

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет им. Г.И. Носова»

Многопрофильный колледж



**МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ
ПРАКТИЧЕСКИХ ЗАНЯТИЙ**

по учебной дисциплине

Основы алгоритмизации и программирования

для студентов специальности

**09.02.01 Компьютерные системы и комплексы
базовой подготовки**

Магнитогорск, 2017

ОДОБРЕНО

Предметно-цикловой комиссией
Информатика и вычислительная техника
Председатель И.Г.Зорина
Протокол № 7 от 14 марта 2017

Методической комиссией МпК
Протокол №4 от «23» марта 2017г

Составитель (и):

преподаватель ФГБОУ ВО МГТУ МпК, к.т.н., доцент
преподаватель ФГБОУ ВО МГТУ МпК, к.п.н.

В.Д.Тутарова
Ю.В.Федосеева

Методические указания по выполнению практических занятий разработаны на основе рабочей программы учебной дисциплины «Основы алгоритмизации и программирования».

Содержание практических работ ориентировано на подготовку студентов к освоению профессионального модуля основной профессиональной образовательной программы по специальности 09.02.01 «Компьютерные системы и комплексы» и овладению профессиональными компетенциями.

СОДЕРЖАНИЕ

1 Введение	4
2 Методические указания	7
Практическое занятие 1,2	7
Практическое занятие 3	12
Практическое занятие 4,5	17
Практическое занятие 6,7,8	20
Практическое занятие 9,10,11	23
Практическое занятие 12,13,14	26
Практическое занятие 15,16	31
Практическое занятие 17,18	34
Практическое занятие 19,20	42
Практическое занятие 21,22,23,24	47

1 ВВЕДЕНИЕ

Важную часть теоретической и профессиональной практической подготовки студентов составляют практические занятия и лабораторные работы. Являясь частью изучения учебной дисциплины, они призваны, экспериментально подтвердить теоретические положения и формировать общие и профессиональные компетенции, практические умения.

Ведущей дидактической целью *практических занятий* является формирование практических умений - профессиональных (умений выполнять определенные действия, операции, необходимые в последующем в профессиональной деятельности) или учебных (умений решать задачи по математике, физике, химии, информатике и др.), необходимых в последующей учебной деятельности по общим гуманитарным и социально-экономическим дисциплинам, математическим и естественнонаучным, общепрофессиональным дисциплинам.

Состав и содержание практических и лабораторных работ направлены на реализацию действующих федеральных государственных образовательных стандартов среднего профессионального образования.

В соответствии с рабочей программой учебной дисциплины «Основы алгоритмизации и программирования» предусмотрено проведение практических занятий.

В результате их выполнения, обучающийся должен:

уметь:

- формализовать поставленную задачу;
- применять полученные знания к различным предметным областям;
- составлять и оформлять программы на языках программирования;
- тестировать и отлаживать программы;

Содержание практических занятий ориентировано на подготовку студентов к освоению профессионального модуля основной профессиональной образовательной программы по специальности и овладению профессиональными компетенциями:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем.

ПК 2.2. Производить тестирование, определение параметров и отладку микропроцессорных систем.

ПК 3.3. Осуществлять установку и конфигурирование персональных компьютеров и подключение периферийных устройств.

В процессе освоения дисциплины у студентов должны формироваться общие компетенции:

ОК 1. Понимать сущность и социальную значимость своей

будущей профессии, проявлять к ней устойчивый интерес.

ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.

ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.

ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

ОК 5. Использовать информационно-коммуникационные технологии в профессиональной деятельности.

ОК 6. Работать в коллективе и команде, эффективно общаться с коллегами, руководством, потребителями.

ОК 7. Брать на себя ответственность за работу членов команды (подчиненных), результат выполнения заданий.

ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации.

ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности.

Выполнение студентами *практических работ* по учебной дисциплине «Основы алгоритмизации и программирования» направлено на:

- обобщение, систематизацию, углубление, закрепление, развитие и детализацию полученных теоретических знаний по конкретным темам учебной дисциплины;

- формирование умений применять полученные знания на практике, реализацию единства интеллектуальной и практической деятельности;

- формирование и развитие умений: наблюдать, сравнивать, сопоставлять, анализировать, делать выводы и обобщения, самостоятельно вести исследования, пользоваться различными приемами измерений, оформлять результаты в виде таблиц, схем, графиков;

- приобретение навыков работы с различными приборами, аппаратурой, установками и другими техническими средствами для проведения опытов;

- развитие интеллектуальных умений у будущих специалистов: аналитических, проектировочных, конструктивных и др.;

- выработку при решении поставленных задач профессионально значимых качеств, таких как самостоятельность, ответственность, точность, творческая инициатива.

Продолжительность выполнения практической работы составляет не менее двух академических часов и проводится после соответствующей темы, которая обеспечивает наличие знаний, необходимых для ее выполнения.

Требования к отчету практической работы.

Студент должен:

- показать на примерах правильность результатов, полученных с помощью разработанной программы.

- при необходимости кратко сформулировать алгоритм работы программы и выделить в программе основные блоки, ответственные за выполнение определенных действий (например, ввод исходных данных, объявления переменных, вывод на экран и т.д.).

- знать и при необходимости объяснить работу всех использованных в программе языковых конструкций.

- знать и при необходимости объяснить назначение и смысл всех использованных констант и переменных.

2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Тема 1.2. Основные алгоритмические конструкции: линейные, разветвляющиеся, циклические.

Практическое занятие № 1,2

Построение блок схем основных алгоритмических конструкций.

Формируемая компетенция:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем.

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

Цель работы: получить навыки разработки алгоритмов решения задач различной сложности.

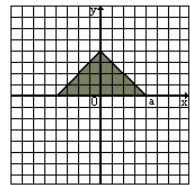
Выполнив работу, Вы будете:

уметь:

- разрабатывать алгоритмы для конкретных задач;

Задание:

1. Запишите произвольную функцию, например $f(x, y) = 7x^2 + y^4$
2. Составьте алгоритм вычисления значения функции для произвольных значений аргументов.
3. Постройте произвольный график (не менее двух кривых), например, как показано на рисунке.



4. Запишите алгоритм, в результате выполнения которого будет выведено значение ИСТИНА, если точка с заданными координатами (x,y) лежит внутри заштрихованной области и ЛОЖЬ в противном случае
5. Составьте блок схему алгоритма соответствующего варианта.

Блок схема должна содержать:

- начало;
- ввод исходных данных;
- блок решение;
- вывод результатов;
- конец.

Вариант	Задача 1 Алгоритм вычисления	Задача 2
1.	температуры из градусов по шкале Цельсия (С) в градусы шкалы Фаренгейта	Начав тренировки, спортсмен в первый день пробежал 10км. Каждой следующий день он

	(F) для значений от 15°C до 30°C с шагом 1°C (Перевод осуществляется по формуле $F=(1,8C+32)$).	увеличивал дневную норму на 10% от -нормы предыдущего дня. Какой суммарный путь пробежит спортсмен за 7 дней?
2.	веса из фунтов в кг для значений от 1 до 10 фунтов с шагом 1 фунт (1фунт =400г).	Написать алгоритм, который бы по введенному номеру времени года (1 — зима, 2 — весна, 3 — лето, 4 — осень) выдавал соответствующие этому времени года месяцы, количество дней в каждом из месяцев.
3.	перевода расстояний в дюймах в сантиметры (1 дюйм=2,54см) для значений от 1 до 10 дюймов с шагом 1.	Для целого числа k от 1 до 99 напечатать фразу «Мне k лет», учитывая при этом, что при некоторых значениях k слово «лет» надо заменить на слово «год» или «года». Например, 11 лет, 22 года, 51 год.
4.	перевода расстояний в км для значений от 8000 до 9000 м с шагом 10 метров.	Написать алгоритм решения уравнения $ax^3 + bx = 0$ для произвольных a, b.
5.	температуры из градусов по шкале Цельсия (C) в градусы шкалы Фаренгейта (F) для значений от 15°C до 30°C с шагом 1°C (Перевод осуществляется по формуле $F=(1,8C+32)$).	Даны две точки A(x1, y1) и B(x2, y2). Составить алгоритм, определяющий, которая из точек находится ближе к началу координат.
6.	перевода расстояний в м для значений от 10 до 100 см с шагом 10 см.	Написать алгоритм, который по вводимому числу от 1 до 5 (номеру курса) выдает соответствующее сообщение «Привет, k-курсник». Например, если k=1, «Привет, первокурсник»; при k=4: «Привет, четверокурсник».
7.	веса из тонны в кг для значений от 1 до 10 т с шагом 1 т (1т =1000кг).	Дано трехзначное число N. Проверить, будет ли сумма его цифр четным числом.
8.	веса из фунтов в кг для значений от 1 до 10 фунтов с шагом 1 фунт (1фунт =400г).	Записать заданное трехзначное число N в обратном порядке

9.	перевода расстояний в дюймах в сантиметры (1 дюйм-2,54см) для значений от 1 до 10 дюймов с шагом 1.	Дано трехзначное число N. Проверить, будет ли произведение его цифр четным числом.
10.	перевода расстояний в м для значений от 10 до 100 см с шагом 10 см.	Дано трехзначное число N. Проверить, будет ли сумма его цифр нечетным числом.

Краткие теоретические сведения:



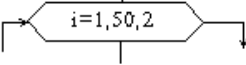
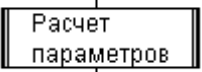
Блок-схемой называют графическое представление алгоритма, в котором он изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий.


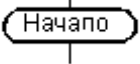
В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий.

Приведем в табл. 1 наиболее часто употребляемые символы.

