Федеральное агентство связи
Государственное образовательное учреждение
высшего профессионального образования

“САНКТ-ПЕТЕРБУРГСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ”
им.проф.М.А.Бонч-Бруевича

Л.А.Груздева
М.Ю.Колесникова

ИНФОРМАТИКА. ОСНОВЫ АЛГОРИТМИЗАЦИИ

Учебное пособие

САНКТ-ПЕТЕРБУРГ

2017

СОДЕРЖАНИЕ

Введение…….4
Тема 1. Линейные вычислительные процессы…10
Теоретический материал……..
Язык программирования С…… .
… Типы данных……. 12
Структура программы…...…….
Переменные и выражения……
Операции. Приоритеты операций……14
Математические функции…….15
Ввод/вывод информации…….17
Пример линейной программы……
Общие требования к выполнению лабораторных работ и написанию отчетов……23.
Варианты заданий к работе 1……25.
Тема2.Разветвляющиеся вычислительные процессы……27
Теоретический материал……..
Управляющие операторы, реализующие разветвляющие алгоритмы……..
Операторы *switch и break*……29…
Условный оператор *if*……..
Пример на работу оператора *SWITCH*…..
Пример разветвляющейся программы……
Варианты заданий к работе 2……
Тема 3. Циклические вычислительные процессы
Теоретический материал……
Цикл *for*……
Пример циклической программы.
Построение таблицы значений функции……
……..
Пример цикла с предусловием……...
Цикл с постусловием…….
Варианты заданий к работе 3…….
Тема 4. Массивы ……..
Теоретический материал……
Одномерные массивы …….
Инициализация одномерных массивов…….

Пример программы, работающей с одномерным массивом…….
Варианты заданий к работе 4…….
Тема 5.Функции ……
Объявление(декларация) функции…….
Определение(описание) функции(стандарт ANSI)
Вызов функции……
Обмен информацией между функциями……..……
Передача параметров

ВВЕДЕНИЕ

Учебное пособие посвящено изучению основ программирования на базе языков C/C++.

Существует более 3000 языков программирования.

C- универсальный язык, эффективный и удобный для решения самых различных задач. Первоначально автором его был Денис Ритчи. Язык тесно связан с системой UNIX,так как был в этой системе и разработан.

Однако язык не привязан к какой-либо системе, он универсален. Определение современного языка C дано в соответствии со стандартом ANSI(American National Standarts Institute). В начале 80-х годов Бьерн Страуструп вместе с коллегами AT&T Bell Laboratories разработал C++, который является расширенной версией языка C с подключенными средствами поддержки объектно-ориентированного программирования(ООП). В настоящее время язык продолжает непрерывно развиваться. Это гибкий, мощный, имеющий эффективный объектный код язык. Стандартом языка C++ является ISO/IEC 14882 “Standard for the C++ Programming Language” (1998).

Язык программирования C характеризует:

-скорость работы написанных программ соизмерима с языком ассемблера, при этом программы более наглядны и просты;

-большая распространенность;

-переносимость программ;

-совместимость с большинством других средств программирования;

-легкий доступ к аппаратным средствам, включение языка ассемблера;

-отсутствие ограничений на режимы работы с памятью;

-отсутствие жесткого контроля над действиями программиста.

На С написаны: операционные системы Windows, UNIX; языки программирования для UNIX Паскаль, АПЛ ,Лисп и многие др.программы.

АЛГОРИТМИЗАЦИЯ –описание очередности выполнения различных операций, необходимых для решения той или иной задачи в форме алгоритма.

АЛГОРИТМ (алгорифм)-совокупность предписаний о последовательном выполнении системы различных операций(вычислений), необходимых для решения очередной задачи.

Важность составления алгоритма для решения задач повысилась в связи с развитием ЭВМ, используя которые можно реализовывать большинство алгоритмов.

ОСНОВНЫЕ ЭТАПЫ СОЗДАНИЯ ПРОГРАММ

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

-Постановка задачи- содержательная постановка задачи, определение конечных целей решения.

-Алгоритм- точно определенная последовательность действий, необходимых для решения задачи.

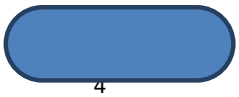
-Блок-схема – графическая форма записи алгоритма.

-Программа-последовательность команд ЭВМ, реализующих алгоритм и данные, необходимые для его выполнения.

-Алгоритм + Структура данных=ПРОГРАММА

ТИПОВЫЕ БЛОКИ ДЛЯ ЗАПИСИ АЛГОРИТМА

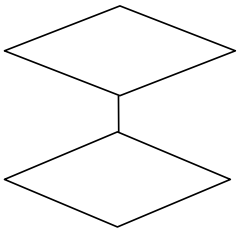
Начало программы или функции



Все операции ввода/вывода

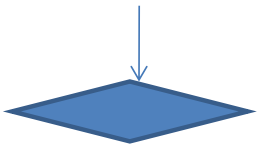
Блок обработки данных

Вызов функции



Проверка условий

Переключатель



Конец программы или функции

Первая программа

```
#include<stdio.h>
int main()
{printf(“ \n Привет! “);}
```

Комментарии к программе:

#include<stdio.h> Включение информации о стандартной библиотеке.

printf(); Вызов функции вывода на экран.

{ } Тело функции main().

Программа состоит из функций и директив препроцессора. Одна из функций является главной. Имя главной функции main(). Выполнение программы начинается с первого оператора этой функции. Завершение работы этой функции означает завершение работы всей программы. Поэтому в любой программе имеется функция с именем main(). Она может быть единственной, если программа состоит из одной функции. Все остальные функции программы имеют отличные от main имена.

Определение любой функции имеет вид :

```
<тип><имя_функции>([параметры])
{операторы тела функции}
```

Параметры помещаются в круглые скобки, наличие квадратных скобок означает, что внутри круглых скобок параметры могут быть, но могут и не быть.

Примечания:

1.Если функция возвращает в точку своего вызова некоторое значение ,но тип возвращаемого значения не указан, то тип по умолчанию считается , что этот тип- int.

2.Если функция не возвращает значение, то тип void(пустой).

3.Тело функции- это блок, который заключается в фигурные скобки.

4.Функции не могут быть вложенными .

5.Операторы функции заканчиваются точкой с запятой.

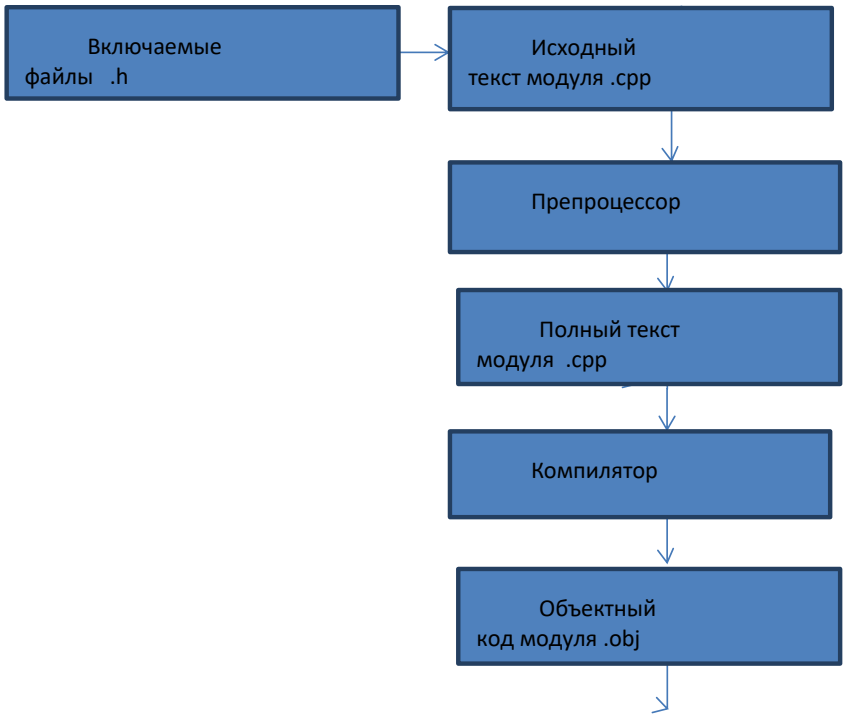
Путь прохождения программы показан на Рис.1.

Программу, написанную на языке высокого уровня необходимо перевести на язык , понятный процессору – в машинные коды. Специальные программы, осуществляющие перевод с языков высокого уровня в коды машины, называются компиляторами. Для того чтобы программа стала исполняемым файлом (файлом, имеющим расширение .exe), она должна быть обработана

специальной программой –компоновщиком (редактор связей) . На этапе компоновки подключаются библиотеки функций, которые хранятся в уже откомпилированном виде.

Путь прохождения программы состоит из нескольких этапов Исходный текст программы на языке C /C++ пишется в редакторе , затем передается препроцессору, который в соответствии с директивами , содержащимися в программе , присоединяет к тексту программы текст заголовочных файлов, образуя полный текст модуля с расширением .cpp. Этот текст поступает на вход компилятора, который сначала выделяет лексемы, затем в соответствии с грамматикой языка на основе этих лексем формирует выражения и операторы, ищет синтаксические ошибки и выдает соответствующие сообщения.

Компиляция программы



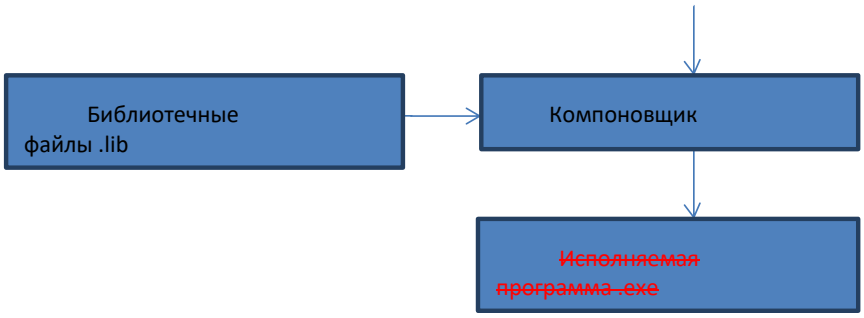


Рис.1 Путь прохождения программы в компьютере.

После исправления ошибок строит объектный модуль. Далее объектный модуль поступает на компоновщик , который формирует исполняемый модуль и идет подключение к объектному модулю необходимых библиотек функций. На выходе компоновщика получаем исполняемый модуль, который и запускается на исполнение.

Интегрированная среда

Для облегчения работы компилятор, компоновщик , а также редактор текста программ, отладчик и другие средства объединяют в так называемые интегрированные среды программирования(IDE- Integrated Development Environment). Различные среды программирования создаются под различные платформы и имеют различный интерфейс.

Студентам предлагается работать в интегрированной среде Dev C++ или Code::Bloks, которые работают на таких платформах, как, например, Unix,Linux,Windows и научиться создавать программы на языке C/C++.Программы должны иметь достаточно простую структуру , определенную упорядоченность, хорошо воспринимаемую при чтении и высокую надежность. Для этого будем использовать технологию создания программ, которая носит название структурное программирование.

Особенностью структурного программирования являются:

- модульность ;

-использование базовых конструкций следования, ветвления и цикла;

-ограниченное использование глобальных переменных. Модульность позволяет свести к минимуму взаимодействие логически независимых частей программы и осуществить принцип сокрытия информации. В С++ отдельным модулям будут соответствовать файлы с исходным текстом.

Структурное программирование напрямую связано с фон-неймановской архитектурой(управление потоками команд).

