывода на экран является частным вариантом функции **`fprintf()`**. Функция `printf()` работает лишь со стандартным потоком `stdin`, который всегда связывается системой с дисплеем. Не будет ошибкой, если в программе вместо `printf()` написать `fprintf (stdin,...)`.

Правила использования спецификаторов форматов при записи в файлы на диске точно такие же, как и при выводе на экран.

Пример 3.

Составим программу, по которой будет рассчитана и записана в файл таблица квадратов для целых чисел от 1 до 10. Для контроля эта же таблица выводится на экран.

```
//Таблица квадратов
#include <stdio.h>
#include <iostream.h>
#include <math.h>
void main () {
FILE *fp; int x;
fp=fopen("test.dat", "w");
//Вывод на экран и в файл шапки таблицы
printf("\t Таблица квадратов\n");
fprintf(fp, "\t Таблица квадратов\n");
printf("\t x\t\t square(x)\n");
fprintf(fp, "\t x\t\t square(x)\n");
//Вычисление и вывод таблицы квадратов на экран и в
файл
for(x=1; x<=10; x++){
    printf("\t%f\t%f\n", float(x), pow(float(x), 2));
    fprintf(fp, "\t%f\t%f\n", float(x),
pow(float(x), 2));
}
fclose(fp);
}
```

Если теперь с помощью текстового редактора (например, входящего в систему программирования) открыть файл «test.dat», то на экране увидим:

Таблица квадратов	
x	square(x)
1.000000	1.000000
2.000000	4.000000
3.000000	9.000000
4.000000	16.000000

5.000000	25.000000
6.000000	36.000000
7.000000	49.000000
8.000000	64.000000
9.000000	81.000000
10.000000	100.000000

Форматный ввод из текстового файла осуществляется с помощью функции **fscanf()**, общий формат которой выглядит следующим образом:

int fscanf(указатель на поток, форматная_строка, список адресов переменных);

Данной функцией удобно пользоваться в тех случаях, когда исходные данные заранее подготавливаются в текстовом файле.

В следующем примере числовые данные из файла test.dat, полученного в результате выполнения предыдущей программы, вводятся в числовые массивы X и Y. Для контроля значения элементов массивов выводятся на экран. Предварительно с помощью текстового редактора в файле test.dat удаляются две первые строки с заголовками. В результате в файле останутся только числа.

Пример 4.

```
//Ввод чисел из файла
#include <stdio.h>
#include <iostream.h>
#include <math.h>
void main() {
    FILE *fp; int i ;
    float x[10], y[10];
    fp=fopen("test.dat", "r") ;
    i = 0;
    while (feof(fp)==0) {
        fscanf(fp, "%f%f", &x[i], &y[i]) ;
        printf("%f      %f\n", x[i], y[i]);
        i++;
    }
    fclose(fp);
}
```

Функция **fscanf()** удобна также для чтения текстовых файлов по словам. (если разделителем слов является пробел).

Варианты индивидуальных заданий по теме: «Файлы»

Замечание:

исходные текстовые файлы создаются в редакторе языка Си и сохраняются на диске с расширением “ТХТ”. При создании программы обязательным условием является использование собственных функций.

1. Дан файл, содержащий текст. Получить в другом файле тот же текст, записанный заглавными буквами.
2. Дан файл, содержащий произвольный текст. Выяснить, чего в нем больше: букв или цифр.
3. Дан файл, содержащий текст. Выяснить, входит ли данное слово в указанный текст, и если да, то сколько раз.
4. Дан файл, содержащий текст. В предложениях некоторые из слов записаны подряд несколько раз (предложение заканчивается точкой или восклицательным знаком). Получить в новом файле отредактированный текст, в котором удалены повторные вхождения слов в предложение.
5. Дан файл, содержащий текст. Провести частотный анализ текста, т. е. указать (в процентах), сколько раз встречается та или иная буква.
6. Дан текстовый файл. Определить, сколько раз встречается в нем самое длинное слово.
7. Дан файл, содержащий произвольный текст. Проверить, правильно ли в нем расставлены круглые скобки (т.е. находится ли правее каждой открывающейся скобки закрывающаяся и левее закрывающейся — открывающаяся).
8. Дан файл, содержащий текст. Составить в алфавитном порядке список всех слов, встречающихся в этом тексте.
9. Дан файл, содержащий текст. Определить, сколько раз встречается в нем самое короткое слово.
10. Дан файл, содержащий текст и два произвольных слова. Определить, сколько раз они встречаются в тексте и сколько из них — непосредственно друг за другом.
11. Дан файл, содержащий текст. Выбрать из него те символы, которые встречаются в нем только один раз.
12. Дан файл, содержащий текст и арифметические выражения вида **аОшибка! Объект не может быть создан из кодов полей редактирования.b**, где **Ошибка! Объект не может быть создан из кодов полей редактирования.** — один из знаков +, -, *, /. Вывести все арифметические выражения на экран и вычислить их значения.
13. Даны файл, содержащий текст, и некоторые буквы. Найти слово, содержащее наибольшее количество указанных букв.