Таблица 1

Представление элементов блок схемы

Название символа	Обозначение и пример заполнения	Пояснение
Процесс		Применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных.
Решение		Используется для обозначения переходов управления по условию.
Модификация		Используется для организации циклических конструкций. (Слово модификация означает видоизменение, преобразование).
Предопределенный процесс		Используется для указания обращений к вспомогательным алгоритмам, существующим автономно в виде некоторых самостоятельных модулей, и для

		обращений к библиотечным подпрограммам.
Ввод-вывод		Ввод-вывод в общем виде
Пуск-останов		Начало, конец алгоритма, вход и выход в подпрограмму

Пример Составьте блок-схему алгоритма определения высот h_a , h_b , h_c треугольника со сторонами a , b , c , если

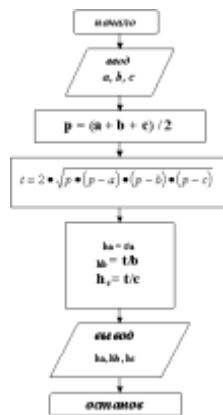
$$h_a = \left(\frac{2}{a}\right) \cdot \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$$

$$h_b = \left(\frac{2}{b}\right) \cdot \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$$

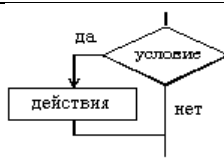
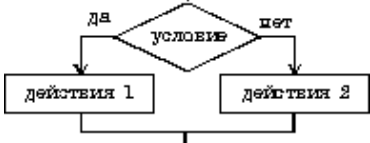
$$h_c = \left(\frac{2}{c}\right) \cdot \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$$

Решение. Введем обозначение

$t = 2 \cdot \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$ тогда $h_a = t/a$, $h_b = t/b$, $h_c = t/c$. Блок-схема должна содержать начало, ввод a , b , c , вычисление p , t , h_a , h_b , h_c , вывод результатов и остановку.



Виды структуры ветвления

Структура	Блок-схема
если-то если условие то действия конец если	
если-то-иначе если условие то действия 1 иначе действия 2 конец если	

<p>Выбор</p> <p>выбор при условии 1: действия 1 при условии 2: действия 2 при условии N: действия N конец выбора</p>	
<p>Выбор-иначе</p> <p>выбор при условии 1: действия 1 при условии 2: действия 2 при условии N: действия N иначе действия N+1 конец выбора</p>	

Виды структуры цикла

Структура	Блок-схема
<p>Цикл типа для.</p>	
<p>Цикл типа пока.</p>	

Порядок выполнения работы:

1. Изучить теоретический материал по теме.
2. Разработать алгоритм решения в соответствии с заданием.
3. Определить результаты работы алгоритма для контрольных значений.

Форма представления результата:

Отчет по работе должен содержать:

- а) наименование работы и цель работы;
- б) формулировку задачи;
- в) блок схему;
- г) результаты работы алгоритма.

Тема 2.2. Ввод и вывод данных

Практическое занятие № 3

Операции ввода - вывода

Формируемая компетенция:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем

Цель работы: научиться работать в среде программирования Borland C++ Builder и получить навыки составления и отладки простейших программ на C++.

Выполнив работу, Вы будете:

уметь:

- применять стандартные функции в программном коде для решений задач.

Задание:

1. Составить программу, которая бы спрашивала у пользователя ФИО, возраст, образование и организовывала вывод этой информации следующим образом:

Ваше ФИО: _____

Ваш возраст: _____

Образование: _____

Телефон: _____

2. Разработать алгоритм решения в соответствии с заданием 1,2 по вариантам.
3. Записать программу на языке программирования C++, где ввод и вывод осуществить двумя способами (консольный и поточный).
4. Протестировать программу для контрольных значений.

Задание 1. Напишите программу, рассчитывающую значение заданной функции и выводящую его на экран. Значения аргументов должны вводиться с клавиатуры.

а) $z(x, y) = e^{-x^2/2 - y^2/2}$;	б) $y(x) = \sin^5\left(\frac{2x^2}{3}\right)$;
в) $y(x) = \arctg\left(\frac{x-1}{x+1}\right)^2$;	г) $z(x, y) = \frac{1}{\sqrt{2\pi}} e^{-x/y}$;

д) $z(x, y) = \left \frac{y}{1+x^2} \right ;$	е) $y(x, n) = \left(\frac{\sqrt{x^3+1}}{5x} \right)^n;$
ж) $z(x, y) = \frac{1}{1-e^{-x}} \ln y;$	з) $y(x) = \sqrt{1 + \frac{x^2}{2} + \frac{x^3}{3}};$
и) $y(x) = \left \frac{1-2e^{-x}}{1+2e^{-x}} \right ;$	к) $z(x, y, k) = \frac{\cos(kx)}{\sqrt{ x+y^k }}.$

Задание 2.

1. Напишите программу, запрашивающую у пользователя радиус круга и выводящую на экран диаметр, длину окружности и площадь этого круга.

2. Напишите программу, запрашивающую у пользователя трехзначное число и определяющее из каких цифр оно состоит.

3. Напишите программу, запрашивающую длину основания и высоту равнобедренного треугольника и вычисляющую длины всех его сторон.

4. Напишите программу, вычисляющую объем и площадь поверхности шара по заданному с клавиатуры значению радиуса.

5. Напишите программу, вычисляющую радиус окружности, вписанной в правильный n -угольник, по известной длине его стороны a . Значения n и a вводятся с клавиатуры.

6. Напишите программу, вычисляющую радиус окружности, описанной вокруг правильного n -угольника, по известной длине его стороны a . Значения n и a вводятся с клавиатуры.

7. Даны координаты трех точек на плоскости $A(x_1, y_1)$, $B(x_2, y_2)$ и $C(x_3, y_3)$. Напишите программу, вычисляющую величину угла ABC (в градусах).

8. Напишите программу, вычисляющую объем параллелепипеда.

9. Напишите программу, вычисляющую площадь треугольника, зная координаты трех его вершин.

10. Напишите программу, запрашивающую у пользователя трехзначное число и определяющее из каких цифр оно состоит.

Краткие теоретические сведения:

Решение задач по программированию предполагает ряд этапов:

1) Разработка математической модели. На этом этапе определяются исходные данные и результаты решения задачи, а также математические формулы, с помощью которых можно перейти от исходных данных к конечному результату.

2) Разработка алгоритма. Определяются действия, выполняя которые можно будет от исходных данных придти к требуемому результату.

3) Запись программы на языке программирования. На этом этапе каждому шагу алгоритма ставится в соответствие конструкция выбранного алгоритмического языка.

4) Выполнение программы (исходный модуль ->компилятор ->объектный модуль -> компоновщик -> исполняемый модуль)

5) Тестирование и отладка программы. При выполнении программы могут возникнуть ошибки 3 типов:

- a. синтаксические – исправляются на этапе компиляции;
- b. ошибки исполнения программы (деление на 0, логарифм от отрицательного числа и т. п.) – исправляются при выполнении программы;
- c. семантические (логические) ошибки – появляются из-за неправильно понятой задачи, неправильно составленного алгоритма.

Чтобы устранить эти ошибки программа должна быть выполнена на некотором наборе тестов. Цель процесса тестирования – определение наличия ошибки, нахождение места ошибки, ее причины и соответствующие изменения программы – исправление. Тест – это набор исходных данных, для которых заранее известен результат. Тест выявивший ошибку считается успешным. Отладка программы заканчивается, когда достаточное количество тестов выполнилось неуспешно, т. е. программа на них выдала правильные результаты.

В зависимости от требований задания программист выбирает тип для объектов программы. Типы Си++ можно разделить на простые и составные. К простым типам относят типы, которые характеризуются одним значением. В Си++ определено 6 простых типов данных:

int (целый)	}	целочисленные
char (символьный)		
wchar_t (расширенный символьный)		
bool (логический)		
float(вещественный)		
double (вещественный с двойной точностью)	- с плавающей точкой (число=мантисса x 10 ^k)	

Существует 4 спецификатора типа, уточняющих внутреннее представление и диапазон стандартных типов

short (короткий)
long (длинный)
signed (знаковый)
unsigned (беззнаковый)

Консольный ввод-вывод организуется с помощью стандартных библиотек `stdio.h` и `conio.h`, что предполагает наличие директив `#include <stdio.h>` и/или `#include <conio.h>` в заголовочной части программы.

Функция **printf** используется для вывода информации на экран. С ее помощью в окне приложения можно распечатать как строку простого текста, так и значения переменных различных типов. Общая форма записи соответствующего оператора `printf("форматная_строка"[, перем1][, перем2][, ...]);`

Консольный ввод может быть организован с помощью функции `scanf`. Вводимая при этом информация помещается в некоторую переменную. Общая форма записи этого оператора

```
scanf("форматная_строка", &перем1[, &перем2][, ...]);
```

Пример. Напишите программу, выводящую на экран символы звездочки «*» в форме ромба, как это показано на рисунке справа.

Решение. Задача может быть решена несколькими способами. Ниже приводится один из вариантов.

```
//----- Консольный вывод -----
```

```
#pragma hdrstop
#include <stdio.h> // Подключаем библиотеки
#include <conio.h> // консольного ввода-вывода
#pragma argsused
int main(int argc, char* argv[])
{
    printf(" * \n"); // В программе используем несколько
    printf(" * * \n"); // последовательных вызовов функции
    printf(" * * * \n"); // printf для вывода на экран
    printf(" * * * * \n"); // отдельных строк рисунка.
    printf(" * * * * \n"); // Каждую строку заканчиваем
    printf(" * * * * \n"); // управляющим символом '\n'.
    printf(" * * * * \n"); //
    getch(); // Задержка до нажатия любой клавиши
    return 0; // Завершение программы
}
```

```
      *
     * *
    * * *
   * * * *
  * * * *
 * * * *
*
```

Потоковый ввод-вывод организуется с помощью библиотеки `iostream.h`, что предполагает наличие директивы `#include <iostream.h>` в заголовочной части программы. В библиотеке определены два потоковых объекта с именами `cin` и `cout`, которые связаны с клавиатурой и экраном компьютера соответственно.

Примеры:

```
cout << index;           // выводит на экран значение index
cin >> x;                // вводит значение x с клавиатуры
cout << "Enter your name"; // выводит текстовую строку
```

```
system("chcp 1251>>NULL"); // подключение русских символов
cout << "Имя, " << name << endl; // вводит сообщение Hello, затем
переменную name с переходом на новую строку
```

Язык C++ дает удобные возможности использования математических функций. Большая их часть содержится в библиотеке math.h, и для пользования ими требуется соответствующая директива #include <math.h>. Наиболее употребительные математические функции приведены в таблице

<i>название функции</i>	<i>обозначение</i>	<i>запись в C/C++</i>
синус	$\sin(x)$	sin(x)
косинус	$\cos(x)$	cos(x)
тангенс	$\operatorname{tg}(x)$	tan(x)
квадратный корень	\sqrt{x}	sqrt(x)
возведение в степень	x^y	pow(x,y)
экспонента	e^x	exp(x)
натуральный логарифм	$\ln x$	log(x)
модуль	$ x $	fabs(x)
арксинус	$\arcsin(x)$	asin(x)
арккосинус	$\arccos(x)$	acos(x)
арктангенс	$\operatorname{arctg}(x)$	atan(x)

Порядок выполнения работы:

4. Изучить теоретический материал по теме.
5. Разработать алгоритм решения в соответствии с заданием.
6. Составить программный код.
7. Протестировать программу.