Машина фон-Неймана – это однородная, состоящая из ячеек память, и процессор, имеющий локальную память в виде регистров. Процессор может

Машина фон-Неймана – это однородная, состоящая из ячеек память, и процессор, имеющий локальную память в виде регистров. Процессор может загружать данные из памяти в регистры, выполнять арифметические и логические операции над содержимым регистров и память.

Программа для таких машин – это последовательность команд, выполняющих перечисленные операции вместе с дополнительным кодом управления, влияющим на выбор очередной команды.

Команды интегрированной среды.

Все действия можно выполнить либо выбрав пункт меню, либо нажав установленные функциональные клавиши.

‘+’ – одновременное нажатие нескольких клавиш

’,’ –последовательное нажатие клавиш.

F1 – помощь.

Ctrl+F1 – помощь по конкретному слову.

F10 – выход в меню.

Alt+X –выход в DOS.

F10,’F’,’N’ – создание нового файла.

F3 - загрузка существующего

F2 – запись редактируемого файла на диск.

Ctrl+’K’,’B’ – отметить начало блока.

Ctrl+’K’,’K’ – отметить конец блока.

Ctrl+’K’,’C’ – копировать отмеченный блок в указанное место.

Ctrl+’Y’ – удалить строку

F6 – переход между окнами.

Alt+<Номер> - переход к окну с номером.

Alt+F3 – закрытие окна.

F9 – трансляция программы(проверить ошибки).

Ctrl+F9 –транслировать и запустить на выполнение.

F7 – пошаговое выполнение программы.

F4 – выполнить до указанного места.

Ctrl+F7 – контроль значений переменных.

Alt+F5 – показать экран.

ЯЗЫК ПРОГРАММИРОВАНИЯ С.

Язык состоит из алфавита и синтаксиса, который определяет правила построения выражений из этих символов . Минимальная единица языка- лексема . Она имеет самостоятельный смысл. Выражение определяет правило вычисления некоторого значения. Оператор задает законченное описание некоторого действия.

Символы алфавита С :

-строчные и прописные латинские буквы и знак подчеркивания;

-арабские цифры от 0 до 9;

-специальные символы:

( ) { } [ ] + - / % * . \ ‘ “ , : ; <=> & # ^ | ! ?

-пробелы, табуляция, переход на новую строку

Лексемы языка

-идентификаторы;

-ключевые слова (зарезервированные);

-знаки операций ;

-константы;

-разделители (скобки, точка, запятая, пробелы).

Идентификатор(имя программного объекта) – любая последовательность прописных и строчных букв латинского алфавита, цифр,знака подчеркивания. Внутри имени не могут использоваться пробелы.

Ключевые слова- зарезервированные идентификаторы , использующиеся компилятором.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	virtual
do	if	static	while и др.

Программист не может использовать ключевые слова для обозначения своих пользовательских идентификаторов.

Знаки операций – один или более символов, определяющих действие над операндами .Внутри знака операций пробелы не допускаются.

Операции бывают унарные (один операнд), бинарные(два операнда) и тернарные(три операнда) по количеству участвующих в них операндов.

Константы: целые, вещественные , символьные (литерные), стринговые.

Компилятор определяет принадлежность константы к тому или иному типу по внешнему виду . Константа представляет фиксированную величину, которая не может быть изменена в программе.

Целые константы могут задаваться в десятичном, восьмеричном, шестнадцатеричном форматах. В восьмеричном формате целая константа начинается с 0, за которым следуют восьмеричные цифры(от 0 до 7) , в шестнадцатеричном формате целая константа начинается с 0X, за которым следуют шестнадцатеричные цифры(от 0 до 9,A,B,C,D,E,F).

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

Вещественная константа в экспоненциальной форме представляется в виде мантиссы и порядка. Мантисса записывается слева от знака экспоненты(E или e), порядок – справа от знака. Пробелы внутри числа недопустимы, отделение целой части числа от дробной выполняется с помощью точки, а не запятой (0.42e-01, 3F 5).

Символьная константа содержит один или два символа, заключенные в апострофы(одинарные ‘A’, ‘\n’).

Строковая константа – это последовательность символов, заключенная в кавычки(“Изучаем C/C++”).

Excape последовательности(управляющие последовательности) начинаются с обратной косой черты:

\n - новая строка(при выводе на экран или принтер переводит строку).

\t – горизонтальная табуляция.

\v – вертикальная табуляция.

\r – возврат каретки.

\a – сигнал звонок.

\? – знак вопроса.

\’ – одиночная кавычка.

\” – двойная кавычка.

Комментарии.

В программе можно использовать комментарии, которые не влияют на выполнение программы, могут служить для пояснения текста программы.

Каждую строку символов можно начинать с двух символов “косая черта” //.

/*Все что заключено в этих “ скобках “ является комментарием*/

ТИПЫ ДАННЫХ

Данные , обрабатываемые на ПК, относятся к разным типам и в зависимости от этого по-разному хранятся и обрабатываются.

Тип данных определяет :

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

- множество значений , которые могут принимать величины этого типа ;

-внутреннее представление данных в памяти ПК;

-функции и операции, которые могут применяться к величинам этого типа.

Программист для представления объектов своей задачи сам выбирает тип каждой величины. Компилятор проводит проверку конструкций программы в соответствии с указанными типами. Тип величины определяет машинные команды, которые будут использоваться для обработки данных, объем выделяемой компилятором памяти под тот или иной объект программы.Понятие *тип* – фундаментальное для всех языков программирования. Есть несколько основных(базовых) типов данных:

int- целый,
char- символьный,
float- вещественный,
doubl- вещественный с двойной точностью,
void- пустой, не имеющий значения.

Переменные и выражения.

Выражения – это последовательность операндов, знаков операций и скобок. Операнды – это переменные, константы или другие выражения. Выражение определяет выполнение одной или нескольких операций по преобразованию информации. Порядок выполнения выражений определяется компилятором в соответствии с их приоритетами

Переменная – это именованная область памяти, в которой хранятся данные определенного типа. Значения переменной в ходе выполнения программы могут изменяться. Перед использованием любая переменная должна быть задекларирована . Тип объявляется при декларации переменных.

<имя типа><имя_переменной>, <имя_переменной>,…

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

<имя_переменной>;

int a; // целая переменная

float b; // вещественная переменная

char z; // символьная переменная

const int x=3; /* константа целого типа не может изменить свое значение в процессе выполнения программы. Модификатор const показывает, что значение переменной целого типа x изменить нельзя. Такую переменную называют именованной константой */

Пример, int x,y,z; //переменные x,y,z целого типа

double a,b; // переменные a,b типа double

Компилятор выделяет место в памяти компьютера в соответствии с декларацией. Важное значение имеет то, в каком месте программы эта декларация стоит . В C/C++ имеется три места , где переменная может быть объявлена. Если переменная объявлена *вне функций*_(в том числе и вне функции main) , она называется) *глобальной* (global) и видна из любого места программы(может быть использована везде).

Переменная может быть объявлена *внутри блока* , в том числе *внутри тела функции* , тогда она называется *локальной* (local) и может использоваться только внутри этого блока. Переменная может быть объявлена *как формальный параметр* функции, тогда переменная по сути локальна и служит для передачи информации в эту функцию. Формальным называется параметр, стоящий в круглых скобках в заголовке определения функции.

Замечание 1. Две глобальные переменные не могут иметь одинаковые имена.

Замечание 2. Локальная переменная одной функции может иметь такое же имя, как локальная переменная другой функции.

Замечание 3. Две локальные переменные в одном блоке не могут иметь одинаковые имена, и

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

Отформатировано: Шрифт: (по умолчанию) Times New Roman, курсив, без подчеркивания

локальный параметр не может совпадать с формальным параметром функции.

ОПЕРАЦИИ. ПРИОРИТЕТЫ ОПЕРАЦИЙ. Операции задают действия, которые необходимо выполнить.

Знак операции – это символ или комбинация символов ,которые сообщают компилятору о необходимости произвести определенные арифметические, логические или другие действия. Каждая операция имеет свой приоритет . Приоритет можно изменить с помощью круглых скобок . Если скобок несколько, то первым будет выполняться выражение во внутренних скобках.

АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ

Отформатировано: Шрифт: (по умолчанию) Times New Roman, без подчеркивания

- вычитание и унарный минус;

+ сложение ;

* умножение;

/ деление;

% деление по модулю;

++ увеличение операнда на 1(increment);

-- уменьшение операнда на 1(decrement);

Операции сложения, вычитания, умножения и деления действуют так же, как и в большинстве других алгоритмических языков. Они могут применяться ко всем типам данных. Операции выполняются слева направо.

Операция деления по модулю % дает остаток от целочисленного деления и может применяться только к целочисленным переменным.

Пример

```
#include<stdio.h>
int main()
{ int x,y;
printf("\n Введите делимое и делитель:");
scanf("\n %d%d",&x,&y);
printf("\n целая часть %d\n",x/y);
printf("остаток от деления %d\n",x%y);
}
```

Операции инкремента и декремента могут применяться только к переменным.

Существуют две формы записи: префиксная ++x, в которой

сначала выполняется увеличение операнда на единицу , а затем это увеличенное значение используется в выражении ;

постпрефиксная x++ , в которой в выражении используется исходное значение операнда и только потом это значение увеличивается на единицу . Три написанные ниже операции дают один и тот же результат, но имеют различие при использовании в выражениях x=x+1; ++x; x++.

Пример

```
#include<stdio.h>
int main() {
 int x=5;
 int y=60;
x++;
++y;
printf("x=%d y=%d\n",x,y);
printf("x=%d y=%d\n", x++,++y);
}
return 0;
```

Протокол работы программы:

x=6, y=61.
x=6, y=62.

ОПЕРАЦИИ ОТНОШЕНИЙ:

Отформатировано: Шрифт: (по умолчанию) Times New Roman, без подчеркивания

> больше
>= не меньше
< меньше
<= не больше
== равно
!= не равно

Операции отношений используются в условных выражениях, в которых проводится проверка некоторого условия на истинность(true) или ложь(false).

ЛОГИЧЕСКИЕ ОПЕРАЦИИ

Отформатировано: Шрифт: (по умолчанию) Times New Roman, без подчеркивания

В языке C/C++ имеются три логические операции :

&& (and) логическое И

|| (or) логическое ИЛИ

! (not) отрицание

Примеры условных выражений:

a<0 , ‘a’ ==’A’, ‘a’!=’A’ , !(a>b), (5<a)&&(a<8), (ch==27)||(a=10).

ОПЕРАЦИИ ПРИСВАИВАНИЯ

= простое присваивание

+= дополнительные

-= операции

*= присваивания

/=

%=

Вместо выражения n=n+5 можно использовать выражение n+=5. Здесь += аддитивная операция присваивания , в результате выполнения которой величина, стоящая справа , прибавляется к значению переменной, стоящей слева.

Аналогично

n-=5 тоже самое, что и n=n-5;

n*=3 тоже самое, что и n=n*3;

k/=10 тоже самое, что и k=k/10;

k%=6 тоже самое, что и k=k%6.

Операция n+=5 выполняется быстрее, чем n=n+5.

Эти операции имеют тот же приоритет , что и операция присваивания = ,т.е. ниже, чем приоритет арифметических операций .

Примечание. Разрешается использовать многократное присваивание: a=в=c=x*y.

Операция запятая.

Эта операция имеет самый низкий приоритет из всех операций . Операция запятая выполняется слева направо , и ее значением является значение правого операнда . В выражении (выр.1) , (выр.2) сначала вычисляется значение (выр.1), а затем -

Отформатировано: Шрифт: (по умолчанию) Times New Roman, без подчеркивания

Отформатировано: Шрифт: (по умолчанию) Times New Roman, без подчеркивания

значение (выр.2). Это значение и будет результатом операции.

Операция sizeof.

