Московский государственный технический университет

имени Н.Э. Баумана

Факультет «Информатика и вычислительная техника»

Кафедра «Компьютерные системы и сети»

Т.Н. Ничушкина, А.В. Никитин, В.В. Гуренко

**Разработка программ с использованием сложных структурных типов данных**

Учебно-методическое пособие

Москва

(С) 2024 МГТУ им. Н.Э. БАУМАНА

УДК 004.432

Рецензент: доцент кафедры ИУ7 к.т.н. Новик Н.В..,

Т.Н. Ничушкина, В.В. Гуренко

Разработка программ с использованием сложных структурных типов данных. Учебно-методическое пособие по выполнению Домашнего задания № 2 первого семестра по дисциплине «Информатика» **-** М.: МГТУ имени Н.Э. Баумана, 2020. 32 с.

Учебное-методическое пособие содержат теоретические сведения об особенностях описания сложных структур данных, таких как массивы,иразработки программ с использованием этих структур на языке С++. Учебно-методическое пособиепредназначено для самостоятельной подготовки студентов к выполнению домашнего задания 2 модуля 2 дисциплины «Информатика».

Для студентов МГТУ имени Н.Э. Баумана, обучающихся по программе бакалавриата направления «Математика и компьютерные науки»

*Рекомендовано Научно-методическим советом МГТУ им. Н.Э. Баумана*

Ничушкина Татьяна Николаевна,

Гуренко Владимир Викторович

Разработка программ с использованием сложных структур данных.

**Учебно-методическое пособие по выполнению домашнего задания № 2первого семестра по дисциплине «Информатика»**

© Т.Н. Ничушкина, А.Н. Никитин, В.В. Гуренко 2024

© МГТУ имени Н.Э. Баумана, 2024

**Оглавление**

[**Предисловие 4**](#_Toc172574152)

[**1. Теоретическая часть 5**](#_Toc172574153)

[**1.1. Описание массивов 5**](#_Toc172574154)

[**1.2. Классификация задач по обработке массивов 6**](#_Toc172574155)

[**1.3. Приемы обработки одномерных массивов 7**](#_Toc172574156)

[**1.3.1. Операции над одномерными массивами 8**](#_Toc172574157)

[**1.3.2. Однотипная обработка массивов 11**](#_Toc172574158)

[**1.4. Программирование обработки двумерных массивов. 24**](#_Toc172574159)

[**1.5. Применение подпрограмм для обработки массивов. 32**](#_Toc172574160)

[**1.5.1. Особенности описания подпрограмм 32**](#_Toc172574161)

[**1.5.2. Передача массивов в подпрограммы 36**](#_Toc172574162)

[**1.5.3. Пример применения подпрограмм при обработке массивов 38**](#_Toc172574163)

[**1.6. Контрольные вопросы 41**](#_Toc172574164)

[**2. Практическая часть 42**](#_Toc172574165)

[**2.1. Порядок выполнения домашнего задания 42**](#_Toc172574166)

[**2.2. Требования к отчету 43**](#_Toc172574167)

[**2.3. Требования к защите 44**](#_Toc172574168)

[**2.4. Пример выполнения задания 44**](#_Toc172574169)

[**Литература 57**](#_Toc172574170)

[**Приложение А. Пример титульного листа ДЗ 58**](#_Toc172574171)

[**Приложение Б. Примеры вариантов задания 59**](#_Toc172574173)

# Предисловие

Настоящее учебно-методическое пособие предназначено для студентов 1 курса факультета ФН, изучающих дисциплину «Информатика» на кафедре ИУ-6 МГТУ им. Н.Э. Баумана в соответствии с программой подготовки бакалавров по направлению подготовки «Математика и компьютерные науки».

Цель учебно-методического пособия к домашнему заданию 2 «Разработка программ с использованием сложных структурных типов данных» модуля 2 по курсу «Информатика»– углубление знаний по использованию сложных структурных типов данных в и формирование навыков разработки программ с использованием этих данных, а также освоение приемов программирования задач с использованием массивов В языке С++ .

В учебно-методическом пособии представлены основные структурные типы данных, без изучения которых знания, умения и навыки в области программирования не будут отличаться полнотой. На примере обработки одномерных и двумерных массивов рассмотрены основные типы задач обработки этих структур данных и некоторые наиболее известные приемы и алгоритмы их решения. Приведены как сами алгоритмы, так и их реализация на языке С++.

Основная задача, которую автор ставил перед собой – помочь студентам в освоении приемов обработки сложных структур данных, которые непосредственно востребуются как при выполнении домашнего задания указанного модуля названной учебной дисциплине, так и в практической работе по некоторым математическим дисциплинам, таким как, например, «Вычислительная математика» и «Численные методы».

# Теоретическая часть

**Цель домашнего задания**: изучение особенностей реализации сложных структурных типов данных в языке С++ и разработки программ с использованием этих данных, а также освоение приемов программирования задач с использованием массивов.

Объем работы: 20 часов.

*Сложными структурными типами данных* называются такие данные, которые в памяти компьютера определенным образом организованы и имеют средства для работы с такой организованной информацией. К таким типам данных относятся массивы, строки, структуры, файлы.

## Описание массивов

***Массив*** *–* это упорядоченная совокупность однотипных данных. Каждому элементу массива соответствует один или несколько индексов, определяющих положение элемента в массиве.

Объявление массива:

**<Тип элемента><Имя>[<Размер1>] [<Размер2>] ...[= <Список значений >}];**

Количество индексов задает ***размерность*** массива.

***Тип индекса*** – порядковый – определяет доступ к элементу.

Нумерация индексов **ВСЕГДА** начинается с 0.

***Размер*** – определяет количество элементов по данному индексу.

***Тип элемента*** – любой кроме файла, в том числе, другой массив.

Массив в памяти не может занимать более 2 Гб.



Рисунок 1 –Примеры массивов

Работа с массивом сводится к действиям над его элементами. Для обращения к конкретному элементу массива необходимо указать имя массива и значение его индексов.

Именно с помощью индексов определяется положение элементов в массивах.

По правилам С/С++, имя массива является адресом его начала. Поэтому, с помощью индексов определяется смещение элемента относительно начала массива.

Так, для определения положения элемента одномерного массива достаточно одного индекса. Положение элемента определяется смещением на i\*sizeof (тип элемента).

где i – номер элемента в массиве.

Для матрицы необходимо два индекса. Смещение определяется как i\*m\*sizeof(тип элемента)+j\*sizeof(тип элемента),

где:

m – максимальное количество элементов в строке,

i – номер строки,

j – номер столбца.

## Классификация задач по обработке массивов

Все задачи по работе с массивами можно разбить на следующие классы:

1. Однотипная обработка массивов
2. Поэлементная обработка.
3. Выборочная обработка элементов массива
4. Переформирование массива
5. Без изменения исходных размеров массива
6. С изменением размеров исходного массива.
7. Одновременная обработка нескольких массивов или подмассивов.
8. С синхронным изменением индексов.
9. С произвольным изменением индексов
10. Поисковые задачи.

Выделение перечисленных классов обусловлено тем, что каждый из них имеет определенную методику обработки, получившую название приемов программирования данного класса задач. Как правило, в реальной жизни эти задачи в чистом виде встречаются довольно редко (в практикумах по программированию, лабораторных работах и домашних заданиях). Однако, любая сложная задача может быть разбита методом пошаговой детализации на более простые задачи, для которых уже известны приемы программирования. Кроме того, специфика реализации всех классов задач зависит от количества индексов массивов.

