ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра вычислительных машин, комплексов, систем и сетей

Л.А. Надейкина

ПРОГРАММИРОВАНИЕ

Учебно-методическое пособие
по выполнению практических работ

для студентов I курса
направления 09.03.01
очной формы обучения

Москва
ИД Академии Жуковского
2018

УДК 004.42(07)
ББК 6Ф7.3
Н17

Рецензент:
Черкасова Н.И. – канд. физ.-мат. наук, доц. каф. ВМКСС

Надейкина Л.А.
Н17 Программирование [Текст] : учебно-методическое пособие по выполнению практических работ / Л.А. Надейкина – М. : ИД Академии Жуковского, 2018. – 36 с.

Данное учебно-методическое пособие издается в соответствии с рабочей программой учебной дисциплины «Программирование» по учебному плану для студентов I курса направления 09.03.01 очной формы обучения.

Рассмотрено и одобрено на заседании кафедры 15.02.2018 г. и методического совета 15.02.2018 г.

УДК 004.42(07)
ББК 6Ф7.3

В авторской редакции

Подписано в печать 26.04.2018 г.
Формат 60x84/16 Печ. л. 2,25 Усл. печ. л. 2,09
Заказ № 288/0403-УМП21 Тираж 30 экз.

Московский государственный технический университет ГА
125993, Москва, Кронштадтский бульвар, д. 20

Издательский дом Академии имени Н. Е. Жуковского
125167, Москва, 8-го Марта 4-я ул., д. 6А
Тел.: (495) 973-45-68
E-mail: zakaz@itsbook.ru

СОДЕРЖАНИЕ

1. ОРГАНИЗАЦИОННО-МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

1.1. Цель и задачи выполнения практических занятий

В соответствии с учебным планом студентов направления 09.03.01, рабочей программой дисциплины «Программирование» и изложенными в них требованиями к уровню подготовки выпускников для работы в организациях ГА студенты должны обладать практическими навыками разработки программного обеспечения с использованием различных парадигм программирования на языках высокого уровня.

Целью изучения дисциплины «Программирования» является формирование у студентов представлений о современном состоянии программирования, о языках программирования, методах и средствах для разработки программ различного уровня сложности.

Задачами практических занятий является закрепление студентами теоретического курса дисциплины и приобретение навыков программирования с использованием мультипарадигменного языка программирования С++.

1.2. Методические рекомендации

Особенностью данного пособия является его прикладная направленность, что способствует развитию у студентов навыков:

- разработки алгоритмов и программ числовой и нечисловой обработки данных;
- разработки алгоритмов и программы линейной, разветвляющей и циклической структуры;
- работы с алгоритмами быстрого поиска и сортировки данных;
- работы с многомерными статическими и динамическими массивами;
- работы с библиотеками функций, с библиотеками классов, со связанными динамическими объектами;
- использования типовых решений, библиотек программных модулей, шаблонов, классов при разработке программного обеспечения;
- программирования в современных системах программирования, в различных режимах;
- владения методологиями объектно-ориентированного и обобщенного программирования;
- работы с текстовыми, типизированными и бинарными файлами, используя прямой и последовательный доступ к данным;
- тестирования и отладки приложений, определения основных типов ошибок и методов их локализации;

2. СОДЕРЖАНИЕ ЗАНЯТИЙ

2.1. КОДИРОВАНИЕ ЧИСЛОВОЙ ИНФОРМАЦИИ В ПОЗИЦИОННЫХ СИСТЕМАХ СЧИСЛЕНИЯ. АРИФМЕТИЧЕСКИЕ ОПЕРАЦИИ В ПОЗИЦИОННЫХ СИСТЕМАХ СЧИСЛЕНИЯ (занятие 1)

2.1.1. Цель занятия

Формирование компетенции ОПК 2.1.6 в виде получения навыков использования компьютерного представление чисел в различных форматах, навыков выполнения операций с числами, оценки точности результаты, в зависимости от формы представления чисел.

2.1.2. Методические указания по теме

Система счисления – способ изображения чисел с помощью знаков или символов, имеющих определенные количественные **значения.** ***Символами могут быть цифры, буквы, значки.***

В *позиционных системах* счисления количественное значение каждой цифры зависит от ее места (позиции) в числе.

Каждая позиционная система счисления имеет определенный *алфавит* и *основание q*.

Основание системы счисление является важным понятием и определяет:

1) количество различных знаков, используемых для изображения числа в данной позиционной системе счисления;
2) во сколько раз значение цифр стоящих на соседних позициях отличаются друг от друга

Цифра, стоящая на i-ой позиции или правильно сказать на i-м разряде может иметь значение в пределах: $0 \leq a_i \leq q-1$,

Десятичная система - алфавит арабских цифр от 0 до 9 и основание 10.

Восьмеричная – восемь цифр от 0 до 7 и основание 8.

Шестнадцатеричная – десять цифр от 0 до 9 и шесть первых прописных или строчных букв латинского алфавита A (a), B (b), C(c), D(d), E(e), F(f) и основание 16.

Двоичная система – алфавит - две цифры 0 и 1 и основание 2

Примеры чисел, представленных в различных позиционных системах счисления: $975{,}48_{10}$, 34_8, $41D_{16}$, 10110_2.

Позиционный характер этих систем легко понять на примере развернутой формы записи одного из чисел:

$975{,}48_{10}=9x10^2+7x10^1+5+10^0+4x10^{-1}+8x10^{-2}$

В общем случае запись любого вещественного числа в позиционной системе счисления представляется в виде:

$A = \pm a_{n-1}\, a_{n-2} \ldots a_i \ldots a_1\, a_0, a_{-1}\, a_{-2} \ldots a_{-m}$,

где n - количество целых разрядов , m - дробных , n+m - разрядность числа.

Такая формальная запись числа в q-системе представляет собой разложение числа в ряд по степеням q:

$A_q = \pm a_{n-1} \cdot q^{n-1} + a_{n-2} \cdot q^{n-2} + \ldots + a_i \cdot q^i + \ldots + a_0 \cdot q^0 + a_{-1} \cdot q^{-1} + \ldots + a_{-m} \cdot q^{-m}$, где

A_q – само число в q системе счисления,

i – номер разряда, q^i - вес i-го разряда,

a_i – значение i-го разряда (a_i – одна из цифр данной системы счисления),

Допускается сжатая запись: $A_q = \pm \sum_{i=-m}^{n-1} a_i \cdot q^i$.

Перевод чисел из одной системы счисления в другую (перевод из p-системы -> q -систему)

- Алгоритм перевода числа из любой системы счисления (q) в десятичную осуществляется по формуле:

$$A_q = \pm \sum_{i=-m}^{n-1} a_i \cdot q^i = A_{10}$$

- Перевод в остальные системы счисления производится по следующим правилам. Для перевода вещественного числа из p-системы в q-систему, следует, отдельно переводить его целую и дробную части.

1). Перевод целой части. Число, представленное в ***p***-системе делится на основание той системы (***q***), в которую переводится число, а также делятся частные от деления до тех пор, пока частное не окажется меньше делителя (***q***). Значения последнего частного и полученные остатки, записанные в обратной последовательности, образуя целую часть числа в системе ***q***. При делении новое основание представляется в старой системе.

2). Для перевода дробной части числа необходимо умножать ее на новое основание ***q***. Затем, отбрасывая целую часть результата, продолжать процесс умножения до тех пор, пока дробная часть произведения не окажется равной нулю или не будет достигнута нужная точность дроби. Целые части произведений, записанные после запятой в прямой последовательности (начиная с первого), образуют дробную часть числа в системе ***q***.

Перевод смешанного числа 46,625 из десятичной в двоичную систему:

1) Переводим целую часть числа:

	Остаток
46:2=23	0
23:2=11	1
11:2=5	1
5:2=2	1
2:2=1	0
1	1

Запишем остатки, начиная с последнего - 101110, т.е. $46_{10}=101110_2$

2). Переводим дробную часть числа:

0,625 x 2=1,250
0,250 x 2=0,500
0,500 x 2=1,000

Запишем целые части произведений, начиная с первого – 0,101, т.е. $0,625_{10} = 0,101_2$. Результат перевода числа: $46,625_{10} = 101110,101_2$

Алгоритм перевода чисел из двоичной системы счисления в систему счисления с основанием 2^n

Для того чтобы записать смешанное двоичное число в системе счисления с основание $q=2^n$, нужно:

1). Целую часть данного двоичного числа разбить справа налево, а дробную – слева направо на группы по n цифр в каждой. Если в последних

левой и/или правой группах окажется меньше чем n разрядов, то их надо дополнить слева и/или справа нулями до нужного числа разрядов.

2). Рассмотреть каждую группу как n-разрядное двоичное число и записать ее соответствующей цифрой в системе счисления с основанием $q=2^n$.