14. Даны файл, содержащий текст, и некоторая буква. Подсчитать, сколько слов начинается с указанной буквы.
15. Дан файл, содержащий текст. Найти слово, встречающееся в каждом предложении, или сообщить, что такого слова нет.
16. Дан файл, содержащий текст. Определить сколько слов в тексте? Сколько цифр в тексте?
17. Дан файл, содержащий текст. Получить новый файл, заменив в исходном все заглавные буквы строчными и наоборот.
18. Дан файл, содержащий зашифрованный текст. Каждая буква заменяется на следующую за ней (буква z заменяется на а). Получить в новом файле расшифровку данного текста.
19. Дан текстовый файл. Удалить из него все лишние пробелы, оставив между словами не более одного пробела. Результат поместить в новый файл.
20. Даны текстовый файл и некоторое слово. Напечатать те строки файла, которые содержат данное слово.
21. Дан текстовый файл. Напечатать в алфавитном порядке все слова из данного файла, имеющие заданную длину n.
22. Текстовый файл содержит запись многочлена некоторой степени с одной переменной x , коэффициенты многочлена — целые. Например, $5x^4 - 3x^3 + 15x^2 - 4$. Указать степень многочлена, его коэффициенты. Дописать в указанный файл таблицу значений этого многочлена на данном отрезке $[a, b]$.
23. Дан файл, содержащий текст. Подсчитать количество слов, начинающихся и заканчивающихся на одну и ту же букву.
24. Ввести с терминала ряд чисел вещественного типа и вывести на экран и в файл по строкам введенные значения и их квадраты.
25. Разработать программу которая формирует файл F1, содержащий целые числа, и переписывает этот файл в другой файл - F2, помещая в него из F1 только положительные числа.
26. Разработать функцию "triangle(F,N)", формирующую текстовый файл F из N строк, в первой из которых одна литера '1', во второй две литеры '2', и т.д.
27. В текстовом файле задана последовательность вещественных чисел, разделенных пробелами. Разработать функцию Max(F) для нахождения максимального из этих чисел. Имя файла передать функции как параметр.

Литература

1. Керниган, Брайан В. Язык программирования Си : пер. с англ. / Б. Керниган, Д. Ритчи. — 3-е изд., испр. — СПб. : Невский Диалект, 2001. — 352 с.
2. - Язык программирования C++ : Лекции и упражнения : учебник : пер. с англ. / С. Прата. — СПб. : ДиаСофтЮП, 2003. — 1097 с.
3. -Подбельский, Вадим Валерьевич. Программирование на языке СИ : учебное пособие для вузов / В. В. Подбельский, С. С. Фомин. — 2-е изд., доп. — М. : Финансы и статистика, 2003. — 600 с.
4. Березин, Борис Иванович. Начальный курс С и С+ / Б. И. Березин, С. Б. Березин. — М. : Диалог-МИФИ, 1996. — 288 с

Оглавление

Лабораторная работа №1: «Использование условных операторов»	3
Краткие сведения по языку СИ	3
Структура простейшей программы:	3
Форматированный вывод на экран.	4
Форматированный ввод с клавиатуры.....	6
Программирование ветвлений.....	7
Блок-схемы программ	10
Основные правила составления блок – схем:	10
Примеры простейших алгоритмов:.....	10
Индивидуальные задания по теме: «Условные операторы»	12
Лабораторная работа №2: «Программирование циклических алгоритмов»	15
Общие сведения	15
Цикл с предусловием.....	15
Цикл с постусловием.....	15
Цикл с параметром.	16
Отладка программ.....	18
Синтаксические ошибки	19
Ошибки в процессе работы программы	20
Трассировка программ	21
Просмотр текущих значений переменных и выражений	21
Индивидуальные задания по теме: «Циклические алгоритмы»	23
Лабораторная работа №3: Массивы и строки.....	26
Общие сведения о массивах	26
Многомерные массивы.....	28
Работа с символьными строками	29
Индивидуальные задания по теме: Строки.....	35
Индивидуальные задания по теме: Массивы.....	38
Лабораторная работа №4: «Функции».....	41
Определение функции.....	41
Формат определения функции:	41
Вызов функции.....	41
Прототипы функций.....	42
Использование библиотечных функций.....	44
Рекурсивные определения функций.....	44
Передача значений через глобальные переменные.....	45
Индивидуальные задания по теме: «Функции.».....	46
Лабораторная работа №5: Работа с файлами.....	48

Общие сведения:	48
Форматированный ввод -вывод.....	51
Варианты индивидуальных заданий по теме: «Файлы».....	54
Литература	56

Учебное издание

ШЕСТАКОВ Василий Васильевич

ВВЕДЕНИЕ В ЯЗЫК «СИ»

Методические указания к выполнению лабораторных работ
по курсу «Компьютерные технологии в приборостроении»
для студентов II курса, обучающихся по направлению
200100 «Приборостроение»


**Отпечатано в Издательстве ТПУ в полном соответствии
с качеством предоставленного оригинал-макета**

Подписано к печати _____.2010. Формат 60х84/16. Бумага «Снегурочка».
Печать XEROX. Усл.печ.л. 9,01. Уч.-изд.л. 8,16.
Заказ . Тираж экз.



Национальный исследовательский Томский политехнический
университет
Система менеджмента качества
Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту ISO 9001:2008



ИЗДАТЕЛЬСТВО  ТПУ. 634050, г. Томск, пр. Ленина, 30
Тел./факс: 8(3822)56-35-35, www.tpu.ru