## Приемы обработки одномерных массивов

*Одномерными* называются массивы, в которых положение элемента в массиве определяется одним индексом. Описание массива может быть выполнено как вне функции (глобальные переменные), так и внутри любой функции (локальные переменные). Отличие – в области памяти для размещения и времени жизни массива.

Примеры определения одномерных массивов:

1. Статически на этапе компиляции

inta[10]– массив на 10 целых чисел;//индекс меняется от 0 до 9

floatmas[20]– массив на 20 вещественных чисел;

charsim[8]– массив на 8 символов;

double massiv[30]– массив на 30 вещественных чисел двойной точности;

unsigned int koord[10]– массив целых беззнаковых чисел.

1. Динамически с помощью *указателя* — это переменная, в которой хранится адрес некоторого объекта программы: другой переменной, поименованной константы и т. п.

В этом случае память под массив выделяется во время выполнения программы.

int \*dinmas; // объявление указателя на целое число

dinmas=new int[100]; // выделение памяти

// под массив на 100 элементов

...

delete[] dinmas;

Индекс меняется от 0 до величины, на 1 меньшей указанной в размере

Массив может быть инициализирован при объявлении:

int a[5]={0,-36,78,3789,50};

float b[8]={0f,-3.6f,7.8f,3.789f,5.0f,6.1f,0f,-6.5f};

long double c[4]={7.89L,6.98L,0.5L,56.8L};

short \*m={2,3,5,8,12,0,56};

### **Операции над одномерными массивами**

**1. Доступ к элементу массива:**

**Пример:**

int a[5],i;

...

a[0]=51; {прямой доступ}

...

i=3;

a[i]=3;{косвенный доступ: значения индексов находятся в

переменных}

Косвенный доступ позволяет реализовать последовательную обработку элементов массивов:

for(int i=1;i<6;i++)a[i]=i\*i;

или

for(int i=5;i>=0;i--) a[i]=i\*i;

**2. Ввод массивов.**

Не существует специального формата для ввода массива целиком. Ввод массива осуществляется поэлементно, с использованием циклов:

**Пример.** Ввод элементов одномерного массива

inta[5];//массив на **5** целых чисел

**...**

for(int i=0;i<5;i++)scanf(“%d”,&a[i]);

printf(“\n”); // вводит последнее Enter и очищает буфер

Значения вводятся через пробельный разделитель(пробел, Tab (→) или Enter(↵)):

**а) 2 -6 8 56 34 ↵**

**б) 2 ↵**

**-6 → 8 ↵**

**56 ↵**

**34 ↵**

**3. Вывод массива**

Также осуществляется поэлементно.

int b[7]={-3,5,8,-45,0,-1,8};

...

for(int j=0;j<7;j++)

printf("%4d", a[j]); { a0 a1 a2 a3 a4 a5 a6}

printf("\n");{переходим на следующую строку}

Результат вывода на экран представлен на рисунке 2.

∪∪-3∪∪∪5∪∪∪8∪-45∪∪∪0∪∪-1∪∪∪8

Рисунок 2 –Вид выведенного на экран массива

**Пример.** Написать программу, которая вводит и выводит массив *a* из восьми элементов. На рисунке 3 приведена схема алгоритма программы.

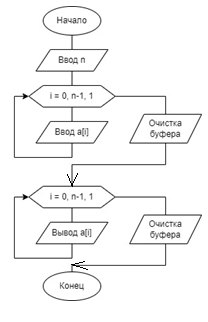


Рисунок 3 – Схема алгоритма ввода и выводамассива

**Текст программы ввода и вывода массива a:**

#include <stdio.h>

int i, n, a[8];

int main(int argc, char\* argv[])

{

puts("Input n<=8");

scanf("%d",&n);

printf("input %3d elementov \n",n);

for(i=0;i<n;i++) scanf("%d",&a[i]);

printf("\n");

puts("Inputed Massiv");

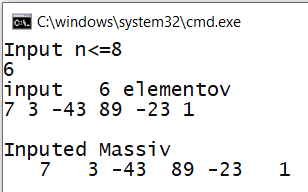
for(i=0;i<n;i++)printf("%4d",a[i]);

printf("\n");

return 0;

}

**Результат работы программы**

****

### **Однотипная обработка массивов**

a) **Поэлементная** (нахождение суммы элементов, произведения элементов, среднего арифметического, среднего геометрического, подсчет количества элементов, отвечающих определенному условию или обладающих некоторыми признаками, а также их суммы, произведения и т.д.).

**Пример**. Написать программу определения максимального элемента массива и его положения в массиве.Схема алгоритма программы приведена на рисунке 4.

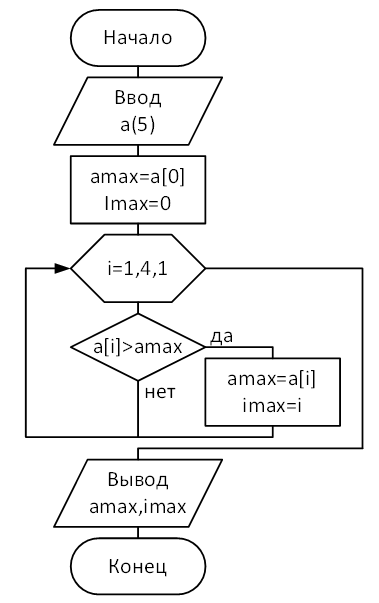


Рисунок 4 – Схема алгоритма программы

**Текст программы:**

#include <stdio.h>

int main(int argc, char\* argv[]){

float a[5], amax;

int i, imax;

puts("Input 5 values:");

for(i=0;i<5;i++)scanf("%f",&a[i]);

printf("\n");

amax=a[0];

imax=0;

for(i=1;i<5;i++)

if(a[i]>amax){

amax=a[i];

imax=i;

}

puts("Values:");

for(i=0;i<5;i++)printf("%7.2f ",a[i]);

printf("\n");

printf("Max = %7.2f number = %5d\n",amax, imax);

return 0;

}

**Результат работы программы**



b) **Выборочная** (задачи по формулировке сходные с задачами предыдущего типа, но операция выполняется не над всеми элементами массива, а только теми, которые имеют вполне определенное значение индексов).

Особенностью таких задач является наличие определенного закона изменения индексов рассматриваемых элементов**.**

**Пример.**Написать программу, определяющую количество отрицательных элементов среди элементов одномерного массива целых чисел, стоящих на четных местах. Решить задачу можно несколькими способами:

1. Счетный цикл

а) i = 1…n/2 с шагом 1

Номер элемента =i\*2-1

б) i = 1…n сшагом 2

Номер элемента =i

2. Цикл с постусловием

Номер элемента изменяет сам программист в цикле

Ниже приведена схема алгоритма и программа для варианта 1б.

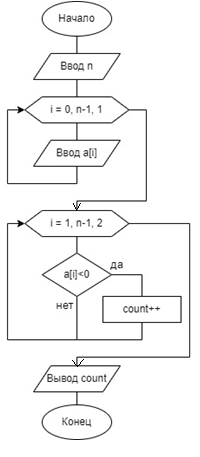


Рисунок 5 – Схема алгоритма выборочной обработки массива

**Текст программы**:

#include <stdio.h>

int main(int argc, char\* argv[])

{

    int a[10], i, count = 0, n;

    puts("Input n<=10");

    scanf("%d", &n);

    printf("Input %d values:\n", n);

    for(i=0; i<n; i++) scanf("%d", &a[i]);

    printf("\n");

    for (i=1; i<n; i+=2)

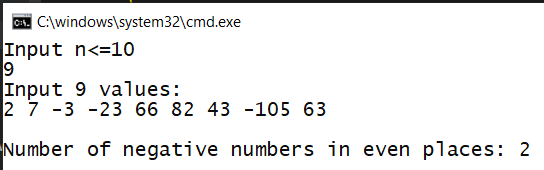
        if (a[i]<0) count++;

    printf("Number of negative numbers in even places: %d \n", count);

    return 0;

 }

**Результат работы программы**



Рассмотрим один из наиболее распространенных приемов программирования обработки одномерных массивов.