Перевод двоичного числа $111100101,0111_2$ в восьмеричную систему ($8 = 2^3$):

Разбиваем целую и дробную части двоичного числа на триады и под каждой из них записываем соответствующую восьмеричную цифру:

111 100 101, 011 100
7 4 5 , 3 4

Результат перевода: $111100101,0111_2 = 745,34_8$

Алгоритм перевода чисел из систем счисления с основанием 2^n в двоичную систему

Для того чтобы смешанное число, представленное в системе счисления с основание $q=2^n$, перевести в двоичную систему, нужно каждую цифру этого числа заменить ее n-значным эквивалентом в двоичной системе счисления. *Рассмотрим перевод шестнадцатеричного числа $4AC,35_{16}$ в двоичную систему счисления ($16=2^4$).*

В соответствии с алгоритмом запишем:

4 A C , 3 5
0100 1010 1100 , 0011 0101

Результат перевода: $4AC,35_{16} = 10010101100,00110101_2$

Арифметические операции в двоичной системе счисления

Сложение производится согласно таблице сложения, которая для двоичных чисел имеет вид:

0 + 0 = 0 0 + 1 = 1
1 + 0 = 1 1 + 1 = 10

При сложении двух единиц происходит переполнение разряда и в данном разряде остается 0, а 1 переносится в следующий старший разряд. Примеры сложения двоичных чисел:

```
  1001      1101      1010011,111
+ 1010    + 1011    +   11001,110
------    ------    -------------
 10011     11000     1101101,101
```

Вычитание производится согласно таблице вычитания, которая для двоичных чисел имеет вид:

```
0 - 0 = 0
0 - 1 = 1
1 - 0 = 1
1 - 1 = 0
```

Во втором случае при вычитании из меньшего числа - (0) большего (1) производится заем из старшего разряда.

```
110011001      1010011,111
-   101111   -   11101,110
---------      -----------
101011010      110110,001
```

В основе ***умножения*** лежит таблица умножения одноразрядных двоичных чисел:

```
0 x 0 = 0
0 x 1 = 0
1 x 0 = 0
1 x 1 = 1
```

Умножение многоразрядных двоичных чисел происходит в соответствии с таблицей умножения по обычной схеме, применяемой в десятичной системе счисления с последовательным умножением множимого на цифры множителя.

```
        1  1  0
x           1  1
-----------------
        1  1  0
     1  1  0
-----------------
1    0  0  1  0
```

Операция ***деление*** производится по тем же правилам, как и деление в десятичной системе счисления. Очередная цифра частного может быть только нулем или единицей. Пример деления двоичных чисел:

```
 100101001|11011
- 11011    1011
  101000
-  11011
    11011
-   11011
      0
```

Арифметические операции в восьмеричной системе счисления

1) $475{,}5_8 + 31_8 = 526{,}5_8$ 2) $5721{,}40_8 - 4230{,}54_8 = 1470{,}64_8$

```
  475,5                5721,40
+ 31                 - 4230,54
  526,5                1470,64
```

3) $1740_8 \times 32{,}5_8 = 63462{,}52_8$ 4) $462{,}2_8 : 31_8 = 14{,}2_8$

```
   1740,2                462,2|31
x   32,5               - 31   14,2
  115412                 152
 +37004                - 144
 56406                    62
 63462,52               - 62
                          0
```

Компьютерное представление чисел

Информация в памяти ЭВМ записывается в форме цифрового двоичного кода. С этой целью ЭВМ содержит большое количество ячеек памяти и регистров для хранения двоичной информации.

В вычислительной технике используются два формата представления двоичных чисел:

- с фиксированной запятой (точкой);
- с плавающей запятой (точкой).

Формат с фиксированной запятой применяется к целым числам, формат с плавающей запятой - к вещественным (действительным) числам.

Представление целых чисел в формате с фиксированной запятой

Множество целых чисел, представимых в памяти ЭВМ, ограничено. Диапазон значений зависит от размера ячеек памяти, используемых для их хранения. Так в n-разрядной ячейке может храниться $\mathbf{2^n}$ различных значений *целых чисел*.

Целые числа без знака.

Итак, чтобы получить внутреннее представление целого числа А без знака, хранящегося в n-разрядном машинной ячейке, необходимо:

1) перевести число А в двоичную систему счисления;

2) полученный результат дополнить слева незначащими нулями до n разрядов.

Полученный код числа называется *прямым кодом*

Целые числа со знаком: обратный и дополнительный коды.

Для записи внутреннего представления целого число со знаком (-А) необходимо:

1) модуль числа записать в *прямом коде* в n двоичных разрядах;

2) получить *обратный код* числа, для этого значения всех бит инвертировать – все единицы заменить нулями, а нули заменить единицами;

3) к полученному обратному коду прибавить единицу, тем самым получить *дополнительный код* целого числа – форма хранения числа со знаком.

Внутреннее представление целого отрицательного числа -1607 в 16-разрядной ячейке запишется следующим образом:

Модуль $|-1607|=1607_{10}=11001000111_2$

1) прямой код модуля числа в 16-разрядной ячейке:
0000 0110 0100 0111

2) обратный код:
1111 1001 1011 1000

3) дополнительный код (результат прибавления 1):
1111 1001 1011 1001 – ***внутренне представление числа (-1607).***

Целые числа в формате с фиксированной точкой, представленные m разрядами:

Диапазон чисел без знака: $\mathbf{0 \le A \le (q^m-1)}$

Диапазон чисел со знаком: $\mathbf{-q^{m-1} \le A \le (q^{m-1}-1)}$

Представление вещественных чисел в формате с плавающей запятой

Форма с плавающей точкой использует представление вещественного числа А в виде произведения ***нормализованной мантиссы m*** на основание системы счисления ***q*** в некоторой целой степени ***p***, которую называют порядком: $\boldsymbol{A=m \times q^p}$

Например, числу 139,76 соответствует запись: $\boldsymbol{0,13976x10^3}$.

Здесь *m=0,13976 – нормализованная мантисса, цифровое значение числа, p=3 – порядок* числа *(целое), показатель степени, в которую надо возвести*

основание системы счисления q (10), чтобы умножив мантиссу на q^p, получить число с фиксированной точкой.

Мантисса в нормализованном представлении должна удовлетворять условию: $0.1 \le m \le 1-q^{-n}$, *где n –разрядность мантиссы,* то есть мантисса меньше единицы и первая значащая цифра - не ноль.

В разных типах ЭВМ применяются различные варианты представления чисел в форме с плавающей точкой. Для примера рассмотрим один из возможных.

Пусть в памяти компьютера вещественное число представляется в форме с плавающей точкой в двоичной системе счисления (q=2) и занимает ячейку размером 4 байта.

В ячейке должна содержаться следующая информация о числе: знак числа, порядок и значащие цифры мантиссы. Вот как эта информация располагается в ячейке:

Знак, маш. порядок	М А Н	Т И С	С А
1 байт	2 байт	3 байт	4 байт

В старшем бите 1-го байта хранится знак числа. В этом разряде 0 обозначает плюс, 1 – минус. Оставшиеся 7 бит первого байта содержат ***машинный порядок.*** В десятичной системе это соответствует диапазону значений от 0 до 127. Всего 128 значений.

Разумно эти 128 значений разделить поровну между положительными и отрицательными значениями порядка. В таком случае между машинным порядком и истинным (назовем его математическим) устанавливается следующее соответствие:

Машинный Порядок $\mathbf{M_q}$	0	1	2	3	…	64	65	…	125	126	127
Математический Порядок **q**	-64	-63	-62	-61	…	0	1	…	61	62	63

Если обозначить машинный порядок M_q, а математический q, то связь между ними выразится формулой: $\mathbf{M_q = q + 64}$

Итак, машинный порядок смещен относительно математического на 64 единицы и имеет только положительные значения. Полученная формула записана в десятичной системе счисления. В двоичной системе счисления формула имеет вид: $\mathbf{M_q = q + 1000000_2}$

При выполнении вычислений с плавающей точкой процессор — это смещение учитывает.

Алгоритм получения представления действительного числа в памяти:

1) Перевод модуля данного числа в двоичную систему счисления;

2)Запись полученного двоичного числа в нормализованном виде;

3) Определение машинного порядка с учетом смещения;

4) Учитывая знак заданного числа (0 – положительное; 1 – отрицательное), записать его представление в памяти ЭВМ.

Запишем внутреннее представление числа**139,76** в форме с плавающей точкой в 4-х байтовой ячейке:

1) Переведем десятичное **139,76** в двоичное число и запишем его **24**-значащими цифрами:

$139{,}76_{10} = 10001011{,}1100001010001111_2$

2) Запишем полученное двоичное число в форме нормализованного двоичного числа с плавающей точкой:

$10001011{,}1100001010001111_2 = 0{,}10001011110000101000111_2 \times 10^{1000}$,

где **$0{,}10001011110000101000111_2$**– мантисса;

10 – основание системы счисления **($2_{10}=10_2$); 1000** – порядок **($8_{10}=1000_2$).**

3) Определим машинный порядок:

$Mq_2 = 1000 + 1000000 = 1001000$

4) Запишем представление числа **139,76** в ячейке памяти:

01001000	**10001011**	**11000010**	**10001111**

Для того чтобы получить внутренне представление отрицательного числа $-139{,}76_{10}$ достаточно в полученном выше представлении заменить в разряде знака числа 0 на 1.

2.1.3. Задания для самостоятельного решения

Задание 1.

Переведите целые числа из десятичной системы счисления в двоичную, восьмеричную и шестнадцатеричную системы счисления:

а) 231 б) 564 в) 1023 г) 4096.

Задание 2.

Переведите в десятичную систему счисления числа:

10011101_2, 11001010110_2, 321_8, 2367_8, $3A_{16}$, $B14_{16}$, $4F04_{16}$.

Задание 3.

Переведите смешанные десятичные числа в двоичную, восьмеричную и шестнадцатеричную системы счисления, оставив пять знаков в дробной части нового числа: 21,5; 432,54; 678,333.

Задание 4.

Выполните арифметические операции:

а) сложите, вычтите, умножьте и разделите двоичные числа 11010101_2 и 1110_2;

б) $1100000011{,}011_2$ x $101010111{,}1_2$;

в) $1510{,}2_8 – 1230{,}54_8$;

г) $3B3{,}8_{16}+38B{,}4_{16}$;

Задание 5.

а) Получите двоичную форму внутреннего представления целых чисел 1689 и -1689 в 2-х байтовой ячейке.