Форма представления результата:

Отчет по работе должен содержать:

- а) наименование работы и цель работы;
- б) формулировку задачи;
- в) блок схему или программный код;
- г) результаты работы программы.

Контрольные вопросы

1. Назовите стандартные объекты потокового, консольного ввода-вывода.
2. Что такое переменная? Чем объявление переменной отличается от ее определения? Привести примеры определений и объявлений.

3. Что такое класс памяти? Какие классы памяти существуют в C++? Привести примеры объявлений и определений переменных разных классов памяти.

4. К какому типу относятся константы 192345, 0x56, 0xCB, 016, 0.7865, .0045, 'c', "x", one, "one", 5, 5.?

5. Что такое тип данных?

6. Чем отличаются типы данных: float и double, char и wchar_t, int и short int?

7. С какой целью используются комментарии в программе, при помощи каких знаков они вводятся?

8. Как объявляются и инициализируются переменные?

9. Назовите все элементы оператора присваивания.

10. Что имеет больший приоритет сложение и вычитание или умножение и деление?

11. Как следует поступить в том случае, когда вы не помните приоритет отдельных операций?

Тема 2.3. Базовые конструкции языков программирования Практическое занятие № 4,5

Оператор условия

Формируемая компетенция:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем

Цель работы: овладение практическими навыками разработки и программирования вычислительного процесса разветвляющейся структур.

Выполнив работу, Вы будете:

уметь:

– применять структуры условия в программном коде для решений задач.

Задание:

1. Составить программный код для алгоритма задачи №2 практическая работа №1.

2. Протестировать программу для контрольных значений.

3. Разработать алгоритм решения в соответствии с заданием по вариантам.

4. Записать программу на языке программирования C++

5. Протестировать программу для контрольных значений.

Варианты задач:

1. Даны три действительные числа. Возвести в квадрат те из них, значения которых неотрицательны, и в четвертую степень — отрицательные.

2. Даны две точки $A(x_1, y_1)$ и $B(x_2, y_2)$. Составить алгоритм, определяющий, которая из точек находится ближе к началу координат.
3. Даны два угла треугольника (в градусах). Определить, существует ли такой треугольник. Если да, то будет ли он прямоугольным.
4. Даны действительные числа x и y , не равные друг другу. Меньшее из этих двух чисел заменить половиной их суммы, а большее — их удвоенным произведением.
5. На плоскости XOY задана своими координатами точка A . Указать, где она расположена: на какой оси или в каком координатном угле.
6. Даны целые числа m, n . Если числа не равны, то заменить каждое из них одним и тем же числом, равным большему из исходных, а если равны, то заменить числа нулями.
7. Написать программу нахождения суммы большего и меньшего из 3 чисел.
8. Даны три числа a, b, c . Определить, какое из них равно d . Если ни одно не равно d , то найти $\max\{d-a, d-b, d-c\}$.
9. Даны два действительных положительных числа x и y . Арифметические действия над числами пронумерованы (1 – сложение, 2 – вычитание, 3 – умножение, 4 – деление). Составить программу, которая по введенному номеру выполняет то или иное действие над числами.
10. По введенному году рождения определить название года по японскому календарю и вывести на экран.

Краткие теоретические сведения:

Оператор условия `if-else` служит для выбора направления работы программы в зависимости от условий, сложившихся в данной точке программы на момент ее выполнения.

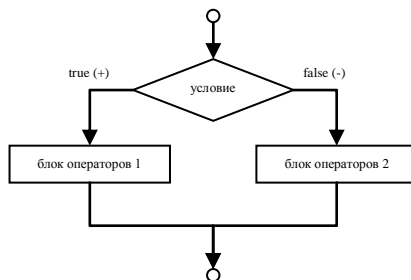
Общая форма записи условного оператора

```

if(<условие>)
{
  <блок операторов 1>;
}
else
{
  <блок операторов 2>;
}

```

Если на момент выполнения `<условие>` истинно, программа передает управление `<блоку операторов 1>` и, далее, первому оператору за пределами конструкции `if-else`. При этом `<блок операторов 2>` не



выполняется. Иначе, если <условие> ложно, выполняется <блок операторов 2>, а <блок операторов 1> пропускается.

Фигурные скобки в синтаксисе оператора if-else используются для выделения в тексте блоков 1 и 2. Старайтесь располагать закрывающую скобку под открывающей для улучшения читаемости программного кода. Для этой же цели текст внутри фигурных скобок необходимо сместить вправо на несколько позиций.

Условие в операторе if-else может быть выражено не только в виде простого сравнения двух числовых значений. Например, весьма распространены двойные условия, которые в математике записываются в виде «a < b < c». Запись означает, что значение b лежит в диапазоне между значениями a и c. В программе такие условия должны быть переформулированы с использованием простых операций сравнения и логических операций «И», «ИЛИ», «НЕ»

<i>логическая операция</i>	<i>знак в C++</i>	<i>наименование знака</i>
И	&&	двойной амперсанд
ИЛИ		двойная вертикальная черта
НЕ	~	тильда

В частности, выражение «a < b < c» сформулируем как «a меньше b, и b меньше c». На C++ это будет записано как (a<b)&&(b<c). В тексте программы соответствующий оператор будет иметь вид

```
if((a<b)&&(b<c))
{
    ... ..;
}
```

При организации множественного выбора бывает удобно использовать не вложенные if-else, а оператор switch. Его синтаксис можно описать так:

```
switch (целая_переменная) {
    case константа1:
        операции;
    case константа2:
        операции;
    ....
    default:
        операции;}
```

Порядок выполнения работы:

8. Изучить теоретический материал по теме.
9. Разработать алгоритм решения в соответствии с заданием.
10. Составить программный код.
11. Протестировать программу.

Форма представления результата:

Отчет по работе должен содержать:

- а) наименование работы и цель работы;
- б) формулировку задачи;
- в) блок схему или программный код;
- г) результаты работы программы.

Контрольные вопросы

1. Перечислить действия, реализуемые при выполнении условного оператора.
2. Что такое вычислительный процесс разветвляющейся структуры?
3. Как организовать разветвление вычислений: а) на две ветви; б) на три ветви?
4. Каким образом в операторе множественного выбора switch осуществить выполнение одного и того же логического блока при нескольких константах выбора?
5. Почему недопустим следующий фрагмент кода программы?

```
int x= 1 ;  
float y= 2, z= 3.52;  
x= z%y;
```

Тема 2.3. Базовые конструкции языков программирования

Практическое занятие № 6,7,8

Оператор цикла

Формируемая компетенция:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем

Цель работы: овладение практическими навыками разработки и программирования вычислительного процесса циклических структур.

Выполнив работу, Вы будете:

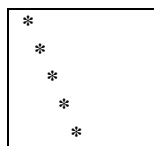
уметь:

- применять структуры цикла в программном коде для решений задач.

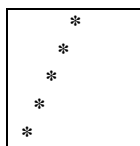
Задание:

1. Составить программный код для алгоритма задачи №1 практическая работа №1.
2. Протестировать программу для контрольных значений.
3. Напишите программу, выводящую на экран символы «*» согласно приведенным ниже шаблонам. Обязательным является использование циклов.

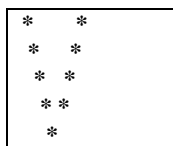
а)



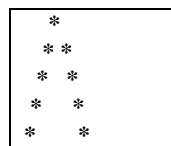
б)

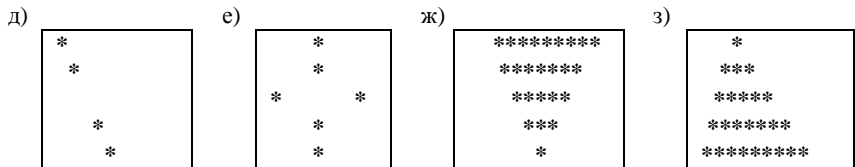


в)



г)





4. Напишите программу, выводящую на экран символы «*» согласно приведенным ниже шаблонам. Обязательным является использование циклов.

Краткие теоретические сведения:

В С++ существует несколько языковых конструкций для организации циклов.

1. Цикл for.

Общий формат его записи

```
for(<инициализация>;<условие_продолжения>;<изменение_счетчика>)
{
    <блок операторов>;
}
```

2. Цикл с предусловием while (пока).

while(выражение)

```
{
    оператор_1;
    оператор_2;
    ....
    оператор
}
```

Выражение может быть любым, допустимым в языке С выражением. *Оператор* может быть либо пустым, либо простым, либо составным.

Схема выполнения оператора while:

1. Вычисляется выражение;
2. Если, выражение ложно (равно нулю) то тело оператора while не выполняется, а управление передается на следующий за while оператор;
3. Если выражение истинно (не нуль), то тело оператора while выполняется;
4. Процесс повторяется с шага 1 .

3. Цикл с постусловием

Оператор цикла do используется в тех случаях, когда тело цикла должно выполняться хотя бы один раз.

Формат оператора do:

```
do
    оператор;
```

while (*выражение*);

Схема выполнения оператора do:

1 Выполняется *оператор*;

2. Вычисляется *выражение*. Если выражение не равно нулю (истинно), то выполнение продолжается с шага 1. Если выражение равно нулю (ложно), то управление передается следующему оператору программы.

Чтобы прервать цикл до того, как условие станет ложным, можно использовать оператор break.

Оператор break.

Оператор **break** обеспечивает прекращение выполнения самого внутреннего из объемлющих его операторов switch, do, for и while .

После выполнения оператора break управление передается оператору, следующему за прерванным.

Оператор continue.

Оператор **continue** работает подобно оператору break , но в отличие от оператора break прерывает выполнение тела цикла и передает управление на следующую итерацию. Формат оператора continue:

continue;

Оператор continue так же, как оператор break прерывает самый внутренний из объемлющих его циклов.