Эта операция имеет две формы : *sizeof(тип)* или

sizeof(выражение). Результатом этой операции является целочисленное значение величины типа или выражения в байтах. При использовании второй формы значение выражения не вычисляется , а лишь определяется его тип. Тип данных указывает компилятору, сколько байтов надо выделить для размещения объекта.

МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

Отформатировано: Шрифт: (по умолчанию) Times New Roman, без подчеркивания

Прототипы стандартных математических функций находятся в заголовочном файле math.h

Прототипы некоторых математических функций приведены в таблице

Имя функции	Прототип	Действие
Sin	double sin(double num);	Вычисляет синус
Cos	double cos(double num);	Вычисляет косинус
Tan	double tan(double num);	Возвращает тангенс аргумента
Log	double log(double num);	Вычисляет натуральный логарифм
log10	double log10(double num);	Вычисляет логарифм по основанию 10
Exp	double exp(double num);	Возвращает степень числа е
Sqrt	double sqrt(double num);	Вычисляет квадратный корень
Fabs	double fabs(double num);	Возвращает модуль числа
Pow	double pow(double base,duble x);	Возводит число в степень

Floor	double floor(double num);	Округляет вниз
Ceil	double ceil(double num);	Округляет вверх
Abs	int abs(int num);	Возвращает модуль числа (<stdlib.h>)

ТЕМА 1.

ВВОД-ВЫВОД ИНФОРМАЦИИ В С/С++

Любая задача, реализуемая в программе должна получать данные из внешнего мира и возвращать обратно результат. Таким образом, в задачах нужно организовать ввод-вывод данных разных типов. Для языков С и С++ ввод-вывод осуществляется по-разному.

Ввод-вывод в языке С называется *форматированным,* так как можно управлять представлением данных (форматом).

Для реализации ввода-вывода в С необходимо подключить стандартную библиотеку ввода-вывода stdio.h, которая содержит прототипы функций ввода - *scanf( )* и вывода – *printf( ).*

Оператор форматированного вывода *printf()* имеет обобщенную форму записи:
printf(“строка_формата”,перем_1, перем_2, . . ., перем_k);
где строка_формата определяет формат вывода (представление) данных и может содержать:

- текст, выводимый на экран и, как правило, поясняющий значения выводимых данных
- управляющие символы, например, ‘\n’ – перевод строки, ‘\t’ - горизонтальная табуляция
- спецификация форматов, которая определяет вид выводимых значений переменных. Одна спецификация соответствует одной выводимой переменной; спецификации перечисляются в спецификации форматов в том же порядке, в котором перечисляются выводимые

переменные в списке вывода; каждая спецификация начинается с символа %; спецификация стоит в том месте текста, где будет выводиться значение соответствующей переменной. Строка формата в общем виде выглядит следующим образом:

%*ширина.точность спецификатор*, где

Ширина- натуральное число, определяющее сколько позиций отводится под вывод числа, при этом, если выводимое число короче ширины вывода- то перед ним выведутся пробелы, если число длиннее ширины вывода, то оно будет выведено целиком, а значение ширины проигнорируется.

Точность- применима только к вещественным числам и показывает количество выводимых символов дробной части числа.

Спецификатор – указывает вид выводимого числа. Основные спецификаторы :

d- выводится целое число

f- выводится вещественное число с фиксированной точкой

e- выводится вещественное число с плавающей точкой

c- выводится символ

s- выводится строка

Например:

```
#include <stdio.h>
int main ( )
{ int a=18765;
float b=126.789456;
printf("\na=%d",a);
printf("\na=%8d",a);
printf("\na=%4db=%f",a,b);
printf("\nb=%8.3f",b);
printf("\nb=%3.2f",b);
printf("\nb=%e",b);
return 0;}
```

После запуска программы на экране увидим:

a=18765 переменная а вывелась целиком, без форматирования

a= 18765 под переменную a отвели 8 знакомест, из них первые 3 заполнились пробелами

a=18765b=126.789456 переменная a длиннее ширины поля вывода, поэтому оно игнорировано, переменная b вывелась без перевода строки

b= 126.789 под переменную b отвели 8 знакомест, из них 3-на дробн. часть, перед целой частью поставлен 1 пробел

b=126.78 длина переменной b больше ширины поля вывода поэтому ширина проигнорирована

b=1.26789456E+02 вывод переменной b в формате с плавающей точкой (в экспоненциальной форме) $1.26789456*10^2$

Оператор форматированного ввода *scanf()* имеет обобщенную форму записи:

scanf("строка_формата", адрес_1, адрес_2,... ,адрес_k);

Функция форматированного ввода данных с клавиатуры *scanf()* выполняет чтение данных, вводимых с клавиатуры, и преобразует их во внутренний формат. При этом возможно задавать правила интерпретации вводимых данных с помощью спецификаций строки_формата. Строка_формата аналогична используемой в функции *printf()*. При этом, для указания адреса переменной используется символ '&':

адрес=&переменная

Например:

```
#include <stdio.h>
int main ( )
{ int a;
float b;
printf("\n Введите целое и вещественное число");
scanf("\n%d%f",&a,&b);
printf("\n%d*%f=%6.2f",a,b,a*b);
return 0; }
```

После запуска программы на экране увидим:

Введите целое и вещественное число

6 2.456 (*ввели с клавиатуры 2 числа через пробел)*

6*2.456= 14.73

Несколько переменных подряд вводятся либо через пробел (тогда ввод производится в строку), либо, разделяя нажатием клавиши Enter, тогда каждая переменная вводится с новой строки.

Ввод-вывод в языке С++ называется потоковым, так как информация вводится и выводится в виде потока байтов – символ за символом.

Для возможности использования такого способа ввода-вывода необходимо подключить библиотеку *iostream* (stream-поток), в котором содержится описание стандартных потоков ввода данных с клавиатуры *cin* и вывода данных на экран *cout*, а также соответствующие операции

<< - операция записи данных в поток;

>> - операция чтения данных из потока.

Компилятор определяет тип вводимой (выводимой) переменной и соответствующим образом форматирует её.

Например

```
#include <iostream>
using namespace std;                    // открывает
//пространство имен библиотеки iostream
int main ( )
{ int a;
float b;
cout<<"\n Введите a и b"<<endl;     // Функция endl
                    // переводит строку аналогично '\n'
cin>>a>>b;   // Ввод значений в переменные a и b //из стандартного потока cin
cout<<endl<<a<<"*"<<b<<"="<<a*b;      //вывод результатов в стандартный поток cout
return 0; }
```

Форматирование данных в потоке.

Для управления форматом вводимого (выводимого) значения используются так называемые манипуляторы. Это функции, которые вставляются между вводимыми (выводимыми) значениями и изменяют состояние потока.

Для использования манипуляторов необходимо подключить заголовочный файл <iomanip>.

setw(n)- устанавливает минимальное количество символов, используемых для вывода значения. Действие
манипулятор setw распространяется только на одно вводимое (выводимое) значение, где n-натуральное число.

setprecision(n)- задаёт точность для вещественных чисел. По умолчанию точность равна 6; n-натуральное число, определяющее количество выводимых цифр (суммарно, до точки и после точки). Изменения, внесенные данным манипулятором, остаются в силе до конца работы программы или до отмены другим манипулятором.

Пример

```
cout<<endl<<a<<"/"<<b<<"="<<a/b;
```

На экране отобразится:

14/5.45=2.5688

```
cout<<endl<<setw(7)<<a<<setw(5)<<"/"<<b<<"="<<
setpresicion(10)<<a/b;
```

На экране отобразится:

14 /5.45=2.56880733

Явное преобразование типов данных.

При выполнении вычислений в С действует правило: результат любого вычисления будет преобразовываться к наиболее точному типу данных, из тех типов данных, которые участвуют в вычислении. То есть, если операнды имеют разные типы (например, int и float), то результат будет иметь тип float.

Неверный результат может получиться, если оба операнда целые:

```
int a=9, b=2;
```

cout<<9/2;

на экране увидим ответ 4, так как при делении целого на целое получим целый результат.

Для того чтобы получить правильный ответ, необходимо выполнить *явное преобразование* типа результата, указав его тип в скобках перед результатом в операторе вывода:

cout<<(float)9/2;

на экране увидим ответ 4.5

ОБЩИЕ ТРЕБОВАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ И НАПИСАНИЮ ОТЧЕТОВ

Отформатировано: Шрифт: (по умолчанию) Times New Roman, без подчеркивания

1. Разработка любой программы должна начинаться с постановки задачи;
2. Поставленная задача должна на предмет использования в программе тех или иных структур данных.
3. Определение общей структуры программы и возможность разбиения задачи на функционально законченные подзадачи меньшей сложности .
4. Использование структурного программирования .

- Программа должна быть написана на языке C (или C++).

-Контрольные расчеты должны быть выполнены до запуска программы (в системе MathCAD, в системе Matlab, в системе Maxima); контрольные расчеты предполагают набор проверочных данных и получение соответствующих этим наборам результатов. При тех же исходных данных результаты работы программы должны совпадать с контрольными расчетами.

- Ввод/вывод исходных данных и результатов в работах с линейными , разветвляющимися и циклическими программами должен производиться в стиле C (или C++).

- Обработка данных в соответствии с заданием.

- Отладка программы, то есть процесс исправления ошибок.

- Тестирование, то есть проверка правильности работы для всех ветвей алгоритма.

Лабораторная работа №1

Типы данных в языке С. Ввод и вывод данных в С.

1.Изучите текст программы и определите назначение ее отдельных элементов

Введите текст программы и сохраните этот файл на диске.

```
#include<stdio.h>  /*Директива препроцессора*/
int main( )
{
/* объявление переменных:*/
float a;        /*вещественной  */
int b;  /* целой */
char c;         /*символьной   */
char s[10];   /*строки символов   */
printf("\       n введите переменные разных типов\n");
printf("\       n введите a – типа float \n");
scanf("%f",&a);
printf("\       n введите b-  типа int \n");
scanf("%d",&b);
printf("\       n введите c – типа char  \n");
scanf("%c",&c);
scanf("%f",&a);
printf("\       n введите строку      s длиной   <=9 символов \n");
scanf("%s",&s);
/*Вывод результатов   */
printf("\       n  float  a=%e",a);
printf("\       n  float  a=%f",a);
printf("\       n  float  a=%12f",a);
printf("\       n  float  a=%6.1f",a);
printf("\       n  int b=%d",b);
printf("\       n  int b=5d",b);
printf("\       n  char c=%c",c);
```

```
printf("\      n  char c=%4c",c);
printf("\      n string s=%s ",s);
return 0;
}
```

2. Составьте программу, вычисляющую сумму, разность, произведение и частное двух целых чисел. Результат выводится на экран в десятичном, восьмеричном и шестнадцатеричном представлении . Арифметические операции обозначаются значками +, -, *, / соответственно . Оператор присваивания обозначается знаком = .

Тема2.

ЛИНЕЙНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ

Вторая работа посвящена линейным вычислительным процессам. Такие процессы реализуются с помощью простых программ, в которых все операторы выполняются линейно , один за другим сверху вниз (слева направо).

Лабораторная работа 2.

Цель работы. Целью настоящей работы является

получение студентами практических навыков в решении на ЭВМ задач, связанных с вычислением значений по заданным формулам.

Необходимо решить задачу вычисления и вывода на экран значений функций $y = f_1(x)$ и $z = f_2(y, a, b)$. Варианты заданий, а также рабочий набор данных приведены в табл.1.

Таблица1.