б) Получите двоичную форму внутреннего представления действительных чисел 224,25 и -224,25 в формате с плавающей точкой в 4-х байтовой ячейке.

Задание 6.

Запишите в десятичной системе счисления целое число, если его дополнительный код 1000000110101110.

Контрольные вопросы

1. Что такое система счисления. Перевод чисел из одной системы счисления в другую.
2. Арифметические операции в позиционных системах счисления.
3. Компьютерное представление чисел
4. Прямой, обратный и дополнительный код числа.
5. Форма хранения чисел с фиксированной и с плавающей точкой.
6. Форма хранения целых чисел со знаком и без знака. Диапазоны чисел.
7. Представление вещественных чисел в формате с плавающей точкой.

2.2. АЛГОРИТМЫ ЛИНЕЙНОЙ СТРУКТУРЫ. ОПЕРАТОРЫ ИЗМЕНЕНИЯ ДАННЫХ. ВЫРАЖЕНИЯ (Занятие 2).

2.2.1. Цель занятия

Формирование компетенций (ОПК-2.3.1) в виде получения навыков разработки и отладки программ на алгоритмическом процедурном языке программирования высокого уровня. Формирование компетенции (ОПК-5.3.4), получая навыки работы с библиотеками функций и классов. Формирование компетенции (ПК-1.2.3) при получении опыта разработки алгоритмов и программ линейной структуры.

2.2.2. Методические указания по теме

В основе решения любой задачи лежит понятие алгоритма. ***Алгоритм – это конечная последовательность точно определенных элементарных действий для решения задачи при всех допустимых вариантах исходных данных задачи.***

Разработка алгоритма для ЭВМ включает в себя выделение этапов процесса обработки данных и представление их в определенной форме и последовательности, например, в виде схемы алгоритма. В схеме алгоритма этапы обработки представляются в виде структур алгоритма – графических элементов, соединенных линиями передачи управления. Каждому действию соответствует некоторая геометрическая фигура. Конфигурация графических элементов определена государственным стандартом (ГОСТ 19.701- 90 ЕСПД).

В алгоритмах линейной структуры этапы обработки данных располагаются строго последовательно. Разработанный алгоритм реализуется в виде программы для ЭВМ на одном из языков программирования.

Конструкции языка, в которых определены действия программы, называются операторами. По характеру действий различают два типа

операторов: операторы преобразования данных и операторы организации обработки данных.

Операторы присваивания, ввода и вывода данных, операторы вызова функций являются типичными операторами преобразования данных, и именно эти операторы используются при программировании линейных процессов вычислений.

Программа на С++ представляет собой совокупность функций, одна из которых главная, называемая ***main*** выполнение программы начинается и заканчивается именно в ней. Другие функции либо непосредственно, либо опосредовано вызываются в главной функции.

Определение любой функции, в том числе и главной в С++ состоит из заголовка и тела функции:

<тип возвращаемого функцией результата><имя> (список параметров)
{ тело функции – последовательность действий функции }

Оператор присваивания

Часто решение поставленной задачи представляет собой процесс формирования результатов из исходных данных. В программах данные фигурируют в виде программных объектов. Данные, которые не изменяются в процессе выполнения программы, называются константами. Данные, определенные в программе и изменяемые в процессе ее выполнения, называются переменными. Правила формирования новых значений в программе задаются с помощью *выражений.* А с помощью *оператора присваивания* переменные получают новые значения. Структура оператора присваивания: ***L–значение = выражение;***

L-значением называют любую форму обращения к именованной области оперативной памяти (ОП), значение которой может изменяться. Имя переменной – частный случай ***L-значения***. Будем пользоваться следующей формой оператора присваивания: **имя_ переменной = выражение;**

Оператор присваивания выполняет следующее действие: “выражение” правой части *вычисляется* и его значение (в двоичной форме) помещается в область памяти левостоящей переменной. Примеры использования операторов присваивания: ***i= 0; z=x+y; i= i+2;***

Выражения

Выражение – это правило получения нового значения. Выражения формируются из операндов, операций и круглых скобок:

- операнды – это константы, переменные и результаты функций;
- операции – действия над операндами, выполняются в соответствии с приоритетом и ассоциативностью данной операции.
- круглые скобки, как и в математических выражениях, задают порядок выполнения операций.

В зависимости от типов операндов и операций, выражения условно делят на *арифметические, логические и выражения над символами и строками*.

Рассмотрим арифметические и логические выражения.

Арифметическое выражение аналогично алгебраическому выражению математики. Операнды арифметических выражений: константы, переменные и результаты вызова функций арифметических типов.

Операции: **+ , - , * , / , % , = , *op* = , ++ , - -** (здесь ***op*** одна из операций + , - ,* , / , %) .

Символ **‘=’** в языке С++ означает бинарную операцию, у которой в выражении должно быть два операнда: левый – переменная и правый – выражение. Если **x** – имя переменной, то **x= 2.5+5.2** есть выражение со значением **7.7**. Например, в следующем операторе вывода будет выведено значение выражения 7.7 на экран и одновременно переменной **x** будет присвоено то же значение: **cout << (x=2.5+5.2);**

Когда в конце выражения с операцией присваивания помещен символ **‘;’** , это выражение становится оператором присваивания. То есть, **x= 2.5+5.2;** есть оператор простого присваивания переменной **x** значения **7.7.**

В языке С++ существует целый набор “составных операций присваивания”. Каждая из составных операций присваивания объединяет некоторую логическую или арифметическую операцию и собственно присваивание. Операция составного присваивания является основой для оператора составного присваивания: ***имя_ переменной op= выражение ;***

Например, операторы: **x*= 4; i+=2; z /= 4*i+x;** - эквивалентны операторам простого присваивания: **x = x*4; i = i+2; z = z / (4*i+x);**

Описания встроенных функций арифметических типов представлены в заголовочном файле cmath.h и подключаются к программе следующей директивой: **#include <cmath>**

Логическое выражение – синтаксическая конструкция для определения истинности или ложности каких-либо положений, элемент средств алгебры логики. Тип результата логической операции – булевский, результат может принимать только два значения: ***true (1)*** и ***false (0)***.

К *логическим* относят следующие операции:

1) *сравнения*:

<, <= меньше, меньше равно; >, >= больше, больше равно;
= = равно ; != не равно; первые четыре операции одного приоритета;
две последние операции имеют более низкий приоритет;

2) *логические операции над данными любых скалярных типов*:

! - логическое отрицание НЕ; && - логическое И; || - логическое ИЛИ; операции расположены в порядке убывания приоритета. Операнды любых скалярных типов преобразуются к логическим значениям по правилу: все значения, отличные от нуля трактуются как истина (1), если значение равно нулю, то это ложь (0).

3) *поразрядные (битовые) логические операции* :

~ - битовое инвертирование; & - поразрядное логическое умножение И ;
| - поразрядное логическое сложение; ^ - поразрядное исключающее ИЛИ;

4) *сдвиги* : << , >>- битовые сдвиги влево и вправо

Ввод – вывод данных

Директива **#include <iostream>** подключает к программе библиотеку ввода/вывода, построенную на механизме классов. ***cin*** – имя стандартного входного потока (по умолчанию связанного с клавиатурой); ***cout*** – имя стандартного выходного потока (по умолчанию связанного с экраном дисплея);

>> - операция извлечение данных из потока или операция ввода;

<< - операция вставки данных в поток или операция вывода;

Оператора ввода

cin >> L-значение ;

Частный случай L-значение – имя переменной: ***cin >> имя переменной;***

Из потока **cin** извлекается значение и помещается в оперативную память, выделенную переменной. Внутри ЭВМ данные хранятся в виде двоичных кодов, которые регламентированы для каждого типа данных. При вводе выполняется преобразование символов из потока (с клавиатуры) в двоичные коды внутреннего представления данных, при этом происходит автоматическое распознавание типов вводимых данных.

Оператора вывода

cout<< выражение ;

Из оперативной памяти извлекается значение выражения и помещается в выходной поток ***cout***. При этом происходит преобразование двоичных кодов типизированного значения выражения в последовательность символов алфавита, изображающих значение на внешнем устройстве.

Рассмотрим ряд задач с линейным алгоритмом.

Задача. Разработать программу линейной структуры расчет площади круга

```
#include <iostream>              // директива подключения библиотеки
using namespace std;
 float r ;                       // объявление глобальной переменной
 const  float   pi = 3.1416 ;    // объявление глобальной константы
 int main ( ) {                  // заголовок функции main
  float s;                            // объявление локальной переменной
  cout<<"Введите радиус окружности";     // оператор вывода данных
  cin>> r;                       // оператор ввода данных с клавиатуры
  s = pi*r*r;                    //оператор присваивания
  cout<<"Площадь круга равна :" << s;  //оператор вывода данных
  return 0;}
```

Задача. Дана формула для вычисления арифметического выражения:

$$\frac{a+y^{b}}{\left(e^{ay+1}-\sin^{3}(x)\right)\cdot\left(2{,}25\cdot 10^{2}-\frac{x\cdot y}{b}\right)}\ ;$$

Разработать программу линейной структуры для вычисления значения арифметического выражения с использованием операторов присваивания, ввода и вывода. Значения переменных x, y ввести с клавиатуры, а константам a и b задать следующие значения a = 0.89 , b= 7.56. Листинг программы:

```
#include <iostream>                //директивы
#include <cmath.>
using namespace std;
const double  a=0.89, b= 7.56;     // определение глобальных констант
  //----------------------- Главная функция-------------------------------------
  int main () {
    setlocale(LC_CTYPE, "Russian");
    double  x, y, z,   //определение локальных переменных
    cout<< "Введите переменные\nx="; // вывод на экран строковой константы
    cin>>x; cout<<"y="; cin>>y;            //ввод значения с клавиатуры
    z= (a+pow(y,b))/(exp(a*y+1)-pow(sin(x),3))/(2.25e+02-x*y/b);//оператор присваивания
    cout<<"\n\nРезультат выражения:\nz=" <<z;
    cout<<"\n\nРезультат с помощью выражения в операторе вывода :\nz="
    << (a+ pow(y,b))/(exp(a*y+1)-pow(sin(x),3))/(2.25e+02-x*y/b);
    system("pause");    return 0; }
```

Задача. Дан рисунок фигуры на плоскости с затемненной областью. Разработать программу для вычисления логического выражения такого, что значение выражения – истина, если координаты точки, вводимые с клавиатуры, попадают в затемненную область фигуры на рис. 1.