Пример решения задачи, которая находит все простые числа в диапазоне от 3 до 32767 с помощью операторов:

```
// с помощью while                                // с помощью оператора for
include<stdio.h>                                    #include<stdio.h>
include<conio.h>                                    #include<conio.h>
main()                                              main()
{ clrscr();                                         { clrscr();
int i; int n=2;                                     int i, n;
    while(++n<=32767)                               for(n = 2;n<=32767;++n)
    { i=1;                                           { for(i=2;i<n; ++i)
while(++i<n)                                         if(n%i ==0) break;
    if(n%i==0) break;                                if(i== n)
    if(i== n)                                        printf(“Простое число%d\n”,n);
printf(“Простое число%d\n”,n);                       } getch();
    } getch();                                       }
    }
}
```

Порядок выполнения работы:

12. Изучить теоретический материал по теме.
13. Разработать алгоритм решения в соответствии с заданием.
14. Составить программный код.
15. Протестировать программу.

Форма представления результата:

- Отчет по работе должен содержать:
- наименование работы и цель работы;
 - формулировку задачи;
 - блок схему или программный код;
 - результаты работы программы.

Контрольные вопросы

- Опишите условия применения цикла с параметром
- При каких условиях применяются циклы `while` и `do_while`?
- Объясните алгоритм работы трех видов цикла.

Тема 2.4. Массивы Практическое занятие № 9,10,11

Поэлементные операции

Формируемая компетенция:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем

Цель работы: овладение практическими навыками разработки и программирования вычислительного процесса обработки элементов одинакового типа.

Выполнив работу, Вы будете:

уметь:

- использовать язык программирования при обработки элементов одномерных и двумерных массивов;

Задание:

- Разработать алгоритм решения в соответствии с заданием по вариантам.
- Записать программу на языке программирования C++
- Протестировать программу для контрольных значений.

Вариант	Задача
1.	Напишите программу, запрашивающую у пользователя N целых чисел и выводящую на экран: только положительные числа
2.	Напишите программу, запрашивающую у пользователя N вещественных чисел и выводящую на экран: сумму чисел с четными порядковыми номерами
3.	Напишите программу, запрашивающую у пользователя N целых чисел и выводящую на экран: а) количество нулей в последовательности; б) количество положительных чисел; в) количество отрицательных чисел.
4.	Напишите программу, запрашивающую у пользователя N вещественных чисел и выводящую на экран: сумму и

	произведение чисел с нечетными порядковыми номерами.
5.	Напишите программу, запрашивающую у пользователя N вещественных чисел и выводящую на экран их среднее арифметическое.
6.	Напишите программу, запрашивающую у пользователя N целых чисел и выводящую на экран: только положительные числа стоящие на четных порядковых номерах.
7.	Напишите программу, запрашивающую у пользователя N целых чисел и выводящую на экран: только отрицательные числа стоящие на нечетных порядковых номерах.
8.	Напишите программу, запрашивающую у пользователя N вещественных чисел и выводящую на экран: количество и сумму чисел с четными порядковыми номерами
9.	Напишите программу, запрашивающую у пользователя N вещественных чисел и выводящую на экран: количество и произведение чисел с нечетными порядковыми номерами
10.	Напишите программу, удаляющую из набора: а) все отрицательные числа; б) все нули; в) все положительные числа.

4. Составить ряд процедур для решения одной из задач. В решении задачи можно использовать произвольный способ ввода матрицы. Результаты должны выводиться на экран. Элементы двумерного массива (n,n) генерируются случайным образом в заданном диапазоне $(a;b)$, где n - количество букв в имени; a - количество букв в фамилии; b - количество букв в отчестве.

5. Протестировать программу для контрольных значений.

Вариант	Задача №2
1.	Обнулить все нечетные элемента матрица
2.	Поменять местами элементы первого и последнего столбца.
3.	Обнулить все четные элемента матрица
4.	Заменить элементы матрицы стоящие на главной и второстепенной диагонали единицами.
5.	Поменять местами элементы двух средних столбцов
6.	Определить сумму элементов стоящих на главной и второстепенной диагонали
7.	Определить сумму элементов стоящих на местах сумма индексов которых четное число

8.	Определить сумму элементов стоящих на местах сумма индексов которых нечетное число
9.	Обнулить все нечетные элемента матрица
10.	Поменять местами элементы первой и последней строки

Краткие теоретические сведения:

Линейным массивом в программе на С++ называется упорядоченный набор однотипных переменных, которые располагаются в памяти последовательно. Любой массив в С++ характеризуется тремя параметрами: именем, типом элементов и размером.

Примеры объявлений

```
int A[15]; // массив из 15 целочисленных элементов с именем А
```

```
float x[3]; // массив x из 3-х элементов типа float
```

```
int F[3][3] // массив из 9 целочисленных элементов с именем F
```

```
int F[3][3] = { {3, 0, 2} , (1, 9, 8), {5, 7, 4} }; // объявлен и инициализирован двумерный массив размерностью 3 на 3.
```

Объявление массива является командой компилятору на выделение памяти для хранения его элементов. Общее количество выделенной памяти зависит не только от числа элементов, но и от размера каждого элемента, то есть от его типа. Например, текстовая строка из 1000 символов (тип char) займет $P = 1000 * \text{sizeof}(\text{char}) = 1000$ байтов, а массив из такого же количества вещественных чисел двойной точности (тип double) займет уже в восемь раз больше – $P = 1000 * \text{sizeof}(\text{double}) = 8000$ байтов.

Нумерация элементов в массиве начинается с нуля. Таким образом, первый элемент массива имеет индекс 0, а последний – индекс $n-1$, где n – размер массива. Обращение к элементу производится с использованием имени массива и индекса элемента в квадратных скобках. Например, запись “ $x[0] = 5.5;$ ” означает “присвоить значение 5.5 нулевому элементу массива x ”.

Для работы с массивами характерным является использование итерационных циклов for. С их помощью организуется выполнение однотипных операций со всеми элементами массива, в частности, поэлементный ввод-вывод, поэлементные арифметические операции и др.

Пример. Напишите программу, запрашивающую у пользователя 10 целых чисел, и выводящую ее на экран их сумму.

Решение. Будем использовать массив с именем А и размером 10 для хранения введенных чисел. Ввод данных и суммирование организуем поэлементно с помощью циклов for.

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

```

int main(int argc, char* argv[])
{
    int A[10]; // объявляем массив из 10 целых
    for(int i=0; i<10; i++) // организуем цикл по i от 0 до 9
    {
        cout << "input A[" << i << "] = ";
        cin >> A[i]; // вводим A[i]
    }
    int sum = 0; // объявляем переменную
    for(int i=0; i<10; i++) // организуем цикл
        sum = sum + A[i]; // в цикле суммируем элементы
    cout << "\nSumma: " << sum; // выводим результат на экран
    getch(); // задержка
    return 0;
}

```

Приведенный ниже фрагмент программного кода выводит на экран все элементы массива G.

```

for(int i=0; i<5; i++) // цикл по строкам i
{
    for(int j=0; j<4; j++) // цикл по строкам j
        cout << G[i][j] << "\t"; // выводим G[i][j]
    cout << endl; // перевод на новую строку
}

```

Тема 2.4. Массивы

Практическое занятие № 12,13,14

Алгоритмы поиска и сортировки и замены.

Формируемая компетенция:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем

Цель работы: научиться решать задачи на сортировку, поиска, замены и суммирования массивов с помощью алгоритмов в языке C++.

Выполнив работу, Вы будете:

уметь:

- использовать язык программирования при обработки элементов одномерных и двумерных массивов;

Задание:

1. Разработать алгоритм и записать программный код одного из пунктов следующей задачи:

В одномерном массиве, состоящем из n целых элементов:

- вычислить сумму отрицательных и положительных элементов;
- найти максимальное и минимальное значение
- расположить элементы в обратном порядке;

- расположить в порядке возрастания;
 - расположить в порядке убывания.
2. Разработать алгоритм решения в соответствии с заданием по вариантам.
 3. Записать программу на языке программирования C++
 4. Протестировать программу для контрольных значений.

Вариант	Задача
1.	Дан набор из N целых чисел. Напишите программу, которая вычислит максимальный недиагональный элемент, а так же определит сумму каждого столбца матрицы.
2.	Написать программу поиска наибольшего из четных элементов данного массива и его порядкового номера, а так же определит сумму каждого столбца матрицы.
3.	Написать программу поиска наименьшего из четных элементов данного массива и его порядкового номера, а так же определит сумму каждой строки матрицы.
4.	Составить программу, после выполнения которой выясняется сколько чисел входит в этот массив более чем один раз, а так же определит сумму каждой строки матрицы.
5.	Написать программу подсчитывающую количество строк, содержащих хотя бы один нулевой элемент, а так же определит сумму каждой строки матрицы.
6.	Упорядочить строки матрицы по возрастанию. Определить номер первой строки, где содержатся повторяющиеся значения.
7.	Определить сумму элементов в тех столбцах, которые не содержат единиц, а так же найти максимальный элемент стоящий на главной диагонали.
8.	Вычислить суммы элементов стоящих выше главной диагонали и максимальный элемент матрицы.
9.	Определить номер минимального элемента. Произведение элементов массива, расположенных между минимальным и максимальным элементами.
10.	Найти произведение элементов массива с четными номерами, а так же сумму элементов, расположенных до максимального элемента.

Краткие теоретические сведения:

1. Линеинный поиск по условию.

Пусть перед нами стоит следующая задача: имеется массив данных, в котором требуется найти элемент, удовлетворяющий заданному условию (например, равный нулю). Организуем перебор всех элементов массива, начиная с нулевого, с помощью итерационного цикла for. Для каждого элемента будем проверять выполнение заданного условия. В случае если условие выполнено, запоминаем индекс найденного элемента во вспомогательной переменной. По выходе из цикла искомый элемент может быть получен через его индекс.

С небольшими изменениями этот же алгоритм может использоваться и для поиска группы элементов, удовлетворяющих условию. Проиллюстрируем это на примере.

Пример 1. Дан массив X из N целых чисел. Напишите программу, подсчитывающую количество элементов, хранящих значение A.