#### Сортировка массивов

*Сортировка* – это изменение положения элементов в массиве таким образом, что элементы становятся упорядоченными по некоторому закону. Для любого алгоритма оценивается вычислительная сложность, которая определяется зависимостью времени сортировки от количества сортируемых элементов. Чем меньше это время, тем более эффективен алгоритм. Кроме того, на практике интересны такие методы сортировки, которые позволяют экономно использовать оперативную память. Поэтому рассмотрим только такие алгоритмы, в которых не предполагается использование дополнительных массивов.

**Сортировка с помощью обмена.** Алгоритм прямого обмена основывается на сравнении и смене мест для пары соседних элементов. Сравнения продолжают до тех пор, пока не будут упорядочены все элементы. Такой метод широко известен под именем «пузырьковая сортировка», т. к. элементы массива можно рассматривать как пузырьки, которые поднимаются к началу массива.

Пусть, например, массив состоит из элементов (2, 5, 12, 1, 8, 4). Если сортировать массив по возрастанию, то после первого «прохода» по массиву мы получим следующий массив (2, 5, 1, 8, 4, 12). После второго «прохода» - (2, 1, 5, 4, 8, 12), после третьего – (1, 2, 4, 5, 8,12).

Анализ показывает, что сортировку можно прекратить, когда при очередном проходе не понадобилось выполнить ни одной перестановки. Кроме того, следует учесть, что после каждого прохода по меньшей мере один элемент становится на место, поэтому можно каждый раз сокращать количество просматриваемых элементов. Схема алгоритма сортировки с помощью обмена представлена на рисунке 6.

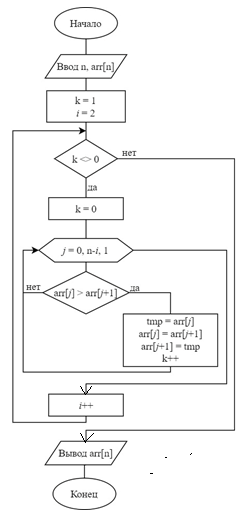


Рисунок 6– Схема алгоритма обменной сортировки

**Сортировка выбором** предполагает такую последовательность действий для сортировки по возрастанию:

* Начиная с элемента под индексом 0, ищем в массиве наименьшее значение.
* Найденное значение меняем местами с нулевым элементом.
* Повторяем шаги 1 и 2 для (n-1) элементов массива, начиная с элемента с индексом 1. Далее – для (n-2) элементов, начиная с индекса 2 и так далее, пока не останется один элемент.

Пример работы алгоритма: пусть дан массив (-4, 6, 5, 13, -7, 1), который необходимо отсортировать по возрастанию.

После первого прохода получим массив (-7, 6, 5, 13, -4, 1).

После второго – (-7, -4, 5, 13, 6, 1).

После третьего – (-7, -4, 1, 13, 6, 5).

После четвертого – (-7, -4, 1, 5, 6, 13).

Согласно алгоритму, во время пятого прохода нужно отсортировать элементы (6, 13), но так как они уже стоят правильно, мы «меняем» элемент 6 на самого себя, так как условие завершение алгоритма – осталось одно число для сортировки – еще не выполнено. Таким образом, алгоритм завершается после 5-й итерации – тогда для рассмотрения остается единственный элемент – (13).

Схема алгоритма показана на рисунке 7.

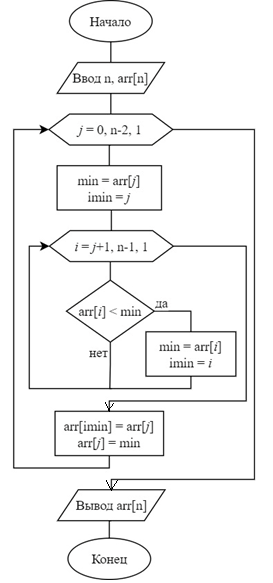


Рисунок 7 – Схема алгоритма сортировки выбором

**Сортировка вставками.**

В исходном состоянии считается, что сортируемая последовательность состоит из двух последовательностей: уже сортированной (на первом шаге она включает в себя в себя только первый элемент) и последовательности элементов, которую еще предстоит отсортировать.

На каждом шаге из несортированной последовательности извлекается элемент и вставляется в первую последовательность так, чтобы она оставалась сортированной. Поиск места вставки осуществляют с конца, сравнивая вставляемый элемент с очередным элементом сортированной последовательности . Если элемент больше , его вставляют вместо , иначе сдвигают вправо и уменьшают *j* на единицу. Поиск места вставки завершают, если элемент вставлен или достигнут левый конец массива. В последнем случае элемент вставляют на первое место. Наглядно сортировка вставками показана на рисунке 8 (голубым отмечена уже отсортированная последовательность элементов). Схема алгоритма показана на рисунке 9.

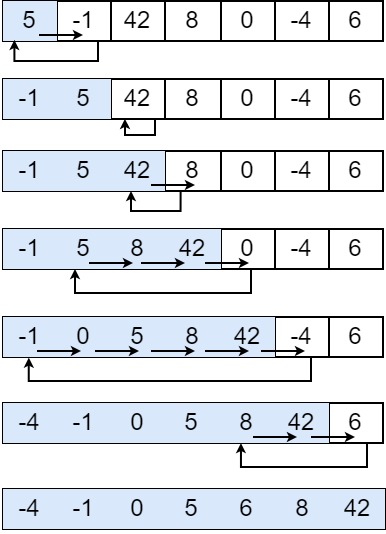


Рисунок 8 – Сортировка вставками

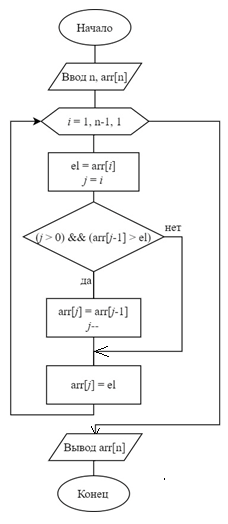


Рисунок 9 – Схема алгоритма сортировки вставками

**Шейкер-сортировка**является усовершенствованным методомпузырьковой сортировки.Анализируя метод пузырьковой сортировки, можно отметить два обстоятельства:

* если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, ее можно исключить из рассмотрения.
* при движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо.

Эти две идеи приводят к модификациям в методе пузырьковой сортировки.

* Массив просматривается поочереднослева направо и справа налево.
* От последней перестановки до конца (начала) массива находятся отсортированные элементы. Учитывая данный факт, просмотр осуществляется не до конца (начала) массива, а до конкретной позиции. Границы сортируемой части массива сдвигаются на 1 позицию на каждой итерации.
* Просмотр массива осуществляется до тех пор, пока все элементы не встанут в порядке возрастания (убывания).