N	Функция $y = f_1(x)$	Функция $z = f_2(y, a, b)$	Рабочий набор		
			x	a	B
1	$\frac{\sqrt{x^2+16}}{x+2}$	$\frac{y+\sqrt{\sin a+3}+b}{y^2+\sqrt{\sin a+3}}$	3,5	1,8	3,7

2	$\frac{e^{x-2,7}+3}{x+1,3}$	$\frac{y+0,75\lvert\cos b\rvert+a}{y^2+0,75\lvert\cos b\rvert}$	8,2	2,2	8,2
3	$\frac{\sin x+1,5}{2}$	$\frac{y^3+\sqrt{\sqrt{a}+3,3}}{b+\sqrt{\sqrt{a}+3,3}}$	8,1	0,8	1,2
4	$\frac{\ln(x-3)+4}{x^2+12}$	$\frac{\sqrt[3]{y+7}+a}{\sin b+\sqrt[3]{y+7}}$	4,7	7,6	8,1
5	$\frac{\lvert x\rvert+8}{x^3+18}$	$\frac{\sqrt[4]{y+15}+a}{\cos b+\sqrt[4]{y+15}}$	3,4	82	2,5
6	$\frac{\cos^2(x)+2}{3}$	$\frac{y\sqrt[5]{a}+1}{\sin b+y\sqrt[5]{a}}$	-8	8,7	1,3
7	$\frac{e^{x+3,1}+2}{x+6,1}$	$\frac{\sqrt[3]{a}+2y+\mathrm{tg}b+3}{\mathrm{tg}b+2y+3}$	2,5	8,7	1,8
8	$\frac{\sqrt{e^{x-2}+3}}{x}$	$\frac{\sqrt[4]{a}+\sqrt{5y+20}}{\sqrt{5y+20}+b}$	2,7	17	11
9	$\frac{\mathrm{tg}x+3,73}{4}$	$\frac{7y+3\sin a+\sqrt{b^2+19}}{7y+\sqrt{b^2+19}+2}$	0,1	1,5	10
10	$\frac{\sin^3(x)+3,7}{5}$	$\frac{\sqrt{14y+2}+6}{\sqrt{14y+2}+\cos b+a}$	2,5	5	6,1
11	$\frac{\sqrt{x+12}}{2x^3+1}$	$\frac{\lvert y^2-a\rvert+6}{2\cos b+\lvert y^2-a\rvert+6}$	18	-3	8,1
12	$\frac{\sqrt[3]{x+8,3}}{x+0,3}$	$\frac{4+y^2+\lvert\sin x\rvert+a}{\lvert\sin x\rvert+y^2+0,2b}$	3,7	-2	8,1
13	$\frac{1+\ln(x+5,3)}{x+5,3}$	$\frac{\sqrt{y+15\sin a}}{\sqrt{y+15\sin a}+2b}$	2	2	3
14	$\frac{e^{x-1,5}+2}{2x+0,3}$	$\frac{\sqrt[4]{27y+54}+a}{\sqrt[4]{27y+54}+\lvert\cos b\rvert+1}$	4,1	9	3,5
15	$\frac{\lvert\sin x\rvert+2}{3}$	$\frac{\sqrt[3]{y+7a}+b}{\sin b+1+\sqrt[3]{y+7a}}$	2,5	1,3	3,3
16	$\frac{2}{\sqrt{\cos x+5}}$	$\frac{\sqrt[3]{y+13a}+5}{\cos b+\sqrt[3]{y+13a}}$	6,1	2,3	2,6

17	$\frac{\sqrt{\lvert\cos x+3\rvert}}{3}$	$\frac{\sin b+\sqrt[4]{y+15a}}{\sqrt[4]{y+15a}}$	8	1,3	2,5
18	$\frac{\sqrt[3]{x-3{,}1}}{x-27}$	$\frac{(y+1)^2+5a}{\sin b+(y+1)^2+5a}$	80	0,8	-2
19	$\frac{3e^{x-2}}{x+1}$	$\frac{a\sqrt[3]{y+2b}}{2-\cos b+\sqrt[3]{y+2b}}$	6,1	8	9,2
20	$\frac{\sin^2(x)+5}{5}$	$\frac{\sqrt[3]{\mathrm{ay}+57}}{3+\lvert\cos b\rvert+\sqrt[3]{\mathrm{ay}+57}}$	-2	7,3	5,1
21	$\frac{x-7}{\ln(x-2)+2}$	$\frac{\sqrt[4]{\mathrm{ay}^2+3}+2}{\sqrt[4]{\mathrm{ay}^2+3}+b}$	10	23	1,1
22	$\frac{\sqrt{\cos^2 x+10}}{5}$	$\frac{\lg(y^2+8)+5\sin a}{\lg(y^2+8)+\lvert\cos b\rvert}$	5,2	2,5	7,2
23	$\frac{\lg(17-2x)+2}{x+1}$	$\frac{\lvert\cos a\rvert(y+3)}{\lvert\cos a\rvert(y+3)-b}$	0,6	5	2,1
24	$\frac{e^{2x-7{,}4}+6}{x+4{,}3}$	$\frac{\lvert\sin a\rvert(y+7)}{\lvert\sin a\rvert(y+7)+2b}$	5	-2	0,7
25	$\frac{3\sin x+21}{\cos 2x+25}$	$\frac{\lg(y^2+99)+a^2}{\lg(y^2+99)+b^2}$	3,5	14	7
26	$\frac{\sqrt[3]{\lvert\sin x\rvert+26}}{\sin(2x)+5}$	$\frac{\lg(17y^2+83)+a^2}{\lg(17y^2+83)+b}$	2	8	4,3
27	$\frac{\sqrt[3]{\lvert\cos x\rvert+8}}{\cos 2x+5}$	$\frac{\cos^2(\mathrm{ay})+5b}{\cos^2(\mathrm{ay})+b}$	-2	8,1	13
28	$\frac{\lvert\lg x\rvert+5}{x+4}$	$\frac{\sin^2(a(2y^2+1))+29b}{\sin^2(a(2y^2+1))+b}$	0,2	2	12
29	$\frac{14\lvert\lg x\rvert+2}{40+x}$	$\frac{(\cos a+11y^2)^2}{(\cos a+11y^2)+b}$	0,2	7,1	9
30	$\frac{4\sin^2 x+3}{2}$	$\frac{\sin^2(a(3y^2-\frac{1}{3}))+11{,}75}{\sin^2(a(3y^2-\frac{1}{3}))+b}$	2,2	3,2	6,8

Тема 3. РАЗВЕТВЛЯЮЩИЕСЯ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ

Теоретический материал
Базовые конструкции
структурного программирования

Для выполнения ветвления применяются операторы if и switch, а так же опция условия ? :

Условный оператор if

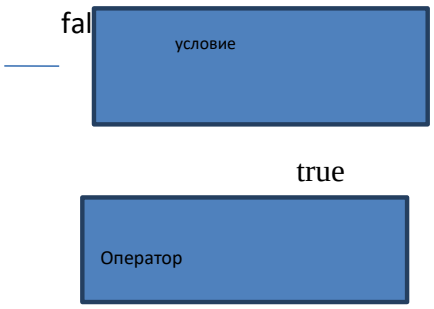


Рис.2 Блок-схема краткой формы оператора if

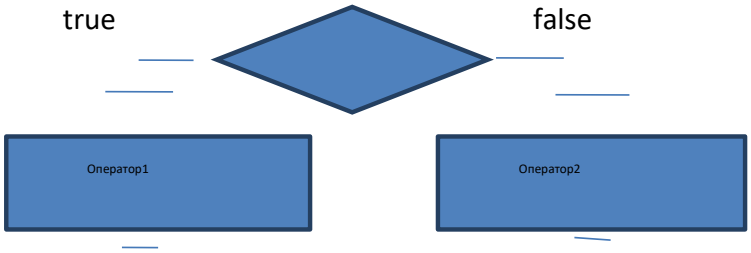


Рис.3 Блок-схема полной формы оператора if

Оператор if используется для ветвления, т.е. для выбора возможного продолжения

вычислительного процесса. Оператор if имеет две формы: краткую(1) и полную(2)

(1) if (выражение_условие) оператор;

(2) if (выражение_условие) опер
else оператор 2;

где в качестве выражения_условия могут использоваться арифметическое выражение, отношение и логическое выражение. Оператор, включенный в условный, выполняется только в случае истинности (true) (т.е.при ненулевом значении) выражения_ условия .

Алгоритм работы оператора if.

Отформатировано: Шрифт: (по умолчанию) Times New Roman, 14 пт, курсив

При выполнении операции условия сначала вычисляется выражение_условие (Результат вычисления может иметь арифметический тип или тип указателя) . Если результат вычисления отличен от нуля (true), то выполняется оператор или оператор 1. Если результат равен нулю (false) , то выполняется оператор 2.

Для случая (1) оператор пропускается, и управление передается на следующий после if оператор.

Оператор 1 и оператор 2 сами могут быть операторами if,образуя вложенный if.

Точка с запятой – есть заключительный символ любого оператора.

Пример.

Решить на компьютере задачу вычисления значений функции y=f(x), где

$$Y\begin{cases} 4+(x-2)^2 \text{при} x>2 \\ 2^x \text{при} 2 \geq x > 1 \\ x+1 \text{при} \geq x > 0 \\ e^x \text{при} x \leq 0 \end{cases}$$

Рабочий набор *x*=4.5 Использовать условный оператор if .

Текст программы:

```
#include<stdio.h>
#include<math.h>
int main( )
{ float x,y;
printf("\n input x=");
scanf("%f",&x);
if(x>2)
y=4+pow(x-2,2);
else

if(x>1)
y=pow(2,x);
else
if(x>0)
y=x+1;
else
y=log(x);
printf("\n y=%f при x=%f",y,x);
return 0;
}
```

Выполнить вариант при x =4.5 . Контрольные расчеты провести для всех ветвей алгоритма. В зависимости от введенного значения x, будет работать та или иная веточка алгоритма. Правильность работы программы должна быть для каждой веточки алгоритма.

Контрольный расчет :

При x=5 y=13 (x>3)

При x=1.5 y=2.828 (x>1)

При x=0.8 y=1.8 (x>0)

При x=0 y=2 (x<=0)

X=1.5 y=2.828 (x>1)

Примечание: В случае необходимости, (при заданном рабочем значении x=4.5) провести проверку типа 2>=x>1 следует написать if(x> 1&& x<=2), где &&-логическое И, которое означает одновременное выполнение двух условий , слева и справа от знака &&.

Пример на использование составного оператора. Даны два числа x1 и x2 . Большее значение присвоить переменной max ,а меньшее min.

Текст программы.

```
#include<stdio.h>
int main( )
 {float x1,x2,max,min;

//Ввод чисел x1,x2
if(x1>=x2) {max=x1; min=x2}; // составной оператор
else {max=x2; min=x1}; // составной оператор

//оператор
//Вывод max, min
return 0;
}
```

ОПЕРАТОРЫ SWITCH и BREAK

Оператор switch используется для выбора одного варианта из многих(вместо нечеткого *if-else*).

Формат оператора имеет вид:

```
Switch(выражение_селектор)
{case constant_1: < оператор1>;[break;]
case constant_2: <оператор2 >;[break;]
.  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
```

```
case constant_N: <операторN >;[break;]
default:<оператор N+1;>
}
```

Сначала вычисляется выражение в скобках за ключевым словом *switch*, затем просматривается список меток (case constant и т.д.) до тех пор, пока не находится метка , соответствующая значению вычисленного выражения. Далее происходит выполнение соответствующая последовательности операторов , следующих за двоеточием. Если же значение выражения не соответствует ни одной из меток оператора switch, то выполняется последовательность операторов , следующая за ключевым словом *default*. Тип значения выражения_селектора – целый: *char, int.*

Допускается конструкция оператора *switch*, когда слово *default* и соответствующая последовательность операторов может отсутствовать. Оператор *break* дает возможность осуществить быстрый выход из оператора *switch*. Если *break* будет отсутствовать, то придется проверить все метки, следующие после метки, совпадающей с селектором, включая *default* . Ветка *default* может отсутствовать.