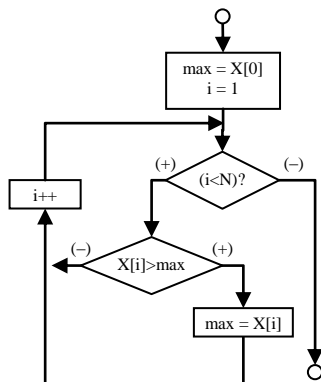
```
#pragma hdrstop
#include <conio.h>
#include <iostream.h>
const int N = 10; // N - размер массива
#pragma argsused
int main(int argc, char* argv[])
{
    int X[N], A; // объявляем X и A
    for(int i=0; i<N; i++) // вводим элементы
        массива
    {
        cout << "input X[" << i << "]=";
        cin >> X[i];
    }
    cout << "\n input A="; // вводим значение A
    cin >> A;

    int sum = 0; // доп. переменная sum
    for(int i=0; i<N; i++) // в цикле перебираем все
        X[i]
        if(X[i]==A) // если X[i] равен A
            sum++; // то увеличиваем sum на 1
    cout << "\n kol-vo chisel " << A <<
        " v massive = " << sum; // выводим результат
    getch();
    return 0;}
```

2. Поиск максимального (минимального) элемента.

Методы поиска максимального (или минимального) элемента в массиве несколько отличаются от методов поиска по условию, в силу того, что искомое значение заранее не известно.

Модифицируем описанный в пункте 1 алгоритм следующим образом. Введем вспомогательную переменную (к примеру, max) и запишем в нее значение нулевого элемента массива. Далее организуем перебор всех оставшихся элементов в цикле. Если по ходу встречается элемент со значением, большим max, то записываем его в max, переопределяя, таким образом, значение этой переменной. После перебора всех элементов max будет хранить искомый (в данном случае максимальный) элемент массива. Блок-схема этого алгоритма показана на рисунке слева.

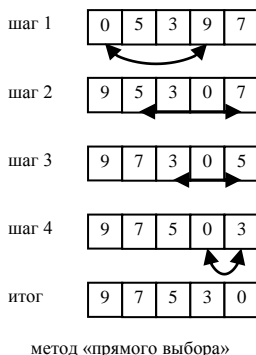


3. Сортировка методом «прямого выбора».

Под сортировкой будем понимать размещение набора данных в определенном порядке, что используется, как правило, для облегчения поиска нужного элемента или группы элементов. Это также одна из важных задач компьютерной обработки данных. От эффективности алгоритмов сортировки зависит, например, производительность работы баз и банков данных.

Мы рассмотрим два простейших метода сортировки: метод «прямого выбора» и метод «пузырька». Для определенности будем считать, что перед нами стоит задача сортировки одномерного массива $A[N]$ по убыванию. Сортировка по возрастанию может быть проведена аналогичным образом.

Алгоритм метода «прямого выбора» основан на последовательном переборе всех элементов массива в поиске максимального и перемещении этого элемента на первую позицию в массиве. После этого процедура повторяется с оставшимися $N-1$ элементами – из них в свою очередь выбирается максимальный, и он меняется местами со вторым элементом массива. Процедура применяется последовательно $N-1$ раз, пока все элементы не окажутся упорядоченными гарантированно. Пример работы алгоритма показан пошагово на



метод «прямого выбора»

рисунке справа. Здесь исходный массив из 5-ти элементов сортируется в 4 этапа.

```

Приведенный ниже фрагмент кода решает эту задачу
const int N = 10; // размер массива
float A[N] = {5, 7, 1, 9, 6, 2, 8, 0, 4, 3}; // исход. массив
cout << "\nPered sortirovki:\n";
for(int i=0; i<N; i++) // выводим в исходном порядке
    cout << A[i] << " ";
for(int i=0; i<N; i++) // цикл по шагам сортировки
{
    int max_index = i; // вводим доп. переменные
    float max = A[i];
    for(int j=i; j<N; j++) // цикл поиска наибольшего элемента
    {
        if(A[j]>=max) // если очередной элемент > max
        {
            max_index = j; // то запоминаем его индекс
            max = A[j]; // и изменяем значение max
        }
    }
    float tmp = A[i]; // доп. переменная для перестановки
    A[i] = A[max_index]; // меняем местами элементы
    A[max_index] = tmp; // A[i] и A[max_index]
}
cout << "\nPosle sortirovki:\n";
for(int i=0; i<N; i++) // выводим отсортированный массив
    cout << A[i] << " ";

```

4. Сортировка методом «пузырька».

Метод «пузырька» получил свое название оттого, что продвижение максимальных элементов массива к его вершине происходит постепенно, подобно всплытию пузырька на поверхность воды.

Как и предыдущий алгоритм, этот метод требует нескольких проходов массива. На каждом проходе сравнивается пара соседних друг с другом элементов. Если пара расположена в порядке возрастания, переставляем эти элементы местами. В противном случае элементы оставляются на исходных позициях. Процедура должна быть повторена N-1 раз для гарантированного достижения конечного результата. Пример



метод «пузырька»

поэтапной работы алгоритма показан на рисунке слева. Изображенные на каждом проходе перестановки должны выполняться последовательно слева направо.

Ниже приведена программная реализация метода

```
const int N = 10;
float A[N]={5, 7, 1, 9, 6, 2, 8, 0, 4, 3};
cout << "\nPered sortirovkoj:\n";
for(int i=0; i<N; i++)
    cout << A[i] << " ";
for(int i=1; i<N; i++)                                // организуем N-1 проходов
    for(int j=0; j<N-i; j++)
        if(A[j]<A[j+1])                               // если порядок неправильный
        {
            float tmp = A[j];                         // запоминаем A[j]
            A[j] = A[j+1];                             // меняем местами элементы
            A[j+1] = tmp;                               // A[j] и A[j+1]
        }
cout << "\nPosle sortirovki:\n";
for(int i=0; i<N; i++)                                // выводим на экран
    cout << A[i] << " ";
```

Контрольные вопросы

1. Опишите условия применения цикла с параметром
2. Линейный поиск по условию. Общий алгоритм и его блок-схема.
3. Поиск максимального и минимального элементов. Общий алгоритм и его работа на примере.
4. Сортировка методом «прямого выбора». Общий алгоритм и пошаговая реализация на примере.
5. Сортировка методом «пузырька». Общий алгоритм и пошаговая реализация на примере.
6. Объясните, почему при поиске максимального и минимального элемента в Примере 3 (см. стр. 62) переменным `min` и `max` первоначально присвоены значения `A[0][0]`. Можно ли для этой цели использовать другие элементы? Чем должен определяться выбор начальных значений `min` и `max`?

Тема 2.5. Функции

Практическое занятие № 15,16

Параметры функции

Формируемая компетенция:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем

Цель работы: овладение навыками написания функций и обращения к ним, выбора параметров подпрограмм.

Выполнив работу, Вы будете:

уметь:

- использовать язык программирования при разработки программ с использованием функций пользователя;

Задание:

1. Изучить:

- правила записи функций и способов обращений к ним;
- способов передачи параметров в функцию;
- правила записи программ, использующих функции;
- порядок выполнения программ, использующих подпрограммы.

2. Разработать алгоритм решения в соответствии с заданием.

3. Составить программу решения задачи в соответствии с вариантом.

1. Напишите пользовательскую функцию, вычисляющую площадь заданной геометрической фигуры:

- а) площадь круга по его радиусу;
- б) площадь трапеции по двум основаниям и высоте;
- в) площадь треугольника по трем сторонам.

2. Напишите функцию, заполняющую окно приложения заданным текстовым символом.

3. Напишите функцию, вычисляющую длину гипотенузы прямоугольного треугольника по двум его катетам.

4. Напишите функции, вычисляющие y и z соответственно.

a. $y = x(1 + x^3); \quad z = \sum_{k=1}^5 (k + y); x = 1; 5$

b. $y = x^2 + x; \quad z = \sum_{k=1}^4 \left(\frac{k}{y} \right); x = 1; 4$

c. $y = 1 + x^4; \quad z = \sum_{k=1}^4 (y^2 + k); x = 1; 5$

d. $y = x + 7; \quad z = \prod_{k=1}^5 (k * y); x = 1; 5$

e. $y = \frac{x^{2n}}{n + 2}; \quad z = \sum_{k=1}^5 (k + y^2); n = 1; 5$

f. $y = x^3; \quad z = \sum_{k=1}^6 (k^2 + y); x = 1; 6$

Краткие теоретические сведения:

Определение функции состоит из двух частей: заголовка и тела. Заголовок определяет имя функции, ее тип и формальные параметры, тело определяет действия над данными, выполняемые функцией. Возвращаемое функцией значение передается в вызывающую программу оператором `return` (выражение). Значение "выражения" и есть результат функции (возвращаемого значения). Если в нашей программе функция физически следует за вызывающей ее функцией `main`, то надо в последней объявить функцию внешней с помощью описателя `extern`: `extern int fun();` или еще проще `int fun();`. В противном случае при компиляции будет выдана ошибка. Всякая функция имеет вид:

```
тип_результата имя_функции(список_аргументов)
{
    <объявления локальных переменных>;
    <операторы>;
    return <возвращаемое_значение>;
}
```

Для обращения к функции (ее вызова) используется имя функции с указанием набора передаваемых ей параметров в круглых скобках *имя_функции(список_фактических_параметров)*

Результатом этой операции является возвращаемое функцией значение. Вызов функции может быть произведен из любого места программы, в котором допускаются исполняемые операторы.

Пример. Напишите функцию, выбирающую наименьшее из трех вещественных чисел.

Решение. Входные параметры для функции – 3 вещественных числа, например `a,b,c`, результат – также вещественное число, содержащее минимальное из них. Функция может быть определена следующим образом

```
float minimal(float a, float b, float c)
{
    float m = a; // локальная переменная m
    if(b<m) // выбираем наименьшее из 2-х
        m = b;
    if(c<m) // выбираем наименьшее из 2-х
        m = c;
    return m; // возвращаем результат m к месту вызова
}
```

Отметим важную особенность использования функций. Определение функции в списке аргументов содержит формальные параметры (`float a`, `float b` и `float c`, см. Пример 1), значения которых неизвестны на этапе написания тела функции. Они обретают конкретное содержание в момент вызова функции, когда в скобках указываются

фактические параметры, то есть переменные или константы, имеющие определенное значение (как x, y, z из примера выше). Имена формальных и фактических параметров не обязаны быть одинаковыми. Неукоснительным требованием является лишь соответствие их типов, то есть тип первого в списке фактического параметра (float x) должен совпадать с типом первого формального параметра (float a), тип второго (float y) – со вторым (float b), и т.д.

Контрольные вопросы

1. Опишите условия применения цикла с параметром
2. Укажите при каких условиях целесообразно использование функции, какие выгоды они предоставляют пользователю.
3. Укажите чем отличие различных видов функции пользователя.
4. Укажите способ обращения к функции пользователя.