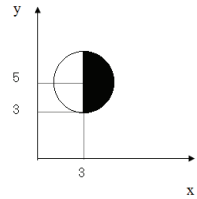


Рисунок 1. Фигура на плоскости.

Листинг программы:

```
#include <iostream>                //директивы
#include <cmath.>
using namespace std;
const double  a=0.89, b= 7.56;     // определение глобальных констант
 int main () {
    setlocale(LC_CTYPE, "Russian");
    double  x, y,    //определение локальных переменных для координат
    cout<< "Введите координаты\nx="; // вывод на экран строковой константы
    cin>>x; cout<<"y="; cin>>y;        //ввод значения координат с клавиатуры
       //Вычисление выражения в операторе вывода cout<<…;
    cout<<"\n\nЗначение выражения:\n"
    <<((pow(x-3,2)+pow(y-5,2)<=4) && (x >= 3));
     //Использование условной операции для вывода слов true или false
    ((pow(x-3,2)+pow(y-5,2)<=4) && (x >= 3))? cout<<" - true": cout<<"-false";
    system("pause");
    return 0; }
```

2.2.3. Задания для самостоятельного решения

Задание 1.

Запись арифметического выражения на языке С++

а) $\frac{0{,}23x+\cos^2(x+1)-1}{(b+c)e^{|\sin x|}}$, б) $\frac{arctg(x/z)-8{,}5\left|x+\sqrt[4]{y}\right|}{10^5+\lg 5}$, в) $\frac{tg(x^{2{,}5})}{1.3\cdot 10^4 e^{x+2}-\sqrt{|\sin x-\cos x|}}$.

Задание 2.

Разработать программу для вычисления логического выражения, которое включает ряд логических операций различного приоритета. Фрагмент такой программы:

а)
```
int  y = 1;
y+=3+1 > 5  &&  3+5>4  | |  4+5 >3;
cout << y;
```

б)
```
int y = 1 ;
y*= 3+1 > 5 && 3+5>4 || 4+5 >3 && 4+5 <3;
cout << y;
```

в)
```
int   B =6 , C=9 , D =B /C, A=1;
A*=( B % 2  ^  C / 2  &&  B ==C  || D ) ;
cout<<"A = " <<A
```

Расставить приоритеты операций и указать результат программы.

Задание 3.

Разработать программу, включающую фрагмент использования логических операций. Указать результат программы.

а)
```
int  x = 01 ,  y = 02, z =02;
cout << ( x & y )  <<' ' <<  ( x && y ) << ' ' << ( z – y && x ) << ' ' <<( z | x ) ;
```

б)
```
int z;   cout << ( z=3, z+2 )  << ' ';
cout << z << ' ';  cout << (4<<2) << '  ' << 20 / 3 << '  '<< ( 5>>1)  << '  ' << ( z &= 9 + 4 );
```

Задание 4.

Вычислить значения операций:

а) `5&8,  07&016,  011|0x12,  03^026 ,  014^ 12,  0x10 & 010$`

б) `!(-5),  !(5),  !(0),  ~077,  ~07,  15>>2,  07<<3,  -6<<3, 070>>3`

Контрольные вопросы

1. Алгоритм. Алгоритмы линейной структуры Свойства алгоритма.
2. Структура программы на С++. Функция **main.**
3. Константы и переменные. Основные типы данных.
4. Выражения. Арифметические выражения.
5. Логические выражения. Выражения с символами и строками.
6. Выражения и преобразования типов.
7. Операторы языка. Операторы преобразования данных.

2.3. ПРИМЕНЕНИЕ УСЛОВНЫХ ОПЕРАТОРОВ, ПРОСТЫХ И ВЛОЖЕННЫХ, ОПЕРАТОРОВ ВЫБОРА ВАРИАНТА (занятие 3)

2.3.1. Цель занятия

Формирование компетенции (ОПК-2.1.1) – получения навыков использования основных алгоритмических структур при разработке программ, компетенции (ОПК-2.3.1) в виде получения навыков разработки и отладки программ, реализующих алгоритмы ветвления и мульти ветвления. Формирование компетенции (ПК-1.2.3) при получении опыта разработки алгоритмов и программ разветвляющей структуры.

2.3.2. Методические указания по теме

На практике решение большинства задач не удается описать с помощью программ линейной структуры. Происходит нарушение естественного порядка выполнения операторов. Для этих целей используют ***управляющие операторы***. Так в алгоритме ветвления (или развилки), в зависимости от проверки некоторого условия выполняется та или иная последовательность операторов. Алгоритм имеет две формы, представленные на рис. 2:

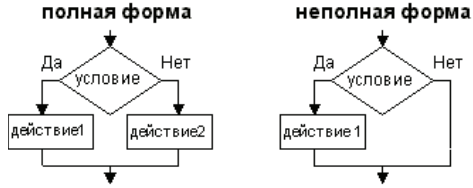


Рисунок 2. Схема алгоритма ветвления

Условный оператор используется для реализации алгоритма ветвления.
Полная форма: ***if (условное выражение) оператор 1; else оператор 2;***
Неполная форма: ***if (условное выражение) оператор;***

В качестве ***выражения*** может быть любое скалярное выражение, которое автоматически приводится к логическому типу. Каждый из ***операторов*** - по синтаксису один оператор простой или составной.

Если ***выражение*** истинно (то есть его значение не равно нулю), то выполняется ***оператор 1*** (прямая ветвь алгоритма), в противном случае выражение – ложно и выполняется ***оператор 2*** (альтернативная ветвь). Примеры условных операторов: ***if(x>0){x = -x; q (x);} else {int i=3; q(i*x);};***
if (a>b) a = -a;

Задача . Разработать программы нахождения минимального значения из двух значений вводимых с клавиатуры, используя полную и неполную форму условного оператора. Для простоты не рассматривать случай равенства значений. Листинг программы:

```
#include <iostream>
using namespace std;
  //----------------------- Полная форма---------------------------------------
  int main () {
    setlocale(LC_CTYPE, "Russian");
    double  m1, m2;   //определение локальных переменных для чисел
    cout<< "Введите два числа\nm1="; // вывод на экран строковой константы
    cin>>m1; cout<<"m2="; cin>>m2;  //ввод значения с клавиатуры
    cout<< Минимальное  число =";
    if (m1>m2) cout<< m2; else cout<< m1;
     system("pause");
    return 0; }
  //----------------------- Неполная форма---------------------------------------
  int main () {
```

```
setlocale(LC_CTYPE, "Russian");
double  m1, m2,min    //определение локальных переменных для координат
cout<< "Введите два числа\nm1="; // вывод на экран строковой константы
cin>>m1;  cout<<"m2="; cin>>m2;    //ввод значения с клавиатуры
min=m1;
if (m1>m2) min= m2;
cout<< Минимальное  число =" <<min;
system("pause");
return 0; }
```

Вложенные условные операторы

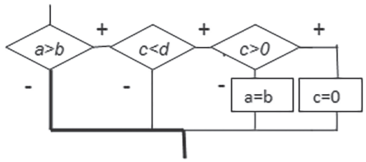
Если ***оператор 1*** или ***оператор 2*** в полной форме или ***оператор*** в неполной форме являются также условными операторами, то эти операторы называются вложенными условными операторами, которые также имеют прямую и альтернативную ветви. При определении, к какому условному оператору какая относится альтернативная ветвь, существует правило: рассматриваются слева направо каждый ***else.*** Очередная альтернативная ветвь (***else …***) принадлежит к ближайшему к ней, свободному (не связанному с другим ***else)*** оператору ***if*** . Пример:

if (x= =1) if (y= =1) cout<<'' x =1 и y =1 ''; else ;
else cout<<''x != 1'';

Задача. Пояснить текст программы и ее результат, используя схему алгоритма

а)

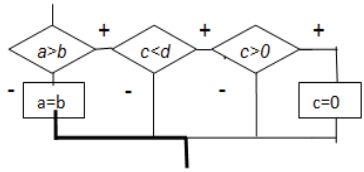
```
#include "stdafx.h"
#include <iostream>
using namespace std;
int  a=1, b=2, c=3, d=4 ;
#include <iostrem.>
 int main ( ) {
if ( a>b ) if (c<d )  if (c<0 )  c=0; else a=b;
 cout<<"a="<<a; return 0;  }
```



Результат: a=1

б)

```
#include <iostream>
using namespace std;
int  a=1, b=2, c=3, d=4 ;
#include <iostrem.>
 int main ( ) {
if ( a>b ) if (c<d )  if (c<0 )  c=0; else ;else; else a=b;
 cout<<"a="<<a; return 0;  }
```



Результат: a=2

Оператор switch реализует алгоритмическую схему мульти ветвления и имеет две формы – форма выбора варианта и форма переключателя. Алгоритм формы выбора варианта представлен на рис.3.