Пример программы, позволяющей производить операции сложения, вычитания, умножения и деления.

```
#include<stdio.h>
int main( )
{float x,y,z;
char ch ;
printf("\n введите x,y");
scanf("%f%f",&x,&y);
printf("\n введите один из символов +,-,*,/");
scanf("%c",&ch);
switch(ch)
{case '+' : z=x+y; break;
case '-' : z=x-y; break;

case '*' : z=x*y; break;
```

```
case '/' : z=x/y; break;
default:printf("\n ошибка ввода");
}
printf("\n z=%f",z);
return 0;
}
```

Лабораторная работа 3.

Необходимо вычислить значение функции, заданной различными формулами на разных участках ее определения, на заданном наборе исходных данных.

№	*Функция y=f(x)*	*Рабочий набор данных*
1.	$\begin{cases} e^{\sqrt{x}} & \text{при} \quad x > 2, \\ \cos^2(x) & \text{при} \quad 2>=x > 1, \\ \lg(20x) & \text{при} \quad 1>=x > 0.5, \\ e^{0.2x} & \text{при} \quad x<=0.5 \end{cases}$	*4.3*
2.	$\begin{cases} \frac{1}{x} & \text{при} \quad x > 4, \\ e^{-x} & \text{при} \quad 2 < x<=4, \\ x^2 & \text{при} \quad -1 < x<=2, \\ \frac{\lvert x-1 \rvert}{2x} & \text{при} \quad x<=-1 \end{cases}$	*10.9*
3.	$\begin{cases} \frac{2}{x} & \text{при} \quad x < -5, \\ x\sin x & \text{при} \quad 2 > x>=-5, \\ \frac{x+10}{2+x^2} & \text{при} \quad 3 > x>=2, \\ x+\sqrt[3]{x} & \text{при} \quad x>=3 \end{cases}$	*-10*

4.	$\begin{cases} \sqrt{x} & \text{при} \quad x > 7, \\ e^{-x} & \text{при} \quad 2 < x <= 7, \\ e^{x} & \text{при} \quad -1 < x <= 2, \\ \sin(x) & \text{при} \quad x <= -1 \end{cases}$	*9.2*
5.	$\begin{cases} 3x^3 & \text{при} \quad x > 20, \\ \ln(x+3) & \text{при} \quad 20 >= x > 10, \\ e^{-x^2} & \text{при} \quad 10 >= x > 3, \\ \lvert\sin(x)\rvert & \text{при} \quad 3 >= x \end{cases}$	*25*
6.	$\begin{cases} \frac{\sin^2(x)}{2} & \text{при} \quad x > 20, \\ \sqrt[3]{x} & \text{при} \quad 20 >= x >= 6, \\ \sin^2(x) & \text{при} \quad 6 >= x > -4, \\ 0 & \text{при} \quad x <= -4 \end{cases}$	*-19.8*
7.	$\begin{cases} \sin(\sqrt{x}) & \text{при} \quad x > 30, \\ \sqrt{x} & \text{при} \quad 30 >= x > 10, \\ \lg(0.5*x) & \text{при} \quad 10 >= x > 2, \\ \lvert\sin(x)\rvert & \text{при} \quad x <= 2 \end{cases}$	*3.141*
8.	$\begin{cases} 2\sqrt{(x^2+15)} & \text{при} \quad x < -6, \\ 4\cos(x) & \text{при} \quad -6 <= x < 2, \\ \frac{\sin(x-3)}{2} & \text{при} \quad 2 <= x < 10, \\ \frac{\mathrm{tg}x}{10} & \text{при} \quad x >= 10 \end{cases}$	*0*
9.	$\begin{cases} 1 + x^2 & \text{при} \quad x > 25, \\ 2 + x^2 & \text{при} \quad 25 >= x > 8, \\ 3 + x^2 & \text{при} \quad 8 >= x > 2, \\ 4 + x^2 & \text{при} \quad x <= 2 \end{cases}$	*100*
10	$\begin{cases} x\lvert x+21\rvert & \text{при} \quad x < -14, \\ x^2\ln\lvert x^2+48\rvert & \text{при} \quad -14 <= x < -5, \\ \frac{x}{3} + \sqrt{(x^2+16)} & \text{при} \quad -5 <= x < 0, \\ 2 + \frac{x}{3} & \text{при} \quad x >= 0 \end{cases}$	*-15.5*
11	$\begin{cases} x + (\lvert x\rvert - 12)^2 & \text{при} \quad x < -8, \\ \frac{\sin(x)+3}{\cos(x)+15} & \text{при} \quad -8 <= x < 7, \\ x\sqrt{(x^2-36)} & \text{при} \quad 7 <= x < 10, \\ x + 8 & \text{при} \quad x >= 10 \end{cases}$	*7.6*

1 *2*	$\frac{\cos(x)+14}{\sin(x)+15}$ при x <=4, $\sqrt[3]{x+\ln(\lvert x-8\rvert)}$ при $4<x<12$, $\sqrt{x-13}$ при 12<= x < 38, $5x$ при x>=38 { { {	*40*
1 *3*	$\begin{cases} \sqrt{\lvert x\rvert}+\frac{\sin(x)}{2} & \text{при} \quad x<-12, \\ \sin(x-5) & \text{при} \quad -12<=x<8, \\ \frac{\sqrt{(x+6)}}{x+3} & \text{при} \quad 8<=x<10, \\ \frac{4}{3+x} & \text{при} \quad x>=10 \end{cases}$	*12*
1 *4*	$\begin{cases} \frac{x\sin(x)+5}{2+\sin(x)} & \text{при} \quad x<5, \\ \sqrt{\lvert x-7\rvert}+\frac{x}{2} & \text{при} \quad 5<=x<10, \\ \lg(x) & \text{при} \quad 10<=x<111, \\ 2x & \text{при} \quad x>=111 \end{cases}$	*1.5*
1 *5*	$\begin{cases} \sqrt{(\lvert x\rvert-1)} & \text{при} \quad x<=-20, \\ \frac{x+\sin(x)}{2} & \text{при} \quad -20<x<6, \\ x^2+\ln(x-2) & \text{при} \quad 6<=x<12, \\ x^2+\ln(10) & \text{при} \quad x>=12 \end{cases}$	*3*
1 *6*	$\begin{cases} \lg(1.5x) & \text{при} \quad 4<=x<7, \\ \sin^4(x) & \text{при} \quad x>=7, \\ 0.5e^{0.1x} & \text{при} \quad -3.5<=x<4, \\ \frac{\cos(x)}{\lvert x\rvert+10} & \text{при} \quad x<-3.5 \end{cases}$	*2*
1 *7*	$\begin{cases} \frac{\sin(x)+2}{\cos(x)+10} & \text{при} \quad x<=5, \\ \sqrt[3]{(2+\ln(\lvert x-8\rvert+2))} & \text{при} \quad 5<x<15, \\ \sqrt{x-13} & \text{при} \quad 15<=x<38, \\ 5+x & \text{при} \quad x>=38 \end{cases}$	*3.141*
1 *8*	$\begin{cases} \frac{x}{2} & \text{при} \quad x>5, \\ 10e^{-x} & \text{при} \quad 2<x<=5, \\ \frac{3+x^2}{2x} & \text{при} \quad -2<x<=2, \\ \frac{\lvert x-100\rvert}{2x} & \text{при} \quad x<=-2 \end{cases}$	*23*

19	$\begin{cases} \sqrt{x} & \text{при} \quad x > 9, \\ 5e^{-x} & \text{при} \quad 2 < x <= 9, \\ e^{x} & \text{при} \quad 1 < x <= 2, \\ \cos(x) & \text{при} \quad x <= 1 \end{cases}$	2
20	$\begin{cases} x^3 & \text{при} \quad x > 10, \\ 2x & \text{при} \quad 10 >= x > 3, \\ 0{,}5x & \text{при} \quad 3 >= x > -5, \\ \sqrt[3]{\lvert x+2 \rvert} & \text{при} \quad x <= -5 \end{cases}$	5.7
21	$\begin{cases} x^2 & \text{при} \quad x > 10, \\ \lg(x) & \text{при} \quad 10 >= x > 1, \\ \sqrt{\lvert x-15 \rvert} & \text{при} \quad 1 >= x >= -1, \\ 4 & \text{при} \quad x < -1 \end{cases}$	20.5
22	$\begin{cases} e^{-x} & \text{при} \quad x <= 0, \\ \sqrt[3]{x+1} & \text{при} \quad 0 < x <= 7, \\ \dfrac{x+13}{10} & \text{при} \quad 7 < x <= 87, \\ 10(x-86) & \text{при} \quad x > 87 \end{cases}$	-2.5
23	$\begin{cases} \dfrac{x+75}{\sqrt{x}+5} & \text{при} \quad x >= 25, \\ 0{,}4x & \text{при} \quad 25 > x >= 5, \\ \sqrt{x-1} & \text{при} \quad 5 > x >= 2, \\ e^{x-2} & \text{при} \quad x < 2 \end{cases}$	-3
24	$\begin{cases} \sqrt{x-1} & \text{при} \quad x >= 5, \\ \dfrac{x+15}{x+5} & \text{при} \quad 5 > x >= 0, \\ 3\cos x & \text{при} \quad 0 > x >= -2\pi, \\ 3 & \text{при} \quad x < -2\pi \end{cases}$	2.4
25	$\begin{cases} \sqrt{x^2+16} & \text{при} \quad x >= 3, \\ x & \text{при} \quad 3 > x >= 1, \\ x^2+1 & \text{при} \quad 1 > x >= 0, \\ e^{-x} & \text{при} \quad x < 0 \end{cases}$	11.5
26	$\begin{cases} x + (\lvert x \rvert - 12)^2 & \text{при} \quad x < -8, \\ \dfrac{\sin(x)+3}{\cos(x)+15} & \text{при} \quad -8 <= x < 7, \\ x\sqrt{(x^2-36)} & \text{при} \quad 7 <= x < 10, \\ x+8 & \text{при} \quad x >= 10 \end{cases}$	18.3
27	$\begin{cases} 3x^3 & \text{при} \quad x > 20, \\ \ln(x+3) & \text{при} \quad 20 >= x > 10, \\ e^{-x^2} & \text{при} \quad 10 >= x > 3, \\ \lvert \sin(x) \rvert & \text{при} \quad 3 >= x \end{cases}$	-16.1

2 8	$\begin{cases} \frac{x}{2} & \text{при} \quad x > 5, \\ 10e^{-x} & \text{при} \quad 2 < x <= 5, \\ \frac{3+x^2}{2x} & \text{при} \quad -2 < x <= 2, \\ \frac{\lvert x-100\rvert}{2x} & \text{при} \quad x <= -2 \end{cases}$	*-2.33*
2 9	$\begin{cases} \sqrt{\lvert x\rvert} + \frac{\sin(x)}{2} & \text{при} \quad x < -12, \\ \sin(x-5) & \text{при} \quad -12 <= x < 8, \\ \frac{\sqrt{(x+6)}}{x+3} & \text{при} \quad 8 <= x < 10, \\ \frac{4}{3+x} & \text{при} \quad x >= 10 \end{cases}$	*-0.56*
3 0	$\begin{cases} \sin(\sqrt{x}) & \text{при} \quad x > 30, \\ \sqrt{x} & \text{при} \quad 30 >= x > 10, \\ \lg(0.5*x) & \text{при} \quad 10 >= x > 2, \\ \lvert\sin(x)\rvert & \text{при} \quad x <= 2 \end{cases}$	*12.8*

ТЕМА 4.