Тема 2.5. Функции

Практическое занятие № 17,18

Рекурсивные функции. Многофайловые проекты

Формируемая компетенция:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем

ПК 3.3. Принимать участие в отладке и технических испытаниях компьютерных систем и комплексов: инсталляции, конфигурировании и настройке операционной системы, драйверов, резидентских программ.

Цель работы: научиться применять рекурсивные алгоритмы, решать задачи с использованием прямого доступа к данным файла на языке C++.

Выполнив работу, Вы будете:

уметь:

- использовать язык программирования при разработки программ с использованием функций пользователя;
- разрабатывать на языке программирования программы состоящие из нескольких файлов.

Задание:

1. Разработать алгоритм и программу решения задачи №1,2.

Задача №1 Вычислить значения величины вклада по формуле сложных процентов: $\text{sum} \left(1 + \frac{p}{100} \right) n$.

Задача №2 Найдите сумму факториалов первых n натуральных чисел.

2. Разработать программу, выполняющую ввод данных, преобразование текстового файла в двоичный файл, вывод двоичного файла

Для работы с файлами использовать библиотеку классов потокового ввода – вывода fstream.h. Варианты заданий приведены в таблице.

№	Объект
1	ФИО, номер подразделения, зарплата
2	Табельный номер, ФИО, номер подразделения
3	Наименование товара, цена, количество
4	Телефон, ФИО, адрес
5	Шифр студента, ФИО, группа
6	Шифр студента, ФИО, средний балл
7	Табельный номер, ФИО, кафедра
8	Наименование товара, цена, количество
9	Шифр студента, год поступления. специальность
10	Шифр студента, четыре оценки
11	ФИО, год рождения, пол
12	Номер заказа, ФИО и телефон заказчика
13	Шифр группы, средний балл
14	Номер заказа, цена заказа, заказчик
15	Автор, название и год издания книги
16	Номер задания, имя задания, количество страниц
17	Код работника, профессия, опыт работы
18	Код детали, наименование детали
19	Табельный номер, должность, подразделение
20	Авторы, название, специальность для учебного пособия

3. Придумайте и напишите свой пример программы, состоящей из двух-трех файлов исходного кода.

Краткие теоретические сведения:

В языке Си можно использовать рекурсивно, т.е. функция может вызывать сама себя. Простейший пример 1, где функция main вызывает сама себя остановить которую можно только с помощью CTRL+C - ^C.

При рекурсивном обращении к функции - создается новый экземпляр данных.

Пример 1

```
#include <stdio.h>
main()
{printf("проверка рекурсии \n");
main();}
```

Опорная схема "Увидеть"

Данная опорная схема является наиболее естественной, так как содержится в постановке задачи. Для разработки триады достаточно использовать параметры, тривиальный случай и соотношения, непосредственно вытекающие из условия. Следующий пример 2 использует рекурсию для вычисления $k!$. При первом вызове $fact(i)$, если не равно 1, функция порождает выражение $i*fact(i-1)$; в свою очередь,

вызов $\text{fact}(i-1)$ производит $(i-1) \cdot \text{fact}(i-2)$, что вместе с ранее полученным результатом дает $i \cdot (i-1) \cdot \text{fact}(i-2)$. Рекурсия закончится, как только следующий вызов функции получит аргумент, равный единице. Нетрудно сообразить, что в конечном счете мы получим требуемое произведение.

Пример 2. Напишите функцию, вычисляющую факториал заданного целого числа N . Используйте эту функцию для вывода на экран факториалов всех целых чисел от 0 до 10.

Решение. Будем использовать итеративный метод расчета факториала, согласно его определению $N! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot N$. Аргумент и возвращаемое значение функции опишем как целочисленные (`int`).

```
#pragma hdrstop
#include <conio.h>
#include <iostream.h>
#include <math.h>
int fact(int);           // прототип функции fact
#pragma argsused
int main(int argc, char* argv[]) // главная функция - main
{
    const int N = 10;
    for(int i=0; i<=N; i++) // вызываем функцию в цикле
        cout<<i<<"! = "<<fact(i)<<endl;
    getch();
    return 0;
}
int fact(int n)         // определение функции fact
{
    if(n<0)             // проверка на отрицательный
    {                   // аргумент
        cout << "Error: n<0!"; // вывод сообщения об ошибке
        return -1;
    }
    int r = 1.;
    for(int i=1; i<=n; i++) // итеративный расчет n!
        r *= i;
    return r; }        // возвращаем результат
```

Опорная схема "Переформулировать"

Часто в условии задачи не только не обозначена рекурсия, но и сама задача не является алгоритмически сформулированной. Иногда ее простая перефразировка, а чаще построение математической модели позволяют обнаружить первоначально скрытую рекурсию.

Рассмотрим задачу о динамике вклада. Большой выбор простых содержательных задач, допускающих рекурсивное решение, можно

встретить в сфере банковской деятельности. Рассмотрим несколько различных рекурсивных вариантов решения задачи о динамике вклада.

Вкладчик положил в банк сумму в sum денежных единиц под p процентов за один период времени. Составим функцию, возвращающую величину вклада по истечении n периодов времени.

Вычислить значения величины вклада по формуле сложных процентов: $sum \left(1 + \frac{p}{100} \right)^n$

База рекурсии: для $n=0$ размер суммы не изменится, то есть останется sum .

Декомпозиция: если $n>0$, то размер вклада вычисляется как сумма за $n-1$ периодов, увеличенная на процент p .

```
float Deposit(float sum, float p, int n){
    if(n==0) return sum; //база рекурсии
    return Deposit(sum,p,n-1)*(1+p/100); //декомпозиция
}
```

Текстовые (ASCII) и бинарные (двоичные) файлы

При создании текстового файла используются два разных режима: текстовый (или просто ASCII) и бинарный (двоичный). В текстовом режиме все данные хранятся в виде символьных строк (char).

Допустим, у нас есть число 1492. Если мы будем сохранять это число в текстовый файл, то число будет представлено строкой символов.: 49 52 57 50. То есть число займёт в памяти четыре байта. В двоичном режиме это число займёт два байта (если использовать тип short), и сохранится в том же виде, в каком оно хранилось в памяти - 1492.

Создание текстового файла

Для использования возможностей работы с файлами, необходимо включить заголовочный файл `fstream`. Также нужно подключить заголовочный файл `iostream` (операторы вставки и извлечения из потоков).

```
ofstream os("text.txt");  
os << "Hello";  
os.close();
```

Конструктор класса `ofstream` принимает параметр - символьную строку содержащую название файла, в который будет осуществляться вывод. Затем потоковый объект используется точно также как и `cout`. Только вывод будет осуществляться не в консоль, а в файл `text.txt`.

Последний оператор - вызов метода `close`. Данный метод закрывает файл. Когда поток открывает файл, то этот файл блокируется, и никто не сможет получить к нему доступ. Для этого и необходим вызов метода `close` - файл закрывается и разблокируется.

Теперь прочитаем этот файл и выведем его содержимое на экран:

```
char a[6];
```

```
ifstream is("text.txt");  
is >> a;  
cout << a;  
is.close();
```

На экран будет выведено слово: Hello

Is использует оператор извлечения из потока, точно также как и объект cin. Обратите внимание на размер массива a. Мы выделили на один байт больше количества букв в слове, так как в этот массив считается ещё и символ конца файла. Файл будет находится в папке решение/debug.

Двоичный (бинарный) режим записи и чтения файлов

Конструкторы классов ifstream и ofstream могут принимать второй параметр - набор флагов. Одним из флагов указывается двоичный режим:

```
ofstream os("text.txt", ios::binary);  
ifstream is("text.txt", ios::binary);
```

Замечание: хотя здесь и создаётся файл с расширением txt, этот файл уже не является текстовым!

Для записи в файлы и чтения из файлов объектами is и os могут использоваться функции write (писать) и read (читать). Первый аргумент этих функций - адрес буфера (указатель на char), второй аргумент - количество байт в буфере. Прототипы read и write довольно сложны, но упрощённо они выглядят примерно так:

```
void ofstream::write (char*,long);  
void ifstream::read (char*,long);
```

Поместим число типа int в файл:

```
ofstream os("text.txt", ios::binary);  
int a = 1492;  
os.write(reinterpret_cast <char*>(&a),sizeof(int));
```

Оператор reinterpret_cast изменяет тип данных в памяти не обращая внимания, имеет это смысл или нет. Допустим этому оператору нужно привести тип int* к типу char*. Оператор reinterpret_cast ничего не будет делать с данными содержащимися в переменной типа int, он просто будет считать этот участок памяти (4 байта занятые переменной типа int) как массив из четырёх байт. Использовать reinterpret_cast нужно только в случаях, подобных этому: когда, например, необходимо привести тип int* к типу char* или совершить преобразования между пользовательскими типами.

Обратите внимание, что аргумент оператора reinterpret_cast - адрес переменной a. Мы передаём адрес, так как в методах write и read происходит приведение к типу char*. Передадим теперь в метод write массив переменных.

```

#include <conio.h>
#include <iostream.h>
#include <fstream>
int main()
{
ofstream os("text.txt", ios::binary);
int a[] = { 1492, 31562, 290893,382 };
os.write(reinterpret_cast<char*>(a),sizeof(a));
os.close();
getch();
return 0;

```

Пример создания текстового файла и записи в него данных в виде структуры.

```

#include <vcl.h>
#pragma hdrstop
#include <conio.h>
#include <fstream.h>
#include <iostream.h>
#include <stdlib.h>
struct rab
{
long nom;
char fam[15];
char name[15];
int zar;
int nal;
};
//-----
#pragma argsused
int main(int argc, char* argv[])
{
system("chcp 1251>>NULL");
rab r;
ofstream f; // выходной поток
char name[12]; // имя файла
cout<<"Введите имя файла";
cin>>name;
f.open(name); //открытие файла для записи
if(!f) //проверка выполнения операции открытия
{ cout<<"Ошибка"; getch(); exit(1); }
for(int i=1;i<=10;i++) //цикл ввода данных и записи в файл
{cout<<"? "; cin>>r.nom>>r.fam>>r.name>>r.zar>>r.nal;
f<<r.nom<<" "<<r.fam<<" "<<r.name<<" "<<r.zar<<" "<<r.nal <<endl;}

```

```
f.close();
return 0;
}
```

Пример чтения из двоичного файла

```
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
struct rab
{
    long nom;
    char fam[15];
    char name[15];
    int zar;
    int nal;
};
void main(void)
{
    rab r;
    int n;
    ifstream f;    //входной поток
    f.open("d:\\user\\b.dat",ios::binary| ios::in);
    if(!f)
        { cout<<"Ошибка"; getch(); exit(1);}
    cout<<"n ? "; cin>>n;
    f.seekg((n-1)*sizeof(r));    // перемещение указателя на запись
    f.read((char*)&r,sizeof(r));    //чтение записи в структуру
    cout<<r.nom<<" "<<r.fam<<" "<<r.name<<" "<<r.zar<<" "<<r.nal
    <<endl;
    f.close();
}
```

Многофайловые проекты.