Пример шейкерной сортировки показан на рисунке 10 (голубым отмечены элементы массива, которые считаются уже отсортированными и не участвуют в дальнейшей сортировке). Схема алгоритма показана на рисунке 11.

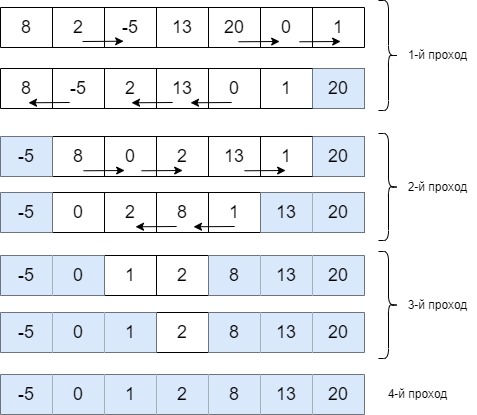


Рисунок 10 – Пример шейкерной сортировки

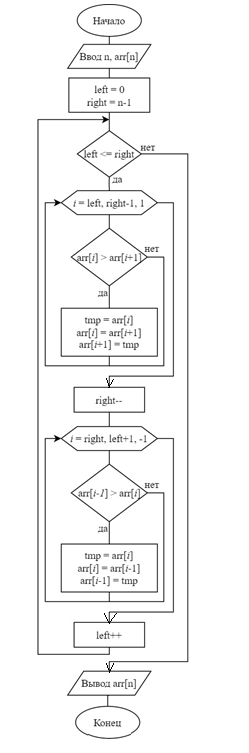


Рисунок 11 – Схема алгоритма шейкерной сортировки

**Гномья сортировка** интересна тем, что, в отличие от большинства методов сортировки, содержит всего лишь один цикл. Рассмотрим шаги алгоритма гномьей сортировки:

* Рассмотрение начинается со второго элемента массива (элемент с индексом 1 в C++). Затем происходит сравнение двух соседних элементов A[i] и A[i-1].
* Если те стоят на своих позициях, то увеличиваем индекс на единицу и продолжаем сравнение с нового положения;
* В ином случае меняем элементы местами, и, поскольку в какой-то момент может оказаться, что элементы в левом подмассиве стоят не на своих местах, уменьшаем индекс i на единицу, осуществляя следующее сравнение с новой позиции.

Так алгоритм продолжает выполняться до тех пор, пока весь массив не будет упорядочен.

Во-первых**,** процесс упорядочивания заканчивается, тогда, когда не выполняется условие , где n – размер массива.

Во-вторых**,** как было сказано, индекс как увеличивается, продвигаясь вперед по массиву, так и уменьшается, и в том случае если он окажется равен нулю, его сразу же следует увеличить на единицу, т. к. в противном случае придется сравнивать два элемента, одного из которых не существует.

Схема алгоритма показана на рисунке 12.

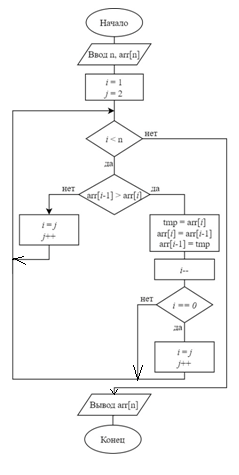


Рисунок 12 – Схема алгоритма гномьей сортировки

## Программирование обработки двумерных массивов.

***Матрицей*** называют двумерный массив, то есть, массив, для которого объявлены два индекса. Именно с их помощью определяется положение элемента в этом массиве. Один индекс обозначает строку, а второй – место элемента в этой строке.

***Объявление матрицы:***

**<Тип элемента><Имя>[<Размер1>][<Размер2>]...[={<Список значений** **>}];**

*Тип индекса* – порядковый – определяет доступ к элементу.

Нумерация индексов **ВСЕГДА** начинается с 0.

*Размер* – определяет количество элементов по данному индексу.

*Тип элемента* – любой кроме файла, в том числе, другой массив.

Массив в памяти не может занимать более 2 Гб.

***Описание матриц:***

1. Статически на этапе компиляции

inta[4][5] – матрица целого типа из 4 строк и 5 столбцов индексы меняются первый от 0 до 3, второй – от 0 до 4

floatmatr[10][20]– матрица вещественного типа из 10 строк и 20 столбцов

doublex[10][10]– матрица вещественного типа с двойной точностью из 10 строк и 10 столбцов

1. Динамически с помощью указателей.

Память под такую матрицу выделяется:

а) статически при инициализации

short \*\*r={{3,5,9,7},{1,4,7,0},{12,5,90,3}} – под массив будет выделена память по количеству определенных элементов (три строки по 4 элемента)

б) динамически во время выполнения

float \*\*c;

c=newfloat\* [4];

for (inti=0;i<4; i++) c[i]= newfloat[5]; – под массив будет выделена память 4 строки по 5 элементов.

Матрицы расположены в памяти “построчно”, т.е. правые индексы меняется быстрее, чем расположенные левее.

Например: Матрица А[4][3] в памяти выглядит так:

i=0

12

45

11

67

21

56

-13

90

-54

-87

44

i=1

i=2

i=3

0

2

1

0

1

2

2

2

0

0

1

1

j=0 - 2

0

Рисунок 13**-** Размещение матрицы в оперативной памяти

Если количество индексов превышает 2, массив называют *многомерным*.

Следует отметить, что программирование всех классов задач для матриц имеет свою специфику, основанную на том, что матрица, фактически, является массивом одномерных массивов. Это значит, что в каждом классе задач имеется гораздо больше различных вариантов решений, да и самих задач тоже. Особенно много сочетаний различных классов задач в одной конкретно поставленной задаче обработки некоторой матрицы. Хотя можно выделить круг задач, который точно укладывается в классификацию.

Рассмотрим наиболее распространенные приемы программирования обработки матриц.

***Поиск элементов матрицы, удовлетворяющих определенным условиям.***

Задано натуральное число . Найти сумму тех элементов целочисленной матрицы , значение которых больше .

Решение задачи заключается в простом поэлементном обходе по всем элементам матрицы и сравнения их с *k* и последующего добавления их к искомой сумме, если элемент соответствует условию задачи. Для этого понадобятся два счетных цикла, один из которых выбирает обрабатываемую строку, а второй – столбец, на пересечении которых находится элемент. Аналогично, при необходимости, можно обрабатывать матрицу не по строкам, а по столбцам, соответственно, во внешнем цикле будет меняться номер столбца, а во внутреннем – номер строки. Схема алгоритма показана на рисунке 14.

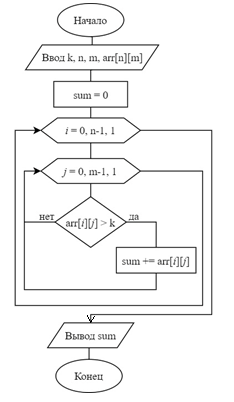


Рисунок 14 – Схема алгоритма нахождения суммы элементов матрицы

***Вычеркивание из матрицы столбцов, удовлетворяющих определенным условиям.***

Пусть дана матрица . Вычеркнуть из матрицы все столбцы, первый элемент которых положительный или равен нулю.

Задачу можно решить двумя способами.

*Первый способ*. Находим столбец, первый элемент которого положительный или равен нулю и сдвигаем все столбцы, стоящие за ним на одну позицию, затем затираем последний столбец и уменьшаем количество столбцов на 1. Так как на месте вычеркнутого столбца может оказаться столбец, у которого тоже первый элемент положительный или равен нулю, использовать для обхода столбцов счетный цикл нельзя. Менять номер анализируемого столбца может только программист и только тогда, когда вычеркивать столбец не нужно. Поэтому, для решения поставленной задачи, нужно использовать цикл с пред- или пост- условием.