ЦИКЛИЧЕСКИЕ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ

Циклы необходимы для многократного повторения одного или нескольких операторов, как правило, пока выполняется некоторое условие. Число таких повторений заранее известно или может определяться в процессе счета. Условия могут проверяться перед началом выполнения тела цикла или по окончании тела цикла.

В C/C++ в этом случае используются три вида операторов*: for,while и do-while*. В этих операторах должны присутствовать начальные условия, тело цикла, включающего операторы, которые должны выполняться несколько раз, модификация параметра цикла(изменение параметра цикла) и проверка условия продолжения цикла.

ОПЕРАТОР АРИФМЕТИЧЕСКОГО ЦИКЛА С ЗАРАНЕЕ ИЗВЕСТНЫМ ЧИСЛОМ ПОВТОРЕНИЙ *for*.

Формат оператора *for*:

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

Каждое из этих выражений можно опустить . Первое выражение служит для инициализации параметра цикла, второе – для проверки продолжения цикла, третье- для изменения параметра цикла(модификация параметра цикла).Эти три раздела в формате оператора цикла должны быть разделены точкой с запятой. Выполнение цикла происходит до тех пор, пока условное выражение истинно. Как только условие становится ложным, начинает выполняться следующий за циклом for оператор.

Цикл начинает работать в такой последовательности :

1.Присваивается начальное значение параметру цикла (параметр цикла должен быть типа int или char).

2.Вычисляется значение выражения, устанавливающего условие продолжения цикла.

3.Если выражение условия возвращает *true,* то сначала выполняется тело цикла, а затем заданная операция над параметром цикла.

На каждой итерации шаги 2 и 3 повторяются.

Простейший *пример* оператора цикла *for* :

```
for(i=0; i<10; i++)
{printf("%d\n",i);}
```

В результате выполнения этого оператора будут напечатаны в столбик цифры от 0 до 9. Для печати этих цифр в обратном порядке можно использовать оператор:

```
for(   ; i>=0; i--)
{printf("%d\n",i);}
```

В качестве параметра цикла можно использовать символьный тип, например:

```
Unsigned char ch;
for(ch='A'; ch<='Z'; ch++)
{printf("%c",ch);}
```

Случайно или намеренно может получиться цикл, из которого нет выхода, так называемый бесконечный цикл.

```
for(; ;) printf("Бесконечный цикл  "\n);
for(i=1; 1; i++) printf("Бесконечный цикл  "\n);
for(i=10;  i>5;  i++)  printf("Бесконечный  цикл
"\n);
```

Для таких циклов может быть организован выход, используя оператор *break*. Если оператор break встречается в составном операторе цикла, то происходит немедленное прекращение выполнения цикла и начинается выполнение следfующего оператора программы.

```
#include<stdio.h>
int main( )
unsigned char ch;
for( ; ;)
{ch=getchar() }  //прочитать символ
If(ch=='Q') break; //проверка символа
printf("%c",ch); //печать символа
```

В данной программе будут печататься символы до тех пор, пока не будет введен символ 'Q'.

Если в теле цикла встретился полезный оператор *continue*, то он передаст управление на начало следующей итерации цикла.

Тот факт, что оператор for допускает использование трех независимых параметра цикла(инициализацию параметра цикла, проверку продолжения цикла, изменение значения параметра цикла) открывает неограниченные возможности в управлении работой цикла.

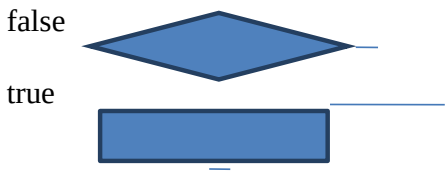


Рис.4 Блок-схема оператора цикла for

ПРИМЕР ЦИКЛИЧЕСКОЙ ПРОГРАММЫ .

ПОСТРОЕНИЕ ТАБЛИЦЫ ЗНАЧЕНИЙ ФУНКЦИИ.

Постановка задачи.

Дана функция вида $y=\frac{\sqrt{a+2.7}}{\sqrt{a+x^3}}$

Набор исходных данных:

N=6, a=5.1, xn=2, xk=3.

Необходимо построить таблицу значений функции для 6 значений аргумента от xn (xn начального) до xk (xk конечного). При программировании циклических алгоритмов с заранее известным количеством выполнений цикла(арифметические циклы) следует использовать оператор цикла *for*.

Текущее значение x будет меняться от xn до xk. Для определения шага dx при переходе от одного текущего значения x к следующему можно воспользоваться формулой*: dx=(xk-xn)/(N-1).*

При рассмотрении исходной формулы видим, что числитель дроби не зависит от x, поэтому, имеет смысл ввести дополнительную переменную

d=(a+2.7)^1/2, произведя таким образом, чистку цикла.

Текущее значение x при каждой итерации цикла будет меняться, увеличиваясь на величину *dx*, но при первом вхождении в цикл текущему значению x должно быть присвоено начальное значение *xn*. Это следует сделать до начала цикла: *x=xn*.

В теле цикла при каждой итерации будет вычислено значение функции *y* для очередного значения аргумента *x*. Но прежде чем вернуться в начало цикла для выполнения новой итерации необходимо изменить текущее значение аргумента, увеличив его на величину *dx*.

Программа построения таблицы значений функции имеет вид:

```
//Табулирование функции
# include<stdio.h>
#include<math.h>
#define xn 2
#define xk 3
#define a 5.1
int main( )
{int I,N;
float dx,d,x,y;
printf("Введите количество точек таблицы N=");
scanf("%d",&N);
printf("%6s%10s%10s\n", "НОМЕР", "АРГУМЕНТ  ", " ФУНКЦИЯ ");
x=xn;
dx=(xk-xn)/(N-1);
d=sqrt(a+2.7);
for(i=1; i<=N; i++)
{y=d/pow(a+pow(x,3),1.0/3);
printf((%6d %10.3f %10.3f\n",I,x,y);
x+=dx'//x=x+dx;
return 0;
}
```

В этой программе добавлена директива препроцессора *#define*, которая определяет подстановку в тексте программы

```
# define  имя  подстановочный текст
```

При любом появлении имени (если оно встречается и тексте, в заключенном в “ “) и не является частью определения другого имени , имя будет заменяться на соответствующий ему

подстановочный текст. Подстановочный текст может быть любой последовательность символов., среди которых могут быть не только цифры. Эти имена рекомендуется писать большими буквами , чтобы зрительно их можно было отличить от имен

переменных и функций. Это- именованные константы, а не переменные.

Результаты вычислений следует оформить в виде таблицы, снабженной заголовком.

При работе с циклическими алгоритмами следует сделать контрольный расчет в системе MathCAD, либо в системе Matlab,либо в системе Maxima.

Контрольный расчет

N	x	y
1	2	1.565
2	2.2	1.427
3	2.4	1.302
4	2.6	1.189
5	2.8	1.0893
6	3.0	0.9999

Лабораторная работа 4.

Протабулировать функцию y=f(x) на рабочем наборе исходных данных. Данные вводятся с клавиатуры, результат необходимо вывести на экран в виде таблицы. Выполнить задание с помощью оператора FOR

Варианты заданий к лабораторной работе 4 .

Таблица 3.

N	Вид функции $y = f(x)$	Рабочий набор исходных данных			
		N	a	$x_{нач}$	$x_{кон}$
1	$\frac{e^{-x} + e^{\sqrt{a}}}{e^{x-a}}$	15	1	0,2	0,5
2	$\frac{\cos^4 x + \cos^2 a}{\cos ax + 1{,}5}$	10	0,5	-1,3	1
3	$\frac{tgax\text{-}x^2}{2 + a^2}$	12	2	0,3	0,35

4	$\frac{\sqrt{a+2,7}}{\sqrt[3]{a+x^3}}$	16	1	2	3
5	$\frac{\ln\lvert a\text{-}x\rvert}{\ln(a+2)+\ln x}$	12	10	2	6
6	$\frac{e^{-a^2}+e^{-x^2}}{2,8ax}$	15	0,5	1	2
7	$\frac{\sin\sqrt{a^3+x}}{14+ax}$	12	1	2	3
8	$\frac{\cos^2 a+\cos ax}{ax+2}$	15	1,5	1	2,5
9	$\sqrt{\frac{1,6ax+\sqrt{x}}{2,9a+1,2}}$	10	0,1	4,2	6
10	$\frac{\ln\lvert a+x\rvert}{\ln\lvert a\rvert+1,5\ln\lvert x\rvert}$	8	-2,5	-1,9	-0,9
11	$e^{x+a^{1,7}}$	10	1,1	1	2
12	$\cos\frac{\sqrt{x}+\sqrt{a}+1}{\sqrt{ax}}$	12	3	2	3
13	$\frac{\sin ax+\sin^2 a}{4+\sin^2 x}$	15	2	1,5	2,9
14	$\frac{\sqrt[3]{a+x}+\sqrt[3]{a+2}}{a+x}$	10	3	1,5	3,5
15	$\ln\frac{\lvert 2ax\rvert}{\lvert a-x\rvert}$	12	3	2	3,5
16	$\frac{1}{e^{ax}+2e^{a}}$	15	1,5	1	2,5
17	$\frac{\sin^5 x+ax}{x+\cos^5 x}$	10	2	2,5	3,5
18	$\frac{\sqrt{a+3x}}{\sqrt{a}+3+x}$	12	2	0	5
19	$\sqrt[5]{\frac{a+30}{a+\sqrt{ax}}}$	20	4	1	10
20	$\frac{\ln(a^2+x^2)}{\lvert a+x\rvert}$	15	2	1	5
21	$1,5\cos^4\frac{a+x}{2a^2+x^2}$	10	1,5	1	4

22	$\frac{\sqrt{a+\sin x}}{\sqrt[3]{4+\cos x}}$	15	2	0	1
23	$\frac{\ln(x^4-a^2)}{a^4+28}$	18	1,5	2	3,5
24	$\frac{\mathrm{tg}ax+\mathrm{tg}x}{\mathrm{tg}(a+1)}$	15	1,2	0,1	0,25
25	$\frac{\sin^4 a+\sin^4 x}{a+x}$	12	0,5	-π	+π
26	$\frac{\sqrt{ax+0{,}2x}}{2a^2+x}$	15	1,5	2	4
27	$\sqrt{\frac{3x+4ax}{10a}}$	20	2	1	2,5
28	$\frac{e^{-x}+e^{a}}{a+x}$	12	2,5	-1	1
29	$5\sqrt{a+\ln a+\ln x}$	15	6	2	5
30	$\mathrm{tg}ax+\mathrm{tg}^2(a+2{,}5)$	16	2	0	1
31	$\frac{\cos\sqrt{x}+\cos x}{\sqrt{a+1}+x}$	20	3	4	8

ТЕМА 5.

РЕАЛИЗАЦИЯ ЦИКЛИЧЕСКИХ АЛГОРИТМОВ ОПЕРАТОРАМИ *WHILE* И *DO-WHILE*.

Операторы *while* и *do-while* так же, как и оператор for реализуют циклические алгоритмы, то есть алгоритмы, в которых оператор или группа операторов повторяются 2 и более раза. Но в отличие от оператора цикла *for*, операторы *while* и *do-while* целесообразно использовать в тех случаях, когда количество повторений тела цикла заранее не известно.

ОПЕРАТОР ЦИКЛА WHILE.

Называется оператором цикла с предусловием, так как условие выхода проверяется до входа в тело цикло.