Рассмотрим пример простейшего проекта, состоящего из двух файлов с расширением сpp и одного файла с расширением h. Для определенности проект назовем MyProj. В проекте необходимо вычислить значение функции $y = \sin(x) + 0,5$. Значение x вводится с клавиатуры. Вычисления организовать с помощью функции, которую поместить в отдельный модуль.

Функция main() будет расположена в файле MyProj.cpp, функция f(), вычисляющая значение y – в файле Modul.cpp, а прототип функции f() – в заголовочном файле Modul.h (обе части модуля имеют одно имя).

При создании программы из нескольких модулей следует соблюдать принцип независимой компиляции модулей. Этот принцип

заключается в том, что при подготовке исполняемого файла сначала надо добиться компиляции каждого модуля независимо от других модулей. Отсутствие синтаксических ошибок во всех модулях проекта – необходимое условие для создания исполняемого файла, но недостаточное, так как ошибки могут появиться при объединении модулей, и пока они не будут исправлены, исполняемый файл не получить.

Следующий текст удовлетворяет принципу независимой компиляции модулей и решает поставленную задачу:

Файл MyProj.cpp

```
#include <stdio.h>
#include "Modul.h"
int main()
{
float y,x;
scanf("%f",&x);
y=f(x);
printf("y=%f\n",y);
return 0;
}
```

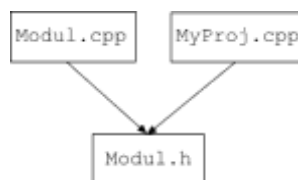
Файл Modul.h

```
float f(float);
```

Файл Modul.cpp

```
#include "Modul.h"
#include <math.h>
float f(float x)
{
float z1;
z1=sin(x)+0.5;
return (z1);
}
```

Проект можно представить графически с помощью диаграммы модулей (файлов). Каждый файл на этой диаграмме изображается в виде прямоугольника, между прямоугольниками линиями со стрелками показаны связи файлов, которые соответствуют директиве include. Стрелки направлены от зависимого файла к независимому. Независимым файлом обычно бывает заголовочный файл, а зависимый – файла типа *.c, *.cpp.



Рассмотрим пример проекта, использующего внешние (extern) переменные. В модуле mod.cpp используется внешняя переменная a. В файле prog.cpp объявлена как глобальная и

инициализирована переменная с таким же именем. Значение глобальной переменной `a` передается в функцию модуля `mod.cpp` для обработки. Переменная класса `extern` может быть объявлена как внутри блока, так и на внешнем уровне.

В нашем примере внешняя переменная `a` объявлена внутри функции `prc()`:

Файл `prog.cpp`

```
#include "mod.h"
```

```
int a=5;
```

```
int main(void)
```

```
{
```

```
prc();
```

```
return 0;
```

```
}
```

Файл `mod.h`

```
void prc(void);
```

Файл `mod.cpp`

```
#include "mod.h"
```

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

```
void prc(void)
```

```
{
```

```
extern a;
```

```
printf("%d\n",2*a);
```

```
}
```

Внешняя переменная `a` представляет собой «мостик», по которому данные (значение глобальной переменной `a`) из модуля `prog.cpp` передаются в модуль `mod.cpp` для дальнейшей обработки.

Контрольные вопросы

1. Почему рекурсию нельзя рассматривать как универсальный метод решения задач?
2. Почему опорные схемы решения задач рекурсивными способами не являются жестко привязанными к отдельным классам задач?
3. Каким образом анализ решения обратной задачи может привести к решению поставленной задачи?
4. Каким способом в программе происходит идентификация открытых, закрытия файлов?

Тема 2.6. Указатели

Практическое занятие № 19,20

Работа с указателями. Динамическое распределение памяти.

Формируемая компетенция:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем.

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

ПК 3.3. Принимать участие в отладке и технических испытаниях компьютерных систем и комплексов: инсталляции, конфигурировании и настройке операционной системы, драйверов, резидентских программ.

Цель работы: научиться распределять и использовать динамическую память.

Выполнив работу, Вы будете:

уметь:

- использовать язык программирования при разработки программ с использованием указателей;

Задание:

1. Разработать алгоритм и программу решения задачи, в которой будет создаваться динамическая переменная, одномерный динамический массив:

- динамическое выделение памяти под объект типа `int`
- инициализация объекта через указатель
- объявление динамического объекта
- вывод данных
- высвобождение памяти
- создание динамического массива на 10 вещественных чисел
- заполнение массива случайными числами в диапазоне от -10 до 10.
- высвобождение памяти

Краткие теоретические сведения:

Указатель - это память, распределенная для другой переменной.

Если переменная объявлена как указатель, она может хранить адрес и таким образом указывать на другое значение. При объявлении переменной типа указатель необходимо указать тип данных, адрес которых будет содержать переменная, и имя указателя с предшествующей звездочкой.

Создание указателя: тип *имя;

Пример:

```
int *p; // указатель на переменную типа int или первый элемент массива типа int
```

```
char *pc; // указатель на переменную или массив типа char
```

Теперь надо заставить указатель на что-то указывать. Для этого мы можем использовать оператор `&` - функция взятия адреса переменной

```
int *p;
```

```
int a=10;
```

```
p=&a; //теперь p указывает на переменную a.
```

Для записи чего-либо в память, на которую указывает указатель надо использовать оператор *, который обозначает, что мы работаем с данными, находящимися по такому-то адресу

```
*p=20; //теперь a==20
```

Также мы можем динамически выделять память под данные во время работы программы. а не во время ее разработки. Например, нам вводят число символов в строке и затем строку. С помощью указателей и выделения памяти мы можем создать массив необходимого размера во время работы программы.

Указатели можно использовать и при определении массивов:

```
int a[100]={1,2,3,4,5,6,7,8,9,10};
```

```
int * pa=a;//поставили указатель на уже определенный массив
```

```
int b=new int[100];//выделили в динамической памяти место под массив из 100 элементов
```

Следующие операции недопустимы с указателями:

- сложение двух указателей;
- вычитание двух указателей на различные объекты;
- сложение указателей с числом с плавающей точкой;
- вычитание из указателей числа с плавающей точкой;
- умножение указателей;
- деление указателей;

Пример.

```
#include <vcl.h>
#pragma hdrstop
#include <iostream>
#include <conio.h>
#pragma argsused
using namespace std;
int main(int argc, char* argv[])
{
int i=2;
int *p=&i;
cout << &i<<endl;
cout <<i;
    getch();
}
```

Динамическая память – это память, выделяемая программе для ее работы за вычетом сегмента данных, стека, в котором размещаются локальные переменные подпрограмм и собственно тела программы.

Для работы с динамической памятью используют указатели. С их помощью осуществляется доступ к участкам динамической памяти, которые называются динамическими переменными. Для

хранения динамических переменных выделяется специальная область памяти, называемая "кучей".

В C++ используется два способа работы с динамической памятью:

- использование операций new и delete ;
- использование семейства функций malloc (calloc) (унаследовано из C).

Операция new позволяет выделить и сделать доступным свободный участок в основной памяти, размеры которого соответствуют типу данных, определяемому именем типа.

Синтаксис:

```
new ИмяТипа;
```

или

```
new ИмяТипа [Инициализатор];
```

Пример:

```
- float *pi; //Объявление переменной pi
- pi=new float; //Выделение памяти для переменной
- * pi = 2.25; //Присваивание значения
- int *a=new int[100]; //выделение динамической памяти
размером 100*sizeof(int) байтов
- double *b=new double[10]; // выделение динамической памяти
размером 10*sizeof(double) байтов
- long(*la)[4]; //указатель на массив из 4 элементов типа long
- int**matr=(int**)new int[5][10]; //еще один способ выделения
памяти под двумерный //массив
- for(int I=0;I<4;I++)matr[I]=new int[6]; //выделяем память под
строки массива
```

Пример объявления двумерного динамического массива.

```
float **ptrarray = new float* [2]; // две строки в массиве
```

```
for (int count = 0; count < 2; count++)
```

```
ptrarray[count] = new float [5]; // и пять столбцов
```

// где ptrarray – массив указателей на выделенный участок памяти под массив вещественных чисел типа float

Пример Дана двумерная матрица размером n,m. Удалить из матрицы строку с номером K

```
#include <iostream.h>
```

```
#include <string.h>
```

```
#include <iostream.h>
```

```
#include <stdlib.h>
```

```
void main()
```

```
{
```

```
    int n,m; //размерность матрицы
```

```
    int i,j;
```

матрицы

```
cout<<"\nEnter n";
cin>>n;//строки
cout<<"\nEnter m";
cin>>m;//столбцы
//выделение памяти
int **matr=new int* [n];// массив указателей на строки
for(i=0;i<n;i++)
    matr[i]=new int [m];//память под элементы

//заполнение матрицы
for(i=0;i<n;i++)
for(j=0;j<m;j++)
    matr[i][j]=rand()%10;//заполнение матрицы
//печать сформированной матрицы
for(i=0;i<n;i++)
{
    for(j=0;j<m;j++)
        cout<<matr[i][j]<<" ";
    cout<<"\n";
}
//удаление строки с номером k
int k;
cout<<"\nEnter k";
cin>>k;
int**temp=new int*[n-1];//формирование новой матрицы
for(i=0;i<n;i++)
    temp[i]=new int[m];
//заполнение новой матрицы
int t;
for(i=0,t=0;i<n;i++)
    if(i!=k)
    {
        for(j=0;j<m;j++)
            temp[t][j]=matr[i][j];
        t++;
    }

//удаление старой матрицы
for(i=0;i<n;i++)
    delete matr[i];
delete[]matr;

n--;
```

```
//печать новой матрицы
```

```

for(i=0;i<n;i++)
{
    for(j=0;j<m;j++)
        cout<<temp[i][j]<<" ";
    cout<<"\n";
}
}

```

Тема 3.2. Директивы и операторы ассемблера

Практическое занятие № 21,22,23,24,25

Программирования для MS DOC

Формируемая компетенция:

ПК 2.1. Создавать программы на языке ассемблера для микропроцессорных систем.