*Второй способ*. Этот метод использует ту особенность задачи, что вычеркиваемыйстолбец не нужен, а вот остальные столбцы нужны. Будем переписывать только те столбцы, у которых первый элемент отрицателен. Для обхода матрицы по столбцам и для переписи нужного столбца можно использовать счетный цикл, а вот для обозначения места, куда следует переписывать нужный столбец, используем специальный индекс, например,k, который в самом начале равен 0 и увеличивается только при обнаружении нужного для переписи столбца. По окончании переписи именно в нем хранится реальное количество столбцов. Оставшиеся после выполнения переписи столбцы можно обнулить и новый размер m можно установить равный этому индексуk. Фрагмент схемы алгоритма представлен на рисунке 15.

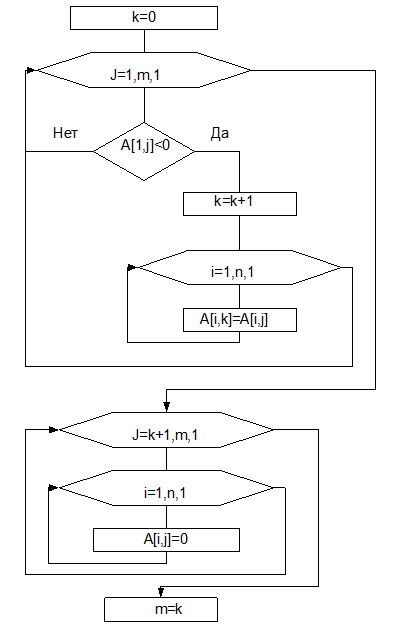


Рисунок 15 –Фрагмент схемы алгоритма вычеркивания столбцов

С учетом того, что индексы матрицы меняются от нуля, приведенный фрагмент на С++ будет выглядеть так:

...

k=0;

for(intj=0;j<m;j++) // переписываем нужный столбец

if(A[0][j]<0) {

for(int i=0;i<n;i++)

A[i][k]=A[i][j];

k++;

}

for(intj=k+1;j<m;j++) // очищаем оставшиеся ненужные столбцы

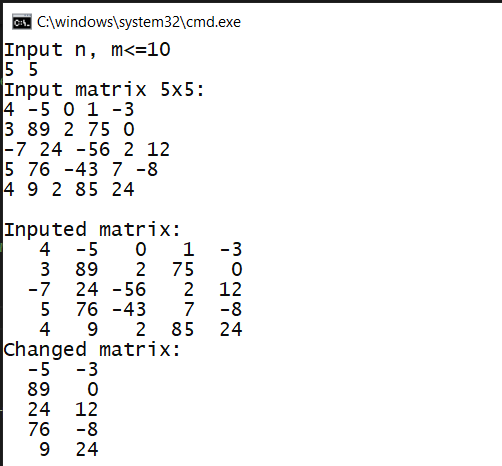
for(int i=0;i<n;i++)

A[i][j]=0;

m=k; // запоминаем новое количество столбцов

...

**Результат работы программы**



***Формирование массива, состоящего из элементов матрицы, соответствующих определенным условиям.***

Пусть дана матрица . Сформировать массив *sum*, содержащий сумму элементов тех строк, которые имеют нулевой элемент на пересечении с побочной диагональю. Вывести количество строк, удовлетворяющих условию, и массив *sum*. В случае, если подходящих строк нет, вывести соответствующее сообщение.

*Побочной диагональю* матрицы называется диагональ, проведённая из левого нижнего угла матрицы в правый верхний; закономерность формирования индексов для побочной диагонали:.

Из анализа условия задачи видно, что для решения понадобятся массив *sum*(n), счетчик подходящих строк count и, аналогично предыдущим задачам, два счетных цикла. Схему алгоритма для решения данной задачи можно увидеть на рисунке 16.

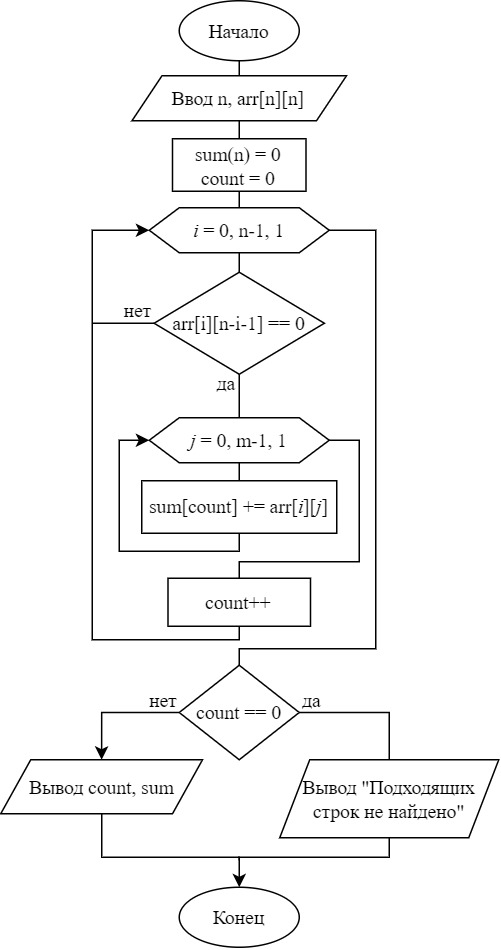


Рисунок 16 – Схема алгоритма формирования массива сумм определенных строк

Фрагмент программы, реализующей этот алгоритм:

...

for(int i=0; i<n; i++) sum[i] = 0;

count = 0;

for(int i=0; i<n; i++) {

    if (arr[i][n-i-1] == 0) {

        for(int j=0; j<n; j++)

            sum[count] += arr[i][j];

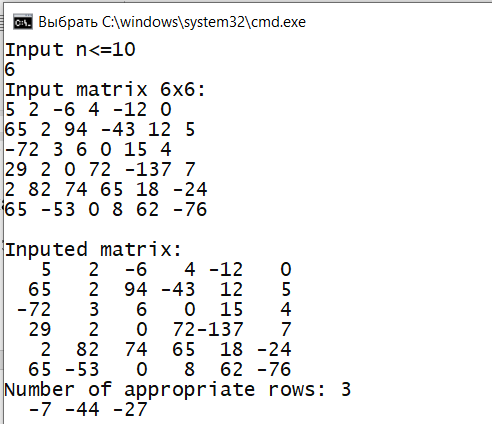
        count++;

    }

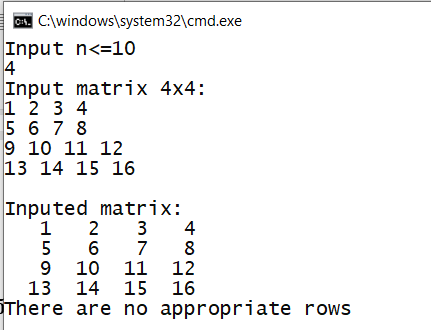
}

...

**Результат работы программы**



**Результат работы программы при отсутствии подходящих строк**



## Применение подпрограмм для обработки массивов.

Как показывает анализ задач по обработке массивов, очень много приемов можно применять как при работе с одномерными, так и с двумерными массивами. Кроме того, из программы в программу кочуют некоторые стандартные фрагменты, отличающиеся только размерами обрабатываемых массивов. С целью упрощения работы с такими структурами, очень полезно оформлять подобные фрагменты в виде подпрограмм.