Форма альтернативного выбора варианта:

```
switch (выражение)
{ case константное варажение: операторы 1;
break;
…
case константное выражение N: операторы N;
break;
default: операторы;
break;
};
```

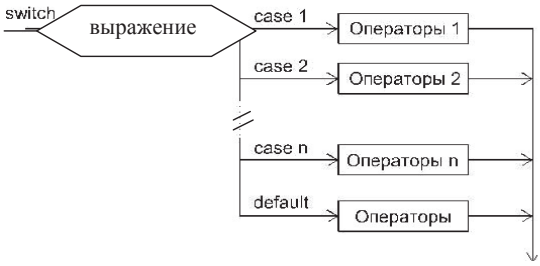


Рисунок 3. Алгоритмическая схема мульти ветвления – форма выбора варианта

Переключающее выражение может быть любого перечисляемого типа: целочисленного или символьного; ***константные выражения*** должны быть того же типа (или совместимого типа), что и переключающее выражение и различны по значению; для каждой ветви алгоритма возможно использовать несколько ***константных выражений***, например, ***case1 : case 5: операторы;***

Сначала вычисляется переключающее выражение. Полученное значение сравнивается со значениями константных выражений. Если значение ***выражения совпадет с одним из константных значений***, то выполняются операторы только данного варианта, после чего управление программой передается оператору, следующему за оператором ***switch.*** Если значение не

совпало ни с одним значением константных выражений, выполняется вариант с меткой ***default,*** если он имеется.

Обработка любого варианта может включать кроме операторов еще и описания, и определения объектов. В этом случае все это нужно заключить в фигурные скобки, тем самым превратить в блок.

Форма переключателя используется достаточно редко, и рассматриваться здесь не будет.

Задача . Разработать алгоритм и программу простейшего калькулятора.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
 int main () {
   setlocale(LC_CTYPE, "Russian");
   double  a, b;   //определение локальных переменных для чисел
   char c  ;  // знак операции
   cout<< "Введите два числа\n"; // вывод на экран строковой константы
   cin>>a>>b;;                                  //ввод значения с клавиатуры
   cout<< " Введите знак операции";
  m: cin>>c;
   switch (c)
    {  case '-' :  cout<< "a-b=" << a-b<<endl; break;
      case '+' :  cout<< "a+b=" <<a+b<<endl; break;
     ...
     default: goto m;
   }
    system("pause"); return 0;}
```

Задача. Вывести название дня недели, соответствующее заданному числу D, при условии, что в месяце 31 день и 1-е число — понедельник. Для решения воспользуемся операцией %, позволяющей вычислить остаток от деления двух чисел**.**

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int main (){ unsigned int D, R; //определены  целые положительные числа
cout<<"D="; cin>>D; R=D%7;
switch (R) {case 1: cout<<"Monday \n"; break;
            case 2: cout<<"Theusday \n"; break;
            case 3: cout<<"Wednesday \n"; break;
            case 4: cout<<"Thursday \n"; break;
            case 5: cout<<"Friday \n"; break;
            case 6: cout<<"Saturday \n"; break;
            case 0: cout<<"Sunday \n"; break; }
  system("pause");
  return 0;}
```

2.3.3. Задания для самостоятельного решения

Задание 1.

а). Составьте программу, вычисляющую корни квадратного уравнения $ax^2+bx+c=0$, в зависимости от коэффициентов уравнения.

б) Составить программу, которая переводит время из минут и секунд, заданных с клавиатуры, в секунды. В случае, если число отвечающие за секунды (например, 4.78), больше 60, программа выдает ошибку.

в) Написать программу вычисления площади кольца по его внешнему и внутреннему радиусу, заданному с клавиатуры.

г) Написать программу, рассчитывающую оптимальный вес пользователя (формула: рост — 100), и выводящую насколько ему нужно изменить настоящий вес для достижения идеала.

д) Ввести с клавиатуры 4 числа. Найти из них минимальное, используя три последовательных условных оператора, неполной формы. Разработать алгоритм и составить программу.

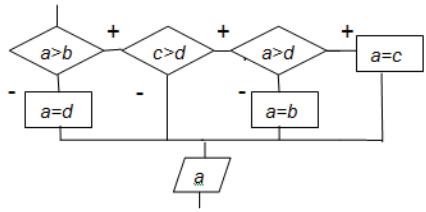
Задание 2.

Результат выполнения следующих операторов. Составить алгоритм и обозначить ветвь алгоритма выполнения программы..

а) *int m = 15/7 , n = 35%8 , p =5 , q =1;*
if (m<n) if (p>q) if (q>m) m =q ; else ; else m=n ; else m = p;
cout<<" m=" <<m;
б) int m, n , p, q;
m= 2; n = 7 % 4; p = m / n; q = n - m;
if(m<n)if (p>q) if (p< m) m =p ; else ; else m = q;
cout<<" m = " <<m;
в) int m = 2 , n = 3, t=4 , q =5;
if(m<n) if (t>q) m=q; else m=t; else if (t>n) m =0 ; else m=n ;
cout<<" m = " <<m;

Задание 3.

Написать программу, включающую реализацию алгоритма. Задать начальные значения всем переменным и получить результирующее значение переменной *a*:



Задание 4.

а) Разработать алгоритм и программу вычисления площадей круга, трапеции, прямоугольника при нажатии соответствующих клавиш русского алфавита ‘к’ или ‘К’, ‘т’ или ‘Т’, ‘п’ или ‘П’ соответственно. В каждом варианте ввести с клавиатуры необходимые данные для расчета.

б) Разработать алгоритм и написать программу меню обработки данных. Вывести строки меню с вариантами обработки и соответствующим значением пункта меню, по которому производить выбор данного варианта. Пользователь вводит значение номера с клавиатуры, программа производит соответствующие действия.

г) Объявить перечисляемый тип – «месяца года». Используя оператор switch и переменную перечисляемого типа, выводить время года, в зависимости от введенного номера месяца.

Контрольные вопросы

1. Алгоритмы ветвления. Алгоритм развилки
2. Условный оператор. Формы.
3. Вложенные условные операторы. Альтернативные ветви вложенных условных операторов.
4. Алгоритм выбора варианта. Оператор **switch;**
5. Переключающее выражение. Константы выбора.
6. Формы переключателя и выбора варианта.

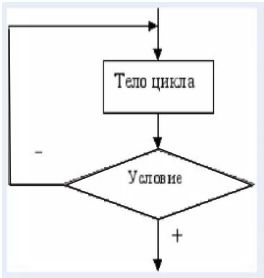
2.4. РЕАЛИЗАЦИЯ АЛГОРИТМОВ ПОВТОРЕНИЯ. ЦИКЛЫ С ПРЕД- И ПОСТУСЛОВИЕМ, ЦИКЛ С ПАРАМЕТРОМ (занятие 4)

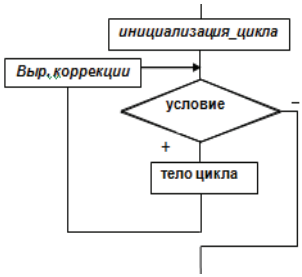
2.4.1. Цель занятия

Формирование компетенции (ОПК-2.1.1) – получения навыков использования основных алгоритмических структур при разработке программ. Формирование компетенции (ПК-1.2.3) при получении опыта разработки алгоритмов и программ циклической структуры.

2.4.2. Методические указания по теме

Циклы реализуют алгоритмическую схему повторения обработки данных. В С++ определены три разных цикла:





цикл с предусловием:
while (выражение)
тело_цикла

цикл с постусловие:
do тело_цикла
while (выражение);

цикл с параметром:
for (инициализация_цикла;
выражение-условие;
выражения коррекции)
тело_ цикла

Тело_цикла - это отдельный оператор, который всегда завершается точкой с запятой, либо составной оператор, либо блок. Операторы цикла задают многократное выполнение операторов тела цикла.

Выражение-условие – во всех операторах скалярного типа (чаще всего логическое или арифметическое) определяет условие продолжение повторения обработки, если оно истинно (отлично от нуля).

Прекращение выполнения цикла происходит в случаях:

- ложное (нулевое) значение ***выражения-условия;***
- выполнение в теле цикла оператора передачи управления (***break, goto, return***) за пределы тела цикла.

Оператор ***while- оператор цикла с предусловием.***

При входе в цикл вычисляется ***выражение-условие***, если оно отлично от нуля, выполняется ***тело_цикла.*** Затем вновь вычисляется выражение - *условие,* и выполнение ***тела_цикла*** будет повторяться, пока условие будет истинным.

Если в теле цикла не изменяется выражение - условие, могут возникнуть бесконечные циклы или "зацикливание".

Пример бесконечного цикла, с пустым оператором в теле: ***while (1);***

Такой цикл может быть закончен за счет событий, явно непредусмотренных в программе. Например, событие – указание от операционной системы "снять задачу".

Оператор ***do while*** является ***оператором цикла с постусловием.***

При входе в цикл выполняется ***тело_цикла.*** Затем проверяется ***выражение-условие*** и, если его значение истинно, вновь выполняется тело цикла и так далее. К выражению-условию требования те же: оно должно изменяться в итерациях за счет операторов тела циклов.

Цикл ***for – цикл с параметром.***

Заголовок цикла после зарезервированного слова ***for*** в круглых скобках включает три части: ***инициализацию_цикла, выражение-условие и список выражений для коррекции***, которые разделяются двумя знаками ***';'***.

Каждая из этих частей может отсутствовать, но даже если все они отсутствуют, два разделителя - ***';'*** должны присутствовать в заголовке.

Секция ***инициализация_цикла*** – это, как правило, определения объектов одного типа. Вычисляется один раз при входе в цикл. Как правило, определяются и инициализируются параметры цикла.