ПК 2.2. Производить тестирование и отладку микропроцессорных систем.

ПК 3.3. Принимать участие в отладке и технических испытаниях компьютерных систем и комплексов: инсталляции, конфигурировании и настройке операционной системы, драйверов, резидентских программ.

Цель работы: изучить процесс создания программ на языке Ассемблера. получить навыков в работе с компилятором, компоновщиком, отладчиком.

Выполнив работу, Вы будете:

уметь:

- использовать язык программирования при разработки программ для MS DOC;

Задание:

1. Создать bat файл по инструкции:

– создать папку в каталоге TASM и исходя из этого настраивать пути в BAT файлах.

– ввести в блокноте текстовый файл 2.asm с командами ассемблера:

```
MODEL TINY
```

```
STACK 256
```

```
CODESEG
```

```
start:
```

```
    mov ah, 04Ch
```

```
    int 21h
```

```
end
```

```
start
```

2. Откомпилировать полученный текст программы. При наличии ошибок внести исправления и



откомпилировать вновь. Для ассемблирования файла наберите команду: TASM 2. В результате компиляции мы получаем OBJ файл. Второй этап сборки выполняет файл TLINK.EXE который собирает исполняемый файл. Нам необходимо написать BAT файл с именем 2.bat и запустить его:

```
..\bin\tasm 2.asm
..\bin\tlink 2.obj
```

Для запуска программы уже готового exe файла наберите в командной строке Td 2.

В результате Вы увидите, что по поводу Вашего кода сказал TASM и будет собрана самая маленькая наверно программа из всех возможных которая пока умеет просто запускаться и не делать ошибок.

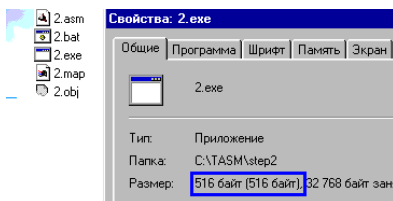
3. Написать, откомпилировать и отладить программу, использующую функции BIOS для работы с клавиатурой. Программа должна выдавать сообщение: нажата простая клавиша или функциональная. Для простых клавиш необходимо также выводить на экран символ, соответствующий нажатой клавише.

```
D_SEG SEGMENT WORD PUBLIC 'DATA'
```

```
Funkc db 'Funkc klavisha', '$'
Prost db 'Prost klavisha - ', '$'
D_SEG ends
```

```
C_SEG SEGMENT PARA PUBLIC 'CODE'
```

```
    assume cs:c_seg,ds:d_seg,ss:s_seg
main proc far
start:
    mov ax,SEG D_SEG
    mov ds,ax
    mov ah,0h
    int 16h
    cmp al,0h
    je mf
    mov ah,09h
    lea dx,Prost
    mov byte ptr Prost+17,al
    int 21h
    jmp mend
mf:
    mov ah,09h
```




```

        lea    dx,Funkc
        int    21h
mend:
        mov    ax,4C00h
        int    21h
main    endp
C_SEG  ends

```

```
S_SEG SEGMENT PARA STACK 'STACK'
```

```
    db    40h dup (?)
```

```
S_SEG ends
```

```
    end    start
```

4. Написать, откомпилировать и отладить программу, аналогичную программе п.3, но с использованием функций MS-DOS. Сделать варианты программы для всех функций.

5. Написать, откомпилировать и отладить программу, позволяющую ввести строку с клавиатуры с использованием функции MS-DOS 0Ah и затем вывести ее в обратном порядке.

6. Изменить программу п.3 (или п.4) таким образом, чтобы она автоматически завершалась по истечении заданного преподавателем интервала времени. Отсчет времени работы реализовать на базе функции 0h прерывания 1Ah BIOS.

7. Набрать в любом текстовом редакторе примеры программ, приведенных ниже. Получить EXE - и COM-модули;

Краткие теоретические сведения:

Программа на языке ассемблера состоит из строк, имеющих следующий вид:

метка **команда/директивы операнды** **; комментарий**

Причем все эти поля необязательны. Метка может быть любой комбинацией букв английского алфавита, цифр и символов `_,$,@,?,.`, но цифра не может быть первым символом метки, а символы `,$,?` иногда имеют специальные значения и обычно не рекомендуются к использованию. Большие и маленькие буквы не распознаются, но различие можно включить, задав ту или иную опцию в командной строке. В поле команда может располагаться команда процессора, которая транслируется в командный код или директива, которая не приводит к появлению нового кода, а управляет работой самого ассемблера. Если метка располагается перед командой процессора, сразу после нее всегда ставится оператор: (двоеточие), который указывает

ассемблеру, что надо создать переменную с этим именем, содержащую адрес текущей команды

Программа на языке ассемблера представляет собой совокупность блоков памяти, называемых сегментами памяти. Каждый сегмент содержит совокупность предложений языка, каждое из которых занимает отдельную строку кода программы. Предложения бывают четырех видов:

- команды или инструкции, представляющие собой символические аналоги машинных команд;
- макрокоманды - оформляемые определенным образом предложения текста программы, замещаемые во время трансляции другими приложениями;
- директивы, являющиеся указанием транслятору ассемблера на выполнение некоторых действий;
- строки комментариев, содержащие любые символы.

В языке ассемблера существует 5 директив для определения данных:

- DB (define byte) – определяет переменную размером в 1 байт;
- DW (define word) – определяет переменную размером в 2 байта (слово);
- DD (define double word) – определяет переменную размером в 4 байта (двойное слово);
- DQ (define quad word) – определяет переменную размером в 8 байт (учетверённое слово);
- DT (define ten bytes) – определяет переменную размером в 10 байт.

Программы, создаваемые с помощью Ассемблеров для компьютеров под управлением Windows, бывают двух типов: COM и EXE.

Пример:

```
ORG 100h
```

```
MOV AH, 02h
```

```
MOV DL, 41h
```

```
INT 21h
```

```
INT 20h
```

```
RET
```

#make_COM# – 1-ая строка. Эта команда – специфическая для Etni8086. Она используется для определения типа создаваемого файла. В нашем случае это файл с расширением .COM.

ORG 100h – 2-ая строка. Эта команда устанавливает значение программного счетчика в 100h, потому что при загрузке COM-файла в память, DOS выделяет под блок данных PSP первые 256 байт (десятичное

число 256 равно шестнадцатеричному 100). Код программы располагается только после этого блока. Все программы, которые компилируются в файлы типа COM, должны начинаться с этой директивы.

MOV AH, 02h – 3-я строка. Инструкция (или команда) MOV помещает значение второго операнда в первый операнд. То есть значение 02h помещается в регистр AH. Для чего это делается? 02h – это ДОСовская функция, которая выводит символ на экран. Мы пишем программу для DOS, поэтому используем команды этой операционной системы (ОС). А записываем мы эту функцию (а точнее ее номер) именно в регистр AH, потому что прерывание 21h использует именно этот регистр.

MOV DL, 41h – 4-я строка. Код символа «А» заносится в регистр DL. Код символа «А» по стандарту ASCII – это 41h. **INT 21h** – 5-я строка. Это и есть то самое прерывание 21h – команда, которая вызывает системную функцию DOS, заданную в регистре AH (в нашем примере это функция 02h).

Команда INT 21h – основное средство взаимодействия программ с ОС.

INT 20h – 6-я строка. Это прерывание, которое сообщает операционной системе о выходе из программы и о передаче управления консольному приложению. Значит, при использовании INT 20h в нашем примере, управление будет передаваться программе Emu8086. А в том случае, если программа уже откомпилирована и запущена из ОС, то команда INT 20h вернет нас в ОС (например, в DOS). В принципе, в случае с Emu8086 эту команду можно было бы пропустить, так как эту же функцию выполняет команда RET, которая вставляется в исходный текст автоматически при создании нового файла по шаблону.

Пример программы COM.

Код программы	Комментарии
.model tiny	Модель памяти, используемая для COM
.code	Начало сегмента кода
org 100h	Начальное значение счетчика - 100h
start: MOV ah,9	Метка start располагается перед первой командой в программе и будет использоваться в директиве END, чтобы указать с какой команды начинается программа.

		Помещаем число 9 в регистр AH (номер функции DOS "вывод строки")
MOV	dx,offset message	Помещаем в регистр DX смещение метки MESSAGE относительно начала сегмента данных, который в нашем случае совпадает с сегментом кода.
int	21h	Вызов системной функции DOS (в нашем случае вызывается функция DOS номер 9, эта функция выводит строку от начала, адрес которого задается в регистрах DS:DX, до первого встречного символа \$)
ret		Завершение программы COM
message	db "Hello World!", ODh,OAh,'\$'	Строка для вывода, ODh-код возврата коретки,OAh - перевод строки. (аналогично \n в C++)
	end	start
		Конец программы

Для превращения программы в исполняемый файл сначала надо вызвать ассемблер, чтобы скомпилировать ее в объектный файл с именем hello.obj, набрав в командной строке следующую команду:

Для TASM:

```
tasm hello-1.asm
```

Для MASM:

```
ml /c hello-1.asm
```

Для WASM:

```
wasm hello-1.asm
```

Для получения файла HELLO.COM:

Для TASM:

```
tlink /t /x hello-1.obj
```

Для MASM:

```
link hello.obj,,NUL,,
```

```
exe2bin hello.exe hello.com
```

Для WASM:

wlink file hello.obj form DOS COM

Контрольные вопросы

1. Из чего состоит процесс ассемблирования программы?
2. Чем отличается трансляция от компоновки?
3. Можно ли в процессе ассемблирования обойтись без ключей?
4. Объясните назначение каждого из перечисленных файлов, получаемых при создании исполняемой программы на языке ассемблер *.asm, *.lst, *.map, *.exe, *.com?
5. Какую структуру имеет программа на языке ассемблера?
6. Объясните в чем разница между директивами .Code, .Data, .Stack?
7. Создание .com, .exe, их структурное отличие на примере сегментов памяти?