### **Особенности описания подпрограмм**

***Подпрограмма*** – это относительно самостоятельный фрагмент алгоритма, соответствующим образом оформленный и снабженный именем.

В зависимости от способа описания, формирования результата и вызова, известны подпрограммы двух видов: процедуры и функции.

*Процедуры* предназначены для выполнения некоторых действий (например, печать строки), а *функция* позволяет получить некоторую величину, которую возвращает в качестве результата.

Однако, принципы программирования С\С++ основаны на понятии функции. Поэтому, в С\С++ нет процедур как элементов языка, однако средства языка позволяют создавать функции, которые не возвращают значения и реализуют конструкцию, аналогичную процедурам.

Данные для обработки процедуры и функции получают из вызвавшей их основной программы или подпрограммы. Для размещения рабочих полей подпрограммы могут объявлять новые типы и переменные в собственном разделе описаний. Результаты же они обычно должны возвращать вызвавшей программе или подпрограмме.

Синтаксис описания функции приведен ниже:

**<Тип результата><Имя > ([<Список параметров>])**

**{ [< Объявление локальных переменных и констант >]**

**<Операторы> }**

По правилам С\С++ подпрограмму можно описывать в любом месте программы и даже в другом файле, но только не внутри другой функции.

При описании функции после функции main или другой функции, в которой она используется, необходимо в начале программы описать прототип этой функции или подключить файл с описанием прототипа. Прототип представляет собой заголовок функции, завершающийся точкой с запятой. Обычно все прототипы функций собирают в начале программы.

Подпрограмма может получать данные двумя способами:

а) неявно – с использованием глобальных переменных;

б) явно – через параметры.

*Неявная* передача:

1) приводит к большому количеству ошибок;

2) жестко связывает подпрограмму и данные.

При использовании *явной* передачи данных через параметры существует два способа передачи аргументов в подпрограммы. Первый из них известен как *передача параметров по значению*. В этом случае в подпрограмму передаются копии фактических параметров, и никакие изменения этих копий не возвращаются в вызывающую программу.

Второй способ называют *передачей параметров по адресу*. При использовании этого метода в подпрограмму передаются адреса фактических параметров, соответственно, все изменения этих параметров в подпрограмме происходят с переменными основной программы. По умолчанию в языке С\С++ применяется передача по значению.

***Формальными*** называются параметры, определенные в заголовке функции при ее описании.

Каждый формальный параметр не только перечисляется (именуется), но и специфицируется (для него задается тип).

Совокупность формальных параметров определяет сигнатуру функции.

Сигнатура функции зависит от количества параметров, их типа и порядка размещения в спецификации формальных параметров.

При отсутствии параметров список можно заменить служебным словом voidили опустить, например:

intprint(void);

intprint();

***Фактическими*** называются параметры, задаваемые при вызове функции.

Формальные и фактические параметры должны совпадать:

* по количеству;
* по типу;
* по порядку следования.

Однако, имена формальных и фактических параметров могут не совпадать.

Например, ниже приведен отрывок программы, в которой можно увидеть различие между формальными и фактическими параметрами:

void print\_something(int n, char a, doubled){

// n, a, d– **формальные** параметры функции print\_something

printf("%d\n", n);

printf("%c\n",a);

printf("%6.2lf\n", d);

}

int main(){

int number = 72;

char sym = 'E';

doubledec = 76.80;

//number, sym, dec – **фактические** параметры функции (аргументы)

//правильныйвызов, аргументы – переменные подходящих типов

print\_something(number, sym, dec);

//правильныйвызов, аргументы – литералы

print\_something(4, 'h', 6.65);

print\_something(14.08, 7, 'y'); //**ошибка** в типах параметров

print\_something(2, 'r'); //**ошибка** в количестве параметров

return 0;

}

Существует два способа завершить выполнение подпрограммы и вернуть управление вызывающей подпрограмме. В первом случае функция полностью выполняет свои операторы и достигает закрывающей скобки, после чего управление автоматически передается в точку вызова. Такая подпрограмма должна либо не возвращать никаких данных, либо должна возвращать результаты через параметры. В этом случае необходимо использовать адреса переменных( *ссылки* или *указатели*).

Во втором случае для возврата в вызывающую программу или подпрограмму используется оператор **return [<выражение>]**. Значение предложения<выражение>возвращается в вызывающую функцию через специальные регистры. Если выражение пропущено, то возвращаемое значение функции не определено (в вызывающую программу вернется случайное значение – *мусор*). Функции типа void не могут указывать выражение в операторе return. Функции всех остальных типов должны указывать выражение в операторе return, по типу совпадающее с заявленным в заголовке типом результата. При необходимости функция может содержать несколько операторов возврата return, как показано в примере, приведенном ниже. Встретив любой оператор возврата, функция формирует возвращаемое значение и передает управление в конец подпрограммы, где и происходит возврат управления в вызывающую программу.

int max(int a,int b) {

if (a>b) return a;

elsereturnb;

a = a + b; //этот оператор никогда не выполнится, так как оператор

//return в любом случае передаст управление в вызывающую программу раньше

}

При использовании подпрограмм для работы с массивами, в нее необходимо в качестве параметра передать массив.

### **Передача массивов в подпрограммы**

***Передача в подпрограмму одномерного статического массива*.**

В данном случае в качестве формального параметра подпрограммы необходимо только указать, что данная подпрограмма принимает массив элементов некоторого типа. Размерность массива указывать не обязательно. Это вызвано тем, что С++ проверяет соответствие фактических и формальных параметров подпрограммы только по порядку и по типу, причем понятие типа при этом не включает конкретной размерности массива. Из этого следует, что помимо самого массива, в качестве формального параметра необходимо передавать его размер.

В качестве примера приведем программу построчной печати элементов массива, которая на вход принимает массив символов и размер массива, а в качестве результата выводит каждый символ на новой строке.

**Пример**. Передача в подпрограмму массива по значению (по имени)

#include <stdio.h>

#include <string.h>

void print\_arr(char a[], int n){

      for (int i=0; i<n; i++)

            printf("%c\n",a[i]);  
      printf("\n");

}

int main(){

      char arr[]="test test";

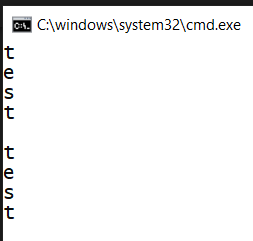
      int t=strlen(arr);

      print\_arr(arr,t);

      return 0;

}

**Результат работы программы**

****

***Передача в подпрограмму одномерного массива по ссылке.***

В данном случае необходимо указывать точный размер массива, иначе компилятор выдаст ошибку. А также важно взять в круглые скобки оператор &с именем массива — (&a). Доступ к элементам массива осуществляется так же, как и при использовании первого способа – посредством квадратных скобок.