Блок-схема оператора цикла *while* выглядит следующим образом:

Рис 5. Блок-схема оператора цикла while

Синтаксис оператора :

while (Условие)

{ Тело цикла; }

Пока Условие равно Истина, тело цикло выполняется, как только условие становится Ложь, происходит выход из цикла. Если Условие сразу будет равно Ложь, то тело цикла не выполнится ни одного раза.

Если тело цикла состоит из одного оператора, то операторные скобки не обязательны, если же тело цикла состоит из 2 и более операторов, то его необходимо заключить в операторные скобки { }, в противном случае, в цикле будет повторяться только один оператор.

Например,

Ввести с клавиатуры целое число К. Вычислить факториал числа К. Результат вывести на экран.

```
#include <stdio.h>
#include <conio.h>
int main ( )
{ int K,i,factor;
```

```
printf("\n Введите целое K");
scanf("%d",&K);
i=factor=1;                    /* инициализация
счетчика цикла и результата*/
while (i<=K)                   /* после оператора
while точка с запятой НЕ ставится!*/
{factor=factor*i;              /* в цикле повторяются
2 оператора, поэтому операторные скобки
обязательны */
i++; }
printf("\n Факториал числа %d=%d",K,factor);
//вывод результата на экран
getch();                       /*задержка экрана для
просмотра результата*/
return 0;
}
```

Обратите внимание на то, что счетчику цикла необходимо присвоить начальное значение (инициировать) до входа в цикл, кроме того, внутри тела цикла обязательно должен присутствовать оператор, изменяющий счетчик цикла.

Оператор цикла do-while.

Называется оператором цикла с постусловием, так как сначала выполняется тело цикла, а затем происходит проверка условия.

Рис 6. Блок-схема оператора do-while

Синтаксис оператора :

```
do
{ Тело цикла; }
while (Условие) ;
```

Цикл do while аналогичен циклу while. Единственное их различие в том, что при выполнении цикла do while один проход цикла будет выполнен независимо от выполнения условия.

Предыдущая задача на вычисление факториала будет выглядеть следующим образом:

```
#include <stdio.h>
#include <conio.h>
int main ( )
{ int K,i,factor;
printf("\n Введите целое К");
scanf("%d",&K);
i=factor=1;
// инициализация счетчика цикла и результата
do
{factor=factor*i;
i++; }
while (i<=K);                    // после оператора
while //точка с запятой ставится!
printf("\n Факториал числа %d=%d",K,factor);
//вывод результата на экран
getch();                         // задержка экрана для
//просмотра результата
return 0;
}
```

При решении различных задач, бывает целесообразно использовать различные виды операторов цикла, но, как правило, операторы цикла взаимозаменяемы.

Лабораторная работа 5.

Написать программу, решающую задачу:

Ввести с клавиатуры символ. Если этот символ-буква *"f"* или *"F",* то вычислить значение y с

помощью цикла *for*, если введена буква *“w”* или “*W”*, то вычислить y, используя оператор *while*, если введена буква “*d*” или *“D*”- оператором *do-while*. Во всех остальных случаях вывести на экран сообщение *“Неверный символ*”.

Все исходные значения вводить с клавиатуры, результат вывести на экран. Выбор типа цикла производить с помощью оператора *switch*.

Варианты заданий

1. y=2.8+$\sum_{i=5}^{N}(sin(3*i)+i)$, при N=12
2. y=2*$\sqrt[3]{a}$+5.3*(N-2)!, при N=10, a=4.6
3. y=cos(x)+$\prod_{i=-3}^{N}(6*i-x)$,при x=0.6, N=3
4. y=3.5*a+$\sum_{k=-2}^{N}(a*k+8.8)$, при a=0.4, N=6
5. y=-6.3*cos(a^4)+(N+a)!, при N=4, a=3
6. y=-12.6*a+$\prod_{k=6}^{N}(2*a*k+a^2)$, при$a=3, N=13$
7. y=|-2*f| +$\sum_{i=-4}^{N}(i*f^2+2.6*f)$, при f=0.5, N=4
8. y=lg(14.3*k)+$\prod_{i=0}^{K}(cos(i^2))$, при K=10
9. y=K*cos(K)+(K-1)!, при K=8
10. y=lg(2*a*|sin(a^2)|)+$\sum_{i=-3}^{K}(|i|+8)$, при a=1.23, K=6
11. y=$\sqrt{N+8}$ *$\prod_{i=5}^{N}(i^2*cos(i))$, при N=12
12. y=$\sqrt[3]{i}$ +cos(i^2) +(i+5)!, при i=9
13. y=$\sqrt[4]{cos(N*C)}$ +$\sum_{i=5}^{N}((i+1)^2+5)$, при C=3, N=12
14. y=lg(|cos(x)|)+$\prod_{i=3}^{P}(i*cos(i))$, при P=4, x=8.6
15. y=26*$cos(\pi)$ (|2) -(z+3)! при z=4
16. y=$\sqrt[2]{18*N}$ + $\sum_{i=4}^{N}(cos(i+1)*3)$, при N=9
17. y=-8.4*b+$\prod_{i=-6}^{N}((i+3)*sin(i))$, при b=8.1, N=4
18. y=$\sqrt{\frac{b}{cos(b)+1.97}}$ + (K+3)!, при b=8.6, K=5
19. y=|6-7.2*a|+ $\sum_{i=4}^{K}(i+\sqrt{6.24*a})$, при a=3.1, K=10
20. y=lg(cos(x)+7.32)+ $\prod_{i=8}^{X}(i*x^2/\sqrt{x})$, при x=12
21. y=$\sqrt[3]{a-3.76}$ -(a+3)! , при a=4
22. y=$(6+a)^2$ + $\sum_{i=4}^{K}((i+a)^2+\sqrt{i})$, при a=2.4, K=8
23. y=$\sqrt{b+\sqrt{b}}$ + $\prod_{i=-3}^{N}(i+b)^2$, при b=5.8, N=3
24. y=cos(a)+$\sqrt{a+6.23}$ - (a-1)! , при a=8

25. y=|a|+$\sqrt{6.03 * a}$ + $\sum_{i=1}^{K}(i^3 + a)$, при a=3.3, K=6

26. y=cos(b^2) + ($\prod_{i=-3}^{N}(i + 1.65)$ |2 + b), при b=1.8, N=4

27. y=$\sqrt[3]{(K + 8)}$ +(K-3)!, при K=8

28. y=$(16.12 - a)^2$ - $\sum_{i=0}^{K}\left((i + 3.51 * a)^2 + \sqrt{i + 1.69}\right)$, при a=3.25, K=5

29. y=lg(|sin($\sqrt{x}$)|)+$\prod_{i=-1}^{P}\left(i * 3.19 - cos(i + x)\right)$, при P=5, x=6.22

30. y= $\sqrt{\frac{\left(\frac{b*K}{2.25}\right)}{5*sin(b)+3.47}}$ + (K+3)!, при b=7.13, K=4

ТЕМА 6.
РАБОТА С ОДНОМЕРНЫМИ МАССИВАМИ.

Массив – это последовательность однотипных элементов, имеющих общее имя. Каждый элемент этой последовательности определяется порядковым номером (индексом) элемента, причем номера элементов идут подряд.

Элементы массива в большинстве языков программирования (в Си в том числе), нумеруются с 0.

A_0, A_1, A_2, …, A_{10}–одномерный массив, состоящий из 11 элементов.

Где *A*-имя массива

0,1…10-номера элементов в массиве (индексы)

A_7-8-й элемент массива

Массивы всегда располагаются в последовательных ячейках памяти.

Элементами массива могут быть, как данные базовых типов : целые, вещественные, символьные, так и данные составных типов. Например, если элементом одномерного массива является другой одномерный массив, то получится двумерный массив (матрица).

Описание одномерного массива в программе на языке С производится в формате:

тип имя[размерность]; например,

int A[9]; описание одномерного массива из 9 элементов целого типа

float MAS[12]; описание одномерного массива из 12 элементов вещественного типа

При описании массива, выделяется память под массив. Например, под массив А выделится 18 байт, так как 1 элемент целого типа занимает 2 байта. Под массив MAS выделится 48 байт (1 элемент вещественного типа занимает 4 байта).

Фактическая размерность массива должна быть не больше, чем указана при описании, то есть массив А может фактически состоять из 1-11 элементов, например, из 7.

Обращение к элементу одномерного массива производится посредством указания имени массива и индекса элемента. (Помним, что элементы массива нумеруются с 0.)

MAS[0]=4.32; -записали число в 1-й элемент массива MAS.

Инициализация одномерных массивов.

Инициализация- заполнение массива значениями, может производиться несколькими способами:

1. Перечисление элементов в фигурных скобках при объявлении:

int M[6]={7,-4,0,2,15,-23};

char S[5]={‘H’,’e’,’l’,’l’,’o’};

при этом значения отделяются друг от друга запятыми.

2. При таком способе инициализации допускается не указывать значение размерности массива, при этом компилятор сам подсчитает количество элементов в списке и выделит под массив необходимый объем памяти. Например:

float MAS[]={4.33, -5.6, 12.788, 0.12};

3. Ввод массива с клавиатуры. Один из самых распространенных способов заполнения массива.

Обработка массива как единого целого не возможна. Массив обрабатывается поэлементно,

поэтому необходима организация цикла. Поскольку количество элементов массива известно, то целесообразно использовать цикл for.

Пример ввода массива с клавиатуры:

```
int M [7];
for (int i=0; i<=6; i++)
{ printf (“\n M[“ %d “]=”,i+1);
  scanf(“%d”, &M[i]); }
```

Поскольку элементы массива в программе нумеруются с “0”, а мы привыкли нумеровать последовательности с “1”, то в операторе printf стоит не i, а i+1.

Кроме того, тело цикла состоит из 2 операторов: *printf* и *scanf*, поэтому их необходимо заключить в фигурные скобки.

4. Заполнение массива, заданного формулой i –го элемента:

Например, массив задан формулой: M_i=i+$\sqrt[2]{i}$

```
float M [7];
        for (int i=0; i<=6; i++)
M[i]=i+pow(i, 0.5);
```

Обратите внимание, что в этом примере помимо стандартной библиотеки ввода-вывода stdio.h, необходимо подключить библиотеку математических функций math.h.

5. Заполнение массива случайными числами:

srand(time(0)); /* функция srand вызывается один раз в программе до входа в цикл и нужна для того, чтобы каждый раз генерировалась новая

последовательность случайных чисел. Если ее не использовать, то при каждом запуске программы набор случайных чисел будет один и тот же.*/

```
for (int i=0; i<=6; i++)
M[i]= rand() % 101+ 50; // генерирует случайное
//число в диапазоне 50-100
```

В данном примере, помимо стандартной библиотеки ввода-вывода *stdio.h,* необходимо подключить библиотеку *stdlib.h*, которая работает с функциями случайных чисел.

Пример программы обработки одномерного массива:

Ввести с клавиатуры одномерный массив из 10 вещественных чисел. Вычислить максимальный и минимальный элементы массива и поменять их местами. Вычисленные значения, а также полученный после замены массив вывести на экран.

```
include <stdio.h>
include <conio.h>
int main()
{ float M[10], min, max, p;
int  nmin, nmax;
for (int i=0; i<10; i++) // ввод массива с
//клавиатуры
{ printf (“\n M[“ %d “]=”,i+1);
 scanf(“%f”, &M[i]); }
min=max=M[0];
//вычисление максимального и минимального эл-тов
nmin=nmax=0;
for (i=0; i<10; i++)
 if M[i]>max
      {max=M[i];
   nmax=i;}
else
if M[i]<min
      {min=M[i];
   nmin=i;}
printf(“\n min=%f max=%f”,min,max);
```

```
// вывод на экран значений min и max
p=min;
    // замена min и max местами с исполь-
//зованием вспомогательной переменной p
M[nmin]=max;
M[nmax]=p;
printf("\nПолученный массив :");      // вывод на
//экран полученного массива
for (i=0;i<10;i++)
printf("\n M[%d]=%8.3f",i+1, M[i]);
getch();
return 0;
}
```

Лабораторная работа 6.