Выражение-условие - логическое выражение, проверяется на каждой итерации цикла. Если его нет, полагают, что оно истинно.

Выражения коррекции – список выражений, разделенных запятыми, которые вычисляются на каждой итерации после выполнения *тела_цикла* до следующей проверки условия.

Тело_цикла – может быть отдельным простым оператором, составным оператором или блоком.

При входе в цикл один раз вычисляются выражения из секции *инициализация_цикла*. Вычисляется значение *выражения-условия,* если оно истинно, выполняются операторы *тела_цикла* и затем вычисляются *выражения коррекции.* Далее цепочка повторяется, пока *выражение-условие* не станет ложным.

for – цикл с заданным числом повторений, для подсчета числа итераций используется управляющая переменная – *параметр цикла*.

Управляющая переменная может изменяться на любую требуемую величину: *for(c =0 ; c<=100 ; c +=10)…; for(b =100; b >= -100; b -= 25)…; for(let ='A' ; let <= 'Z'; let ++)…; for(pr =0.0 ; pr<= 100.0 ; pr +=0.5)…;*

Вложенные циклы

Разрешено вложение любых циклов в любые циклы. Действует следующее правило: для каждой итерации внешнего цикла выполняются все итерации внутреннего цикла.

Использование циклов с предусловием и с постусловием.

Задача. Разработать алгоритм и написать программу, нахождения суммы в некоторой точке X бесконечного ряда убывающих чисел до члена $<=10^{-8}$ используя цикл while, для вычисления очередного члена ряда использовать рекуррентную формулу, которая позволяет вычислить значения очередного члена ряда через предыдущий и получается из формулы: $a_{n+1} = f(x,n) \cdot a_n$;

Ряд : $\sum_{n=1}^{\infty}(-1)^{n+1}\frac{(X-1)^n}{n} = \frac{(X-1)^1}{1} - \frac{(X-1)^2}{2} + \frac{(X-1)^3}{3} - \ldots$ при $0 < X <= 2$;

Рекуррентная формула: $a_{n+1} = -a_n \cdot \frac{(x-1)}{(n+1)} \cdot n$

```
#include "stdafx.h"
#include <iostream>
using namespace std;
 int main () {
   setlocale(LC_CTYPE, "Russian");
   double  a, x, s=0 ;   //определение локальных переменных
   int n=1   ;  // номер члена
   cin>> x; a = x-1 // значение первого члена
   while (fabs(a)>1.e-8)
   {s+=a; a*=(-(x-1)*n/(n+1));
    n++;
   }
   cout<< s;
   system("pause"); return 0;}
```

Задача. Разработать алгоритм и написать программу, нахождения суммы бесконечного ряда убывающих чисел до члена $<=10^{-8}$ используя цикл ***do while***, а для вычисления очередного члена ряда использовать рекуррентную формулу;

Ряд : $\sum_{n=1}^{\infty}(-1)^n \frac{X^{2n-1}}{(2n-1)!} = -X + \frac{X^3}{3!} - \ldots$ в некоторой точке $|X|<\infty$.

Рекуррентная формула: $a_{n+1} = -a_n \cdot \frac{X^2}{(2n+1)\cdot(2n)}$

```
#include "stdafx.h"
#include <iostream>
using namespace std;
 int main () {
   setlocale(LC_CTYPE, "Russian");
   double  a, x, s=0 ;   //определение локальных переменных
   int n=1   ;  // номер члена
   cin>> x; a = - x // значение первого члена
   do { s+=a; a*=-(x*x/(2*n+1)/(2*n));
    n++;}
   while (fabs(a)>1.e-8);
   cout<< s;
   system("pause");
   return 0;}
```

Задача. Вывести степени 2 по семь значений в строке от нулевой степени до N-ой степени. Значение N ввести с клавиатуры.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
 int main () {
   setlocale(LC_CTYPE, "Russian");
   int N; cin>>N; // количество выводимых значений
   cout<<setw(10)<<"Степень"<<setw(10)<<"Значение"<<endl;
   for(int x=1, i=0; i<=N; i++,x*=2)
   { cout<<setw(10)<<i<<setw(10)<<x<<endl
    If ((i+1)%7)==0) cout<<endl;}
   system("pause");
   return 0;}
```

2.4.3. Задания для самостоятельного решения

Задание 1.

а). Разработать алгоритм и написать программу, нахождения суммы n членов, начиная с m-го, бесконечного ряда убывающих чисел в некоторой точке *X*, используя цикл ***for***, а для вычисления очередного члена ряда использовать рекуррентную формулу

Ряд : $\sum_{n=0}^{\infty}(-1)^{n+1}\cdot\frac{1}{(2n+1)\cdot X^{2n+1}} = -\frac{1}{X}+\frac{1}{3X^3}-\frac{1}{5X^5}+...$

Рекуррентную формулу вычислить по формуле: a_{n+1}/a_n

б) Вывести в два столбика номера и значения первых m членов ряда , m – ввести с клавиатуры.

Ряд: $\sum_{n=0}^{\infty}\frac{X^{2n+1}}{(2n+1)}$

Задание 2.

а) Вывести прописные буквы латинского алфавита от A до Z.

б) Даны натуральные числа от 1 до 33. Вывести все эти числа, кроме тех, которые делятся на три и оканчиваются тройкой.

г) Написать программу перевода десятичного натурального числа *X* в шестнадцатеричную систему. Результат вывести на экран.

д) Даны два натуральных числа. Вывести на экран значение наибольшего общего делителя (НОД) этих чисел (алгоритм Евклида). НОД – это максимальное число, на которое делятся без остатка оба исходных числа.

Задание 3.

Разработать алгоритм и программу вывода значений двух выражений x^2 и ***sqrt(x)*** одного аргумента в три столбика – значение аргумента и значения выражений. Использовать цикл ***while.*** Значения ***x*** меняются от 10 до 120 с равным шагом. Всего вывести 25 значений.

Контрольные вопросы

1. Алгоритмические схемы повторение обработки.
2. Реализация циклов в C++. Циклы с пред условием и с пост условием.
3. Цикл с параметром. Условия завершения цикла.
4. Что может быть параметром цикла.
5. Секции инициализации, условия, коррекции.
6. Вложенные циклы. Бесконечные циклы.

2.5. РЕАЛИЗАЦИЯ АЛГОРИТМОВ ПОВТОРЕНИЯ ДЛЯ ОБРАБОТКИ МАССИВОВ ДАННЫХ. ПРИМЕНЕНИЕ ЦИКЛИЧЕСКИХ ОПЕРАТОРОВ (Занятие 5).

2.5.1. Цель занятия

Формирование компетенции (ОПК-5.3.4) в виде получения навыков работы с библиотеками функций, с библиотеками классов, с динамическими объектами. Формирование компетенции (ПК-2.2.3) в виде получения навыков работы с многомерными статическими и динамическими массивами.

2.5.2. Методические указания по теме

Массивы

Массив–это совокупность данных одного типа, рассматриваемых как единое целое. Все элементы массива пронумерованы. Массив в целом характеризуется именем. Обращение к элементам массива выполняется по их номерам (индексам), которые всегда начинаются с 0.

Массивы могут содержать данные любых типов - простые, структурированные, стандартные, придуманные пользователем. Массив – это контейнер данных. Если для обращения к какому-то элементу массива достаточно одного индекса, массив называется *одномерным*. Массивы с числом индексов больше 1 называются *многомерными*.

Форма объявления многомерного массива

type имя массива [K 1] [K2] …[KN];

- type – тип элементов массива;

- N - размерность массива - количество индексов, необходимое для обозначения конкретного элемента;

- K1…KN - количество элементов в массиве по ***1… N***-му измерению, так в одномерном массиве ***K1*** – общее количество элементов, в двумерном массиве ***K1***– количество строк, а ***K2***– количество столбцов;

- Значения индексов по ***i***-му измерению могут изменяться от ***0*** до ***Ki – 1;***

- K1*K2*…*KN – размер массива (количество элементов массива).

Например, ***float Z[13][6];*** определяет двумерный массив, первый индекс которого изменяется от 0 до 12, а второй индекс - от 0 до 5.

Внутреннее представление массивов в оперативной памяти

При определении массива для его элементов выделяется участок памяти, размеры которого определяются количеством элементов массива и их типом: ***sizeof (type)* количество элементов массива,*** где ***sizeof(type) –*** количество байтов, выделяемое для одного элемента массива данного типа. Операция ***sizeof*** имеет две формы: ***sizeof(тип)*** и ***sizeof(объект).***

Учитывая это, а также то, что имя массива – это имя структурированной переменной, размер участка памяти, выделенного для всего массива, можно определить также из следующего выражения: ***sizeof (имя массива).***

В оперативной памяти все элементы массива располагаются подряд. Адреса элементов одномерных массивов увеличиваются от первого элемента к последнему. В многомерных массивах элементы следуют так, что при переходе от младших адресов к старшим наиболее быстро меняется крайний правый индекс массива. Так, при размещении двумерного массива в памяти сначала располагаются элементы первой строки, затем второй, третьей и т. д.

Инициализация- это задание начальных значений объектов программы при их определении, проводится на этапе компиляции. Инициализация одномерных массивов возможна двумя способами: либо с указанием размера массива в квадратных скобках, либо без явного указания:

```
int test [ 4] = { 10, 20 , 30, 40 };     // инициализация могла быть не полной
char ch [ ] = { 'a' , 'b' , 'c' , 'd' };// инициализация должна быть полной
```

Инициализация многомерных числовых массивов. Массив любой мерности в С++ рассматривается как одномерный. Рассмотрим инициализацию на примере трехмерного массива.

Инициализация его элементов может проводиться либо с учетом внутренней структуры массива, отделяя двумерные и одномерные массивы фигурными скобками, либо списком значений в соответствии с расположением элементов в оперативной памяти:

```
1)    int B [ 3 ] [ 2 ] [ 4] = { { {1, 2, 3, 4}, {5, 6, 7, 8} },{ {9, 10, 11, 12}, {13, 14, 15, 16} },
                                 { {17, 18, 19, 20}, {21, 22, 23, 24} } };
```

2) …= {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 21, 23, 24 };

Инициация символьного массива может быть выполнена значением строковой константы, например, следующим образом: *char stroka [10] = "строка";* При такой инициализации компилятор запишет в память символы строковой константы и добавит в конце двоичный ноль '\0', при этом памяти будет выделено с запасом (10 байт).

Можно объявлять символьный массив без указания его длины:
char stroka1 [ ] = "строка"; Компилятор, выделяя память для массива, определит его длину, которая в данном случае будет равна 7 байтов.

Инициализация двумерных символьных массивов проводится следующим образом: *char name [5] [18] = {"Иванов", "Петров", "Розенбаум", "Цой", "Григорьев"};*

При определении многомерных массивов с инициализаций (в том числе и двумерных) значение первого индекса в квадратных скобках можно опускать.

Количество элементов массива по первому измерению компилятор определит из списка инициализации. Например, при определении:

```
char sh [ ] [40] = {"====================================",
                    "|    F    |    F    |    F    |    F    |  F    |",
                    "====================================="};
```

компилятор определит три строки массива по 40 элементов каждая, причем *sh[0], sh[1], sh[2]* – адреса этих строк массива в оперативной памяти. В каждую из строк будет записана строковая константа из списка инициализации.

Для обращения к элементам массива используется имя массива, после которого в квадратных скобках стоит столько индексов, сколько измерений в массиве. В двумерном массиве: ***имя массива [i][j],*** - это обращение к элементу ***i –ой*** строке ***j-го*** столбца двумерного массива.

Ввод/вывод числовых массивов

Ввод/вывод значений арифметических массивов производится поэлементно. Для *одномерного массива* следует организовать цикл (повторение обработки данных), в каждой итерации которого, изменять индекс элемента и производить ввод (вывод) значения соответствующего элемента.

Пример иллюстрирует ввод элементов одномерного массива:

```
int test [100] ;
for (int i = 0; i< 100; i++) cin>> test [i];…
```

При вводе/выводе элементов двумерного массива, для обращения к элементам необходимо устанавливать номера строк и столбцов элементов.

При этом целесообразно, но не обязательно учитывать, как элементы массива располагаются в оперативной памяти. Внешний цикл следует организовать по номерам строк. В теле этого цикла для каждого номера строки организовать внутренний цикл, в котором перебирать номера столбцов.

Пример программы иллюстрирует вышесказанное:

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

```
#include <ctime>
using namespace std;
int main()
{ srand(time(0));//инициализация генератора случайных чисел
for(int i =0 ; i< 3; i++) { cout<<'\n';
for(int j=0 ; j< 4; j++) { T[i][j] = rand( );     cout<<T[i][j] <<"   ";}}
system("pause");
return 0; }
```

При вводе/выводе элементов многомерных массивов, количество циклов должно быть равно мерности массива. Внешний цикл – цикл перебора самого левого индекса, а самый вложенный цикл – цикл перебора правого индекса.

Ввод/вывод символьных массивов

Ввод и вывод символьных массивов можно производить поэлементно, как и числовых массивов, то есть рассматривать символьный массив как набор отдельных символов. Синтаксис языка С++ допускает также обращение к символьному массиву целиком по его имени, а именно по адресу этого массива в оперативной памяти. Объявим некоторый символьный массив: ***char text [80];***

Следующие операторы позволяют произвести ввод символьных строк:

1) ***cin>>text;-*** символы извлекаются из стандартного входного потока ***cin,*** и заносятся в оперативную память, по адресу ***text,*** ввод начинается от первого символа, отличного от пробела до первого обобщенного пробела.

2) ***cin.getline(text, n); -*** извлекаются из стандартного входного потока ***cin*** любые символы, включая и пробелы, и заносятся в оперативную память по адресу ***text.*** Ввод происходит до наступления одного из событий: прочитан ***n-1***символ или ранее появился символ перехода на новую строку ***'\n'.***

3) ***gets(text);*** - читается строка из стандартного потока и помещается по адресу ***text.*** Вводятся все символы до символа перехода на новую строку *'\n'* .

*4)****scanf("%s", text);*** – из стандартного потока читается строка символов до очередного пробела и вводит в массив ***text.***

Вывод строки позволяют произвести следующие операторы:

1) ***cout<<text;-*** выводит всю строку до байтового нуля в стандартный выходной поток ***cout,*** по умолчанию связанный с экраном дисплея.

*2)****puts(text);-*** выводит строку в стандартный поток и добавляет в завершении символ ***'\n'*** – перехода на новую строку.

3) ***printf("%s" ,text);-*** выводит в стандартный выходной поток всю строку;

Задача. Определить двумерный массив. Заполнить его случайными числами, найти минимальное и максимальное значение в массиве. Вывести элементы массива в виде матрицы

```
#include "stdafx.h"
#include <iostream>
#include <ctime>
using namespace std;
int main()
{ srand(time(0));
int T[7][8], min, max, r;
```

```
for(int i =0 ; i< 7; i++) { cout<<'\n';
for(int j=0 ; j< 8; j++) { T[i][j] = rand( );     cout<<T[i][j] <<"   ";}}
min=max=T[0][0];
for(int i =0 ; i< 7; i++) for(int j=0 ; j< 8; j++)
if (T[i][j]<min)min= T[i][j];
else if(T[i][j]>max) max= T[i][j];
cout<<'\n'<<min<< "  "  <<max;
system("pause");
return 0; }
```

Задача. Разработать алгоритм и программу. Определить массив с инициализацией. Сформировать новый массив из старого, но не включать в него элементы строки и столбца, номера которых ввести с клавиатуры.

Задача.

```
 #include "stdafx.h"
#include <iostream>
using namespace std;
const int m=10, n=7;
double A[m][n]={7,4.9,…,5.7}, B[m-1][n-1];
int main()
{ int i, j, l, k;         // номера индексов в старом  и новом  массивах
int  p, q;   //номера строки и столбца элементы которых надо не переносить
cin>>p>>q; //в новый массив, для простоты не проверяем допустимость значений
for(i=0,l=0; i<m; i++)
if(i!=p)
{  for(j=0,k=0; j<n; j++)
     If(j!=q)
   {B[l][k]=A[i][j]; k++;}
  l++;
}
return 0; }
```

Объявление и инициализация указателей

Указатель - это объект, значениями которого являются адреса участков памяти, выделенных для объектов конкретных типов.

type * имя указателя; *//* ***type -*** это тип объекта, на который "указывает" указатель, ***type ** – это тип указателя,

type * имя указателя = инициирующее выражение ;//инициализация

В качестве ***инициирующего выражения*** может быть:

1) явно заданный адрес участка памяти: ***type * имя = (type *) 0x158e0ffc;***

2) выражение, возвращающее адрес объекта с помощью операции ***'&'***:

type count ; type_1* iptr = (type_1*) &count;

3) указатель, имеющий значение:

char ch = '+', *R = &ch;   char *ptr = R;

4) инициирующее выражение равно пустому указателю:

nullptr – специальное обозначение указателя ни на что не указывающего.

char * ch (0)эквивалентно ***char * ch (nullptr).***

Операции над указателями

-Доступ к памяти обеспечивает операция разыменования.

Можно сказать, что выражение ***указатель –*** обладает правами переменной.

- Присваивание и преобразование типа. Если указатель ***p,*** типа ***char**** не обладает значением - адресом участка памяти, куда можно было занести значение, ему можно присвоить адрес участка памяти**:**

1) ***p = new char;*** //адрес участка памяти динамически выделенного на символ;

2) ***p = (char *) 0x157e0ffc;*** //значение адреса памяти преобразуется к указателю *char*;*

Теперь допустимы операции изменения памяти ***p ='+';*** ***cin>> *p;***

Пример работы с указателем на данные арифметического типа ***int***:

```
int n = 6, *pn = &n;
cout<<pn<<'\t'<< *pn;     //будет выведено:    0x1E10FFC      6
                                                //адрес          значение
```

Пример работы с указателем на данные символьного типа ***char***:

```
char c = 65, *pc= &c;
cout<<pc<<'\t'<< *pc<<'\t'<<&c;
```

Результат: ***A*** ***A*** ***A.*** То есть при выводе *указателя, разыменованного указателя и адреса* будет выводиться сам символ с кодом ***65*** – это прописная латинская буква ***A.***

Значение адреса символьной переменной и ее внутренний код получаем:

```
cout<< (void*)pc<<'\t'<< (int)*pc;
    //0x1e76a0c2      65
```

Выполнено явное приведение указатель к типу ***void***, и * указатель к типу ***int.***

Аддитивные операции

- *Разность однотипных указателей – это «расстояние» между двумя участками памяти, адресуемыми указателями, выраженное в единицах кратных длине объекта того типа, который адресуется указателями**:***

```
type*p1= …, *p2= …;      p1 – p2 = ((long) p1 – (long) p2) / sizeof(type)
```

Рассмотрим фрагмент программы:

```
{int a =1, b= 4, *aa = &a, *bb= &b;
cout<< aa <<'\t'<< bb <<endl;
cout<< (aa – bb) <<'\t'<<( (long)aa – (long) bb );}
```

Результат выполнения:

0x18e80fff 0x18e80ffb - адреса переменной *a* и переменной *b*

1 *4* – разность указателей (1) и разность значений указателей (4).

- *При вычитании из указателя целого числа: указатель – K;* формируется значение указателя меньшее на величину *K*sizeof(type),* где *type* - тип объекта.

- *При сложении указателя с целым числом**: указатель + K;*** формируется значение указателя большее на величину *K* sizeof(type).*

Пример:

```
{ int a=0, b=1, c = 3, d=5, *p = &d;
cout<< *p <<'\t'<< p <<'\t'<< *(p + 1) <<'\t'<< (p + 1) <<'\t';
cout<< *(p + 2) <<'\t' << (p + 2) <<'\t'<< *(p + 3) <<'\t' << (p + 3);}
```

Результат:
5 0x15e80fe4 3 0x15e80fe8 1 0x15e80fec 0 0x15e80ff0

- Инкремент ++ (декремент--) - увеличивает (уменьшает) значение указателя на *sizeof(type).*Указатель смещается к соседнему объекту с большим (меньшим) адресом.

- Обладая правами объекта, указатель имеет адрес, длину внутреннего представления и значение:
1) значения указателя – это адреса объектов, на которые "указывает" указатель;
2) адрес указателя получают операцией "взятия адреса": &имя указателя;
3) длина внутреннего представления
sizeof (void) == sizeof (char*) == sizeof (любой указатель) == 4байта*

Массивы и указатели

Имя массива - имя структурированной переменной. С другой стороны, имя массива - это константный указатель, значением которого является адрес первого элемента массива. Это свойство предоставляет еще одну возможность обращения к элементам массива, используя унарную операцию разыменования указателя. Пусть *T* – имя массива:
T[i] ==(T+i)*
*T[i][j] = = *(*(T+i)+j)==*(T[i]+j)==(*(T+i))[j]; и т, д*

Указатели и строки

Как и массивы типа ***char,*** указатели ***char**** могут инициализироваться строковыми константами:

*char * имя_1 = "строка"; char * имя_2 = {"строка"}; char * имя_3("строка");*

В операторе вывода *cout<< имя указателя на строку;* выведется сама строка.

Динамические массивы.

При статическом определении массива до конца программы, либо до конца блока массивам будут соответствовать участки памяти Память на массив можно выделять ***динамически*** в процессе работы программы, с помощью соответствующих операторов и также освобождать ее по желанию программиста. Рассмотрим этот случай.

type имя2 = new type [количество элементов массива]; - объявление одномерного динамического массива.*

Например,
*float*p1 = new float [10];* //динамический массив на *10* значений типа *float*

Доступ к элементам массива через указатель:
1) имя указателя [индекс элемента];
*2) *(имя указателя + индекс элемента);*
*3) *имя указателя++;* (операция допустима, так как указатель – переменная)

Пример фрагмента вывода элементов арифметического массива:
```
int A [10] = {7.3, 5.5, 1, … };
for (int i=0, *pA=A; i<10 ; i++)
cout<<*pA++<<"  ";
```
//что эквивалентно ***cout*<<*pA[i],*** или ***cout*<< *(pA+i),*** или ***cout*<<A[i].**

Массивы указателей

Как любые переменные указатели могут быть объединены в массивы. Пример массива указателей статической памяти: ***char*A [6];***

– одномерный массив указателей из 6 элементов – указателей на объекты типа ***char***, элементы массива **A** имеют тип ***char*.**

Выражение:

(long)(A+1)- (long) A = 4 (байта) **–** дает внутренний размер указателя.

По общему правилу можно определять одномерные динамические массивы указателей, учитывая, что тип элемента массива равен ***type*:***

type* * имя1 = new type* [количество указателей в массиве];

Создание двумерного динамического массива

Двумерный массив составляется из одномерных массивов строк, которые можно представить с помощью указателей и динамического выделения памяти.. Сколько строк в двумерном массиве – столько надо указателей для их определения. Надо объявить массив указателей на строки матрицы. Массив указателей на строки матрицы тоже является одномерным массивом, который можно определить динамически. Элемент массива указателей имеет тип ***type*,** где ***type*** **-** тип элементов исходной матрицы, поэтому массив указателей можно представить с помощью указателя на указатель ***type**.**

Пусть надо выделить память на динамическую матрицу элементов типа ***int***, размером: ***m***- строк по ***n*** элементов в строке (***m***-строк и ***n***– столбцов).

```
int m, n;
cin>>m>>n;          //размеры матрицы mxn введем с клавиатуры
int ** W;           //определим указатель на указатель
W = new int*[m];    // выделяем динамическую память на массив указателей
//В цикле выделяем динамическую память на одномерные массивы из n элементов
for (int i =0; i< m; i++) W[i] = new int [n];
```

Память выделена, к элементам матрицы можно обращаться с помощью стандартных выражений*:* ***W[i] [j]*** или ****(*(W+i)+j);***

После работы можно освободить память, причем сначала надо освободить память, выделенную на элементы строк матрицы, а затем освободить память, выделенную на массив указателей на строки матрицы:

```
for (int i =0; i< m; i++) delete [ ] W[i];      delete[ ] W;
```

Задача. Сформировать упорядоченный одномерный массив из элементов двух других массивов.

```
#include "stdafx.h"
#include <iostream>
using namespace std;
const int m=10, n=7;
int main ( ) {
int i, a [ ] = {7,9,5,4,0,2,89,33,73,11},b [ ] = { 23,87,55,45,4,3,0,6,7,3},
n= sizeof( a ) / sizeof ( a[0]), m= sizeof( b ) / sizeof ( b[0]);
int*x = new int [n+m]; // - выделена память на динамический массив
int i, j, p;
  for ( i=0; i < n+m; i++)     //формирование неупорядоченного массива
```

```
   if (i < n) x[i] = a[i];
   else x[i] = b[i-n];
 for (i = 0; i < n+m-1; i++)//сортировка “пузырьком”
   for (j = n+m - 1; j > i; j -- )
     if (x[j] < x[j-1])
     {p= x[j]; x[j] = x[j-1]; x[j-1] = p;}
for ( i =0; i < n+m; i++)
cout << x[i] << "  ";
delete [ ] x;
return 0;}
```

Задача. Выделить память на динамическую матрицу и заполнить ее случайными числами.

```
#include "stdafx.h"
#include <iostream>
#include <ctime>
using namespace std;
int main(){srand(time(0));
cin>>m>>n;
int** p = new int* [m];
for(int i =0 ; i< m; i++) p[i] = new int [n];
for(int i =0 ; i< m; i++)
for(int j=0 ; j< n j++)
p[i][j]=rand(); . . .
for (i=0; i< m; i++) delete p[i];   //освобождение памяти на значения в строках
delete [ ] p;   //освобождение памяти на массив указателей на строки
return 0;}
```

2.5.3. Задания для самостоятельного решения

Задание 1.

а). Объявить двумерный массив с инициализацией. Размеры массива задать как константы. Написать программу нахождения двух максимальных элементов массива и их индексов. Поменять в массиве местами столбцы, в которых находятся эти элементы.

б) Объявить двумерный массив, заполнить его случайными значениями. Вычислить среднеарифметическое значение массива. Составить новый массив из сумм элементов каждой строки исходного массива, которые больше среднеарифметического значения и количества таких элементов.

Задание 2.

а) Объявить одномерный символьный массив. Заполнить массив строкой с клавиатуры. Преобразовать символы строки в прописные. Инвертировать строку вывести значение строки на экран.

б) Ввести строку с клавиатуры. Задать две подстроки одинакового размера. Найти все точки вхождения первой подстроки в строку и заменить ее второй подстрокой.

Задание 3.

а) Определить трехмерный целочисленный массив. Заполнить его случайными числами. Сформировать и вывести одномерный динамический массив из всех элементов трехмерного массива кратных 3. Освободить память

б) Ввести с клавиатуры размеры двумерного массива, Выделить динамическую память на массив и заполнить матрицу числами с клавиатуры. Посчитать сумму и произведение всех отрицательных элементов матрицы. Освободить динамическую память.

в) Ввести строку с клавиатуры, включающую цифры, латинские буквы и другие символы. Сформировать новую строку из символов старой строки, выделив на нее динамическую память. Новая строка должна содержать только цифры и латинские буквы, сначала должны следовать цифры, а затем латинские буквы. Если таковых нет, следует вывести сообщение. Вывести на экран новую строку и освободить память.

Контрольные вопросы

1. Форматы определения числовых и символьных массивов.
2. Инициализация. Внутреннее представление
3. Форма обращения к элементам массива. Многомерные массивы.
4. Ввод – вывод элементов массивов (числовых и символьных).
5. Массивы и указатели. Определение указателей на массивы.
6. Указатели и строки. Способы ввода и присваивания строковых значений.
7. Массивы указателей. Массивы указателей на массивы строк.
8. Динамическое выделение памяти. Массивы динамической памяти.

Литература

1. Надейкина Л.А. Программирование на языке высокого уровня. Часть 1: учебное пособие – М.:МГТУ ГА, 2012, 84с.

2. Страуструп Б. Программирование: принципы и практика использования C++ 2-е издание, ISBN-13: 978-0321992789, Бином, Невский диалект, 2013,1312 с.

3. Прата С. Язык программирования С. Лекции и упражнения. ISBN: 978-5-8459-1950-2 (рус.), Вильямс, 2015, 928 с.

4. Подбельский В.В. Стандартный Си++. М.: Финансы и статистика, 2008, 688с.

5. Городня Л.В. Парадигмы программирования. М.: НОУ "Интуит", 2016, 278 с.