**Пример.** Передача в подпрограмму массива по ссылке

#include<stdio.h>

void sa\_arr(float (&a)[10]){

double summ=0;

for (int i=0; i<10; i++)

summ+=a[i];

printf("Average: %.2f\n", summ/10);

}

int main(){

float arr[10];

for (int i=0; i<10; i++) {

puts("Input a number");

scanf("%f", &arr[i]);

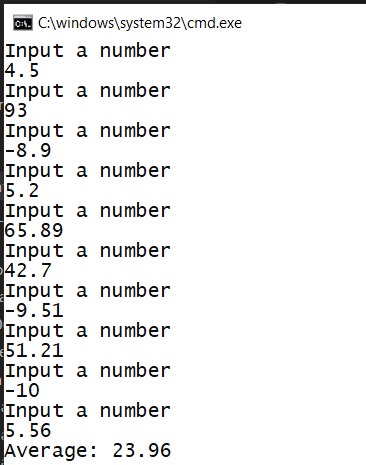
}

sa\_arr(arr);

return 0;

}

**Результат работы программы**

******

***Передача в подпрограмму одномерного массива по указателю.***

Пример передачи в подпрограмму по указателю приведен в лекциях.

### **Пример применения подпрограмм при обработке массивов**

Пусть необходимо решить следующую задачу.

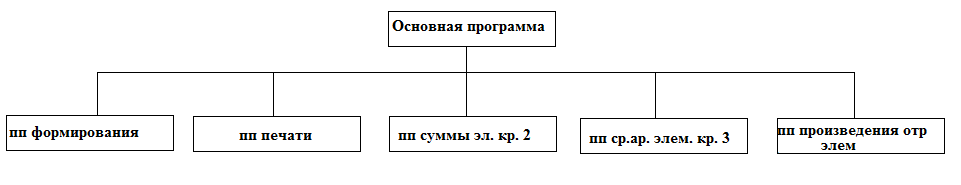
**Задание.** Сформировать одномерный массив целых чисел. В полученном массиве определить сумму четных элементов массива, среднее арифметическое его элементов, кратных трем и произведение его отрицательных элементов. Вывести на экран исходный массив и всю полученную информацию.

Как видно из условия, задачу можно декомпозировать на следующие подзадачи:

* Формирование массива.
* Печать массива.
* Нахождение суммы четных элементов.
* Нахождение среднего арифметического его элементов, кратных трем.
* Нахождение произведения отрицательных элементов.

Для работы с массивом можно использовать поэлементную обработку массивов для всех выделенных подзадач.

Таким образом, мы можем разбить задачу на 5 подпрограмм и основную, которая будет описывать данные и организовывать вывод и печать результатов. В качестве исходных параметров все подпрограммы получают массив и его размер. А так как 3 из 5 подпрограмм возвращают результат в виде числа, все их можно реализовать как функции. Только подпрограммы формирования и печати, не возвращающие явного результата, можно оформить как процедуры. В подпрограмму формирования с помощью генератора случайных чисел можно дополнительно передать еще и диапазон значений (a – нижняя граница, b – верхняя), что позволит гибко менять значения элементов исходного массива, моделируя разные наборы данных. А так как массив может быть до 100 элементов, в подпрограмме вывода предусмотрим вывод по 10 элементов массива в одной строке. Это можно реализовать как с помощью оператора ветвления, так и с помощью условной операции.

Структурная схема программы

Текст программы обработки приведен ниже.

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <time.h>

// подпрограмма печати массива

void printmas(int massiv[], int n)

{

int i;

for(i=0; i<n; i++)

if((i+1)%10 == 0) // организация печати массива по 10 элементов

printf("%5d\n", massiv[i]);

else

printf("%5d ", massiv[i]);

printf("\n");

}

// подпрограмма формирования массива

void formmas(int massiv[], int n, int a, int b)

{

int i;

for(i=0; i<n; i++)

massiv[i] = rand()%(b-a+1)+a;

}

// пп суммы четных элементов

int sumchet(int massiv[], int n)

{

int i, sum = 0;

puts("Elementy kratnye 2 : ");

for(i=0; i<n; i++)

if(massiv[i]%2 == 0)

{

sum = sum + massiv[i];

printf("%5d",massiv[i]);

}

printf("\n");

return sum;

}

// пп произведения отрицательных элементов

int proizotr(int massiv[], int n)

{

int i, p = 1; puts("Elementy <0 : ");

for(i=0; i<n; i++)

if(massiv[i]<0)

{

p = p\*massiv[i];

printf("%5d",massiv[i]);

}

printf("\n");

return p;

}

// пп среднего арифметического элементов, кратных трем

float sredkr3(int massiv[], int n)

{

int i, k = 0;

float sum = 0;

puts("Elementy kratnye 3 : ");

for(i=0; i<n; i++)

if(massiv[i]%3 == 0)

{

k++;

printf("%5d",massiv[i]);

sum = sum + massiv[i];

}

if(k!=0) sum = sum/k;

printf("\n");

return sum;

}

void main(){ // основная программа

int mas[100], i, j, n;

srand((unsigned)time(NULL));

puts("Input n<=100");

scanf("%d", &n);

formmas(mas, n, -5, 20);

puts("InputedMassiv");

printmas(mas, n);

printf("Summa chetnyx elementov =%6d\n", sumchet(mas, n));

int p = proizotr(mas, n);

if(p==1)

puts("otricatelnyx elementov v massive net");

else

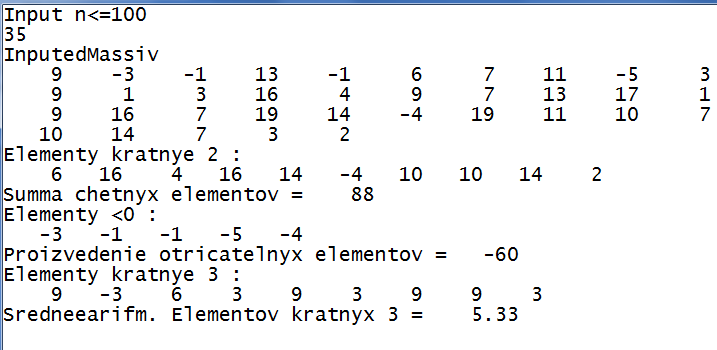
printf("Proizvedenie otricatelnyx elementov =%6d\n", p);

printf("Sredneearifm. Elementov kratnyx 3 =%8.2f\n", sredkr3(mas, n));

getch();

}

**Результаты работы программы**



## Контрольные вопросы

1. Что такое структурные типы данных?
2. Что такое массив и как определяется положение элементов в массиве?
3. Приведите основные методы сортировки одномерных массивов.
4. Как в памяти располагаются матрицы?
5. Какие задачи обработки матриц применимы только для матриц?
6. Что такое подпрограмма?
7. Как подпрограмма получает исходные данные?
8. Какие способы передачи одномерных массивов в подпрограммы вы знаете?
9. Что такое формальные и фактические параметры подпрограммы? В чем их отличие?

# Практическая часть

## Порядок выполнения домашнего задания

Домашнее задание №2 **«Разработка программ с использованием сложных структурных типов данных»** выдается на 6-ой неделе и включает разработку алгоритмов программ и кодирование программ задания 2.

Домашнее задание состоит из трех заданий.

**Первое задание** на сортировку одномерного массива заданным способом.

**Второе задание** на обработку матрицы.

**Третье задание** предполагает использование подпрограмм в обработке массивов.

Для выполнения каждого из заданий необходимо:

1. Прочитать и проанализировать задание в соответствии со своим вариантом.
2. Декомпозировать ее на подзадачи, если это необходимо.
3. Выбрать приемы обработки массивов для каждой подзадачи или фрагмента задачи.
4. Разработать алгоритм программы и, если необходимо, подпрограмм.
5. Разработать тесты для тестирования и отладки программы в соответствии с ее особенностями.
6. В среде программирования не ниже Visual Studio или QtCreator создать новый проект и закодировать программу в соответствие с разработанными алгоритмами.
7. Отладить программу.
8. Составить отчет по домашнему заданию.
9. Продемонстрировать работу программы преподавателю.
10. Защитить программу и отчет преподавателю.