Постановка задачи: ввести с клавиатуры целое число К (от 6 до 15). Ввести с клавиатуры массив из К элементов целого типа. Выполнить над массивом действия, согласно номера варианта. Результаты вывести на экран.

Варианты заданий к лабораторной работе 6.

1. Вычислить кол-во и сумму отрицательных эл-тов массива и поставить эти числа на первое и второе места в массиве.
2. Вычислить среднее арифметическое четных эл-тов массива и поставить это число на последнее место в массиве.
3. Вычислить среднее арифметическое эл-тов, имеющих четные номера и заменить этим числом первый эл-т массива.
4. Найти минимальный эл-т массива и поменять его местами со 2-м эл-том.
5. В массиве заменить все четные значения числом М, а все нечетные -числом Р, где М-произведение всех положительных эл-тов, а Р-сумма отрицательных эл-тов.

6. В массиве заменить все отрицательные значения числом М, а все положительные -числом Р, где М- произведение всех четных эл-тов, а Р- количество нечетных эл-тов.

7. Вычислить сумму всех положительных эл-тов массива и произведение отрицательных эл-тов и заменить этими числами первый и последний эл-ты массива.

8. Вычислить кол-во и сумму положительных эл-тов массива и поставить эти числа на третье и последнее места в массиве соответственно.

9. Вычислить произведение максимального и минимального эл-тов массива и поставить это число на второе место в массиве.

10. Вычислить среднее арифметическое нечетных эл-тов массива и поставить это число на первое место в массиве.

11. Вычислить минимальный и максимальный эл-ты массива и заменить все эл-ты, имеющие четные номера значением минимума, а все эл-ты, имеющие нечетные номера- значением максимума.

12. Вычислить минимальный и максимальный эл-ты массива и заменить все эл-ты, имеющие четные значения- значением минимума, а все эл-ты, имеющие нечетные значения- значением максимума.

13. Вычислить произведение и сумму отрицательных эл-тов массива и поставить эти числа на первое и последнее места в массиве соответственно.

14. Вычислить среднее арифметическое отрицательных эл-тов массива и заменить этим числом все эл-ты = 0.

15. В массиве заменить все положительные значения числом М, а все отрицательные -числом Р, где М- произведение всех положительных эл-тов, а Р-сумма отрицательных эл-тов.

16. Вычислить среднее арифметическое четных эл-тов массива и заменить этим числом все эл-ты, значения которых > 8.

17. Вычислить среднее арифметическое положительных эл-тов массива и заменить этим числом все эл-ты, значения которых <=-3.

18. Вычислить сумму и количество эл-тов массива, лежащих в диапазоне [-5;5] и заменить этими значениями первый и второй эл-ты массива.

19. Вычислить среднее арифметическое эл-тов массива, лежащих в диапазоне [-10;5] и заменить этим значением последний эл-т массива.

20. Вычислить минимальный эл-т массива и заменить этим числом все эл-ты, значения которых >10.

21. Вычислить максимальный эл-т массива и заменить этим числом все эл-ты, значения которых =0.

22. В массиве заменить все нулевые значения числом М, где М- среднее арифметическое всех положительных эл-тов.

23. В массиве заменить все положительные значения числом Р, где Р- среднее арифметическое всех отрицательных эл-тов.

24. Вычислить кол-во и произведение эл-тов массива, лежащих в диапазоне

[-10;5] и заменить этими значениями первый и третий эл-ты массива.

25. Вычислить сумму максимального и минимального элементов массива и заменить этим числом третий элемент массива.

26. Вычислить разность максимального и минимального элементов массива и заменить этим числом все положительные элементы массива.

27. Вычислить сумму и произведение положительных элементов массива и поставить эти числа на первое и последнее места в массиве соответственно.

28. Вычислить произведение отрицательных элементов массива и заменить этим числом все положительные элементы массива.

29. Вычислить сумму и количество элементов, принадлежащих диапазону

[-4;18] и поставить эти числа на первое и последнее места соответственно.

30. Вычислить количество нулевых элементов массива и заменить этим значением все четные элементы.

ТЕМА 7.

ФУНКЦИЯ.

Функция-это минимальный исполняемый модуль программы на языке С. В других языках это называется программа, процедура, функция.

Имя функции- это особый тип указателя, называемый указателем на функцию, значением которого является адрес точки входа в функцию.

Для работы с функцией необходимо:

объявить(задекларировать) функцию;

определить(описать) функцию;

вызвать функцию.

Объявление(декларация) функции.

```
<тип><имя _функции>[(список формальных параметров)]
```

Объявление функции- это ее заголовок , которых называют *прототипом.* Обычно прототипы функций помещают в заголовочный файл, подключаемый директивой *#include* к тексту программы.

Чаще всего прототип функции полностью совпадает с заголовком в описании функции, но не всегда.

Определение(описание) функции(стандарт ANSI).

```
<тип><имя функции>([список формальных параметров])
{Тело функции
```

```
}
```

Имя функции *main* для основной функции, либо любое, по желанию пользователя, все остальные функции программы имеют отличные от *main* имена,

но они не должны совпадать с зарезервированными(служебными) словами.

Внутри функции нельзя определить новую функцию.

Функция обязательно имеет оператор *return*[i]- оператор возврата из функции в точку вызова в программе.

Оператор *return* имеет две формы: *return, return(*выражение*)*.Оператор *return* используется в функции не возвращающей никакое значение, т.е. типа *void*, Если функция имеет определенный тип *float*() , int z(), то она возвращает в программу значение соответствующего типа и в теле функции обязательно *return*(выражение)[ii], причем тип выражения должен совпадать с типом функции.

Пример.

$$y(x)=\begin{cases}1, \text{если} x>0\\ 0, \text{если} x\le 0\end{cases}$$

```
int f (float x)
{int y;
if(x>0) y=1; else y=0;
return y;
}
```

Если функция возвращает значение определенного типа (int,float и т.д.), то имя функции с фактическими параметрами подставляется в выражение в месте вызова этой функции.

Вызов функции может находиться в любом месте программы, где по синтаксису допустимо выражение того типа, которое формирует и возвращает функция.

Обмен информацией между функциями

Связи между функциями осуществляются;

-с помощью глобальных переменных;
-через возвращаемо значение;
-с помощью передачи параметров.
Определение любой функции имеет вид:

Глобальные переменные видны во всех функциях, где не описаны локальные переменные , имеющие те же имена.

Возвращаемые значения реализуются оператором *return.*

Программа состоит из функций , описаний и директив препроцессора . Одна из функций является главной. Имя главной функции main(). Выполнение программы начинается с первого оператора этой функции. Завершение работы этой функции означает завершение работы всей программы. В любой программе всегда имеется функция с именем main(). Она может быть единственной, если программа состоит из одной функции. Все остальные функции программы имеют отличные от main имена.

<тип><имя _функции>([параметры]) {операторы тела функции}

Параметры записываются в круглые скобки, наличие квадратных скобок не обязательно. Квадратные скобки означают, что внутри круглых скобок параметры могут быть, но могут и не быть.

Замечание 1. Если функция возвращает в точку своего вызова некоторое значение, но тип возвращаемого значения не указан, то по умолчанию считается, что этот тип int.

Замечание 2. Если функция не возвращает значение, то тип void (пустой).

Замечание 3. Тело функции – это блок, который заключается в фигурные скобки .

Замечание 4. Функции не могут быть вложенными.

Замечание 5. Операторы функции заканчиваются точкой с запятой.

Структура программы

Директивы препроцессора описания
int main()
{ операторы главной функции
}
int f1( )
{ операторы функции f1
}
int f2( ) { операторы функции f2

} и т.д.

Функции создаются в виде файлов, из которых потом образуется модуль.

Так как программа всегда использует в своей работе стандартные функции , то компилятору необходимо знать, где искать эти функции , поэтому любая программа , в которой содержатся вызовы стандартных функций , должна начинаться с директивы препроцессора:

```
#include<имя заголовочного файла>
```

Директива #include(включить файл) может стоять в любом месте программы, но обычно программа с нее начинается. Когда заголовочный файл будет найден, строка #include заменяется на все содержимое этого заголовочного файла.

Каждая библиотечная функция, определенная стандартом языка C(C++) , имеет прототип в соответствующем заголовочном файле.

Прототип – это заголовок функции, по которому будет проверена правильность вызова этой функции, и найдено ее определение в библиотеке.

Если функция возвращает значение определенного типа (int, float и т.д.), то имя функции с фактическими параметрами подставляется в выражение в месте вызова этой

функции Вызов функции может находиться в любом месте программы, где по синтаксису допустимо выражение того типа, которое формирует и возвращает функция.

Обмен информацией между функциями

Связи между функциями осуществляются;

-с помощью глобальных переменных ;
-через возвращаемое значение;
-с помощью передачи параметров.

Глобальные переменные видны во всех функциях, где не описаны локальные переменные , имеющие те же имена.

Возвращаемые значения реализуются оператором *return.*

Все величины, описанные внутри функции, являются локальными. Областью их действия является функция. При вызове функции в специальной области памяти, называемой *стеком,* выделяется память под *локальные автоматические переменные*. Большинство переменных являются автоматическими , т.е. они относятся к классу *auto.*Стек- время жизни функции. При выходе из функции стек освобождается и готов к приему значений переменных следующей функции. При выходе из функции соответствующий участок стека освобождается , поэтому значения локальных переменных между вызовами одной и той же функции не сохраняются.

Каждая программа имеет только одну функцию с именем *main*. С этой функции начинается исполнение программы. Другие функции могут быть вызваны из функции *main* или из любой другой функции в процессе исполнения программы. Каждая функция может иметь 0 и более параметров. Параметры являются переменными, которые используются для передачи данных между функциями .Имена параметров в вызываемой и вызывающей функции независимы.

Передача параметров.

Механизм передачи параметров- это основной способ обмена информацией между функциями.

Параметры функции при описании называются формальными параметрами. При вызове используются фактические параметры (аргументы), которые потом становятся на место формальных параметров .

Существует два способа передачи параметров в функции:

-*по значению* в стек заносятся копии значений аргументов, и операторы функции работают с этими копиями ; доступа к исходным значениям функция не имеет, а значит, нет возможности их изменения;

-*адресу* (с помощью указателя и по ссылке) в стек заносятся копии адресов аргументов, а функция обращается к сегменту данных по этим адресам и может изменить исходные значения аргументов, так как имеет к ним доступ.

При использовании массива в качестве параметра в функцию передается указатель на его первый элемент, то есть массив всегда передается по адресу.

Но адрес указывает только на начало массива и ничего не говорит о его размере, поэтому *размерность массива надо передавать через отдельный параметр, например:*

float funct(float arr[], const int n); или

float funct(float *arr, const n);

И та и другая форма записи приведут к тому же результату .В обоих случаях происходит передача адреса начала массива и количество его элементов.

Выводы:

1.*Функция*-это логически самостоятельная именованная часть программы.

2.*Интерфейс функции* определяется ее заголовком.

3.Для *вызова* функции надо указать ее имя и список фактических параметров.

4.В *определении, объявлении и вызове функции* типы и порядок следования параметров должны совпадать.

5.Передача параметров в функцию может выполняться по значению и по адресу.

6.*Входные данные* функции надо передать по значению или по ссылке, а результаты работы- через возвращаемое значение .

6.*Массивы* всегда передаются в функцию по адресу.

---

[i] tu

[ii]