## Требования к отчету

Все записи в отчете должны быть либо напечатаны на принтере, либо разборчиво выполнены от руки синей или черной ручкой (карандаш – не допускается). Схемы также должны быть выполнены при помощи программных средств разработки схем алгоритмов на компьютере или нарисованы с использованием чертежных инструментов, в том числе карандаша. Отчет можно оформлять как для всего задания сразу, так и отдельно для каждой части.

Отчет по домашнему заданию для каждой части должен содержать**:**

1. Цель задания
2. Текст задания.
3. Анализ и выбор приемов программирования массивов и строк.
4. Структурную схему, если необходимо.
5. Схемы алгоритмов всех подпрограмм.
6. Текст программы.
7. Таблицы тестов или скриншоты работы программы.
8. Пример работы программы.
9. Выводы.
10. Ответы на контрольные вопросы.

Кроме того, все отчеты должны иметь титульный лист, на котором указывается:

1. наименование факультета, кафедры и направления подготовки;
2. название дисциплины;
3. номер и тема домашнего задания;
4. номер варианта задания;
5. фамилия преподавателя, ведущего занятия;
6. фамилия, имя и номер группы студента.

Если сдаются все части одновременно, титульный лист может быть один на все части.

## Требования к защите

Для получения зачета по домашнему заданию, студент должен представить отчет, оформленный в соответствии с требованиями, продемонстрировать корректно работающие программы, внести требуемые преподавателем правки, если он считает программу не соответствующей требованиям и ответить на вопросы преподавателя по программе и теории.

Проверка домашнего задания может осуществляться поэтапно, по мере выполнения практических заданий. Окончательный срок завершения задания – 10 неделя 1-го семестра.

Студент считается выполнившим домашнее задание, если у него к этому моменту завершены, проверены преподавателем отчеты по выполнению указанных заданий, и задания защищены.

## Пример выполнения задания

**Часть 1**. **Обработка одномерных массивов.**

**Задание.** Сортировать одномерный массив латинских символов по алфавиту. Использовать метод выбора.

Компьютер, фактически, рассматривает символы как целые числа, то есть над ними можно совершать те же арифметические и логические операции, операции сравнения и т.д., однако интерпретируются они как коды по таблицеASCII. Таким образом, например, можно узнать, является ли символ заглавной буквой латинского алфавита путем обычного сравнения:

(ch>='A')&&(ch<='Z')

Как показал анализ задания, для выполнения поставленной задачи необходимо:

- ввести массив

- вывести массив

-отсортировать массив

Вывести полученный массив

Для ввода и вывода массива целесообразно использовать поэлементную обработку массивов, а для сортировки - переформирование массива без изменения его размеров, так как любые сортировки относятся к этому приему обработки.

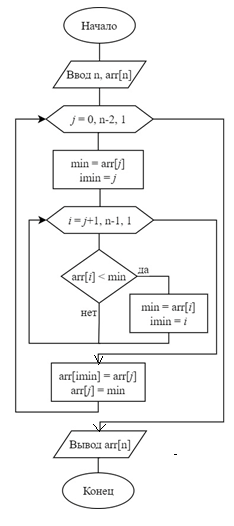
 Сортировка выбором относится к основным видам сортировок с вычислительной сложностью (n – размер массива).Сортировка выбором хороша тем, что ее принцип достаточно прост, а скорость почти не зависит от того, насколько упорядочен массив. Схема алгоритма решения задачи показана на рисунке 17.

Рисунок 17 – Схема алгоритма программы

**Текст программы:**

#include <stdio.h>

int main()

{

char arr[100], min;

int n, imin;

**puts("Input n<=100");**

scanf("%d", &n);

getchar();

printf("Input %d symbols:\n", n);

for(int i=0; i<n; i++) scanf("%c", &arr[i]);

printf("\n");

**puts("Inputed array:");**

for(int i=0; i<n; i++) printf("%c", arr[i]);

printf("\n");

for(int j=0; j<n-1; j++) {

min = arr[j];

imin = j;

for(int i=j+1; i<n; i++)

if (arr[i]<min) {

min = arr[i];

imin = i;

}

arr[imin] = arr[j];

arr[j] = min;

}

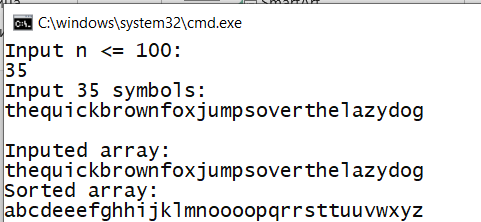
**puts("Sorted array:");**

for(int i=0; i<n; i++) printf("%c", arr[i]);

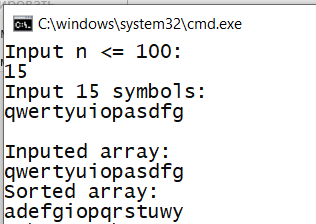
return 0;

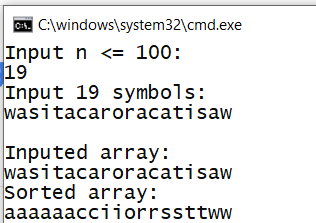
}

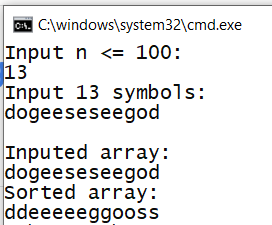
**Результаты работы программы:**



**Тесты**:







**Часть 2**. **Обработка матриц.**

**Задание.** Дана целочисленная матрица , (, ). Выяснить, есть ли в ней ненулевые элементы, и, если есть, определить разность индексов наименьшего и наибольшего из них. Вывести на экран исходную матрицу, найденные элементы и всю информацию о них, а также найденную разность. Если искомых элементов нет, выдать сообщение.

Как показал анализ задания, для выполнения поставленной задачи необходимо:

- ввести матрицу;

- вывести матрицу;

- определить максимальный ненулевой элемент и его индекс;

- определить минимальный ненулевой элемент и его индекс;

- вывести найденные значения.

Для реализации указанных подзадач целесообразно использовать поэлементную обработку матрицы при ее формировании, выводе и поиске заданных элементов.

Для проверки наличия в матрице ненулевых элементов можно использовать, как минимум, три способа:

1. Использовать флаги – логические переменные, показывающие, было ли достигнуто какое-то условие или нет. Например, начальное значение флага равно нулю, а при выполнении этого условия становится равно единице.
2. Задать переменным искомых значений специальное начальное значение.
   1. Задать начальное значение индексов невозможным, то есть, так как в С\C++ нумерация индексов всегда начинается с 0, индекс по умолчанию не может быть отрицательным. Таким образом, если в начале программы присвоить переменной искомого индекса значение «-1», то после обработки матрицы, если подходящих элементов нет, в нее все еще будет записано «-1», в ином случае – искомый индекс минимального или максимального элемента.
   2. Задать переменным максимума и минимума системные значения INT\_MINи INT\_MAX, соответственно, для типа данных int. Однако, в данном случае этот метод может работать некорректно, так как в условии не заданы никакие ограничения для значений элементов матрицы, следовательно, они могут совпасть с INT\_MINи INT\_MAX, из-за чего может возникнуть путаница в переменных индексов.

В данном примере реализуем первый способ. Схема алгоритма приведена на рисунке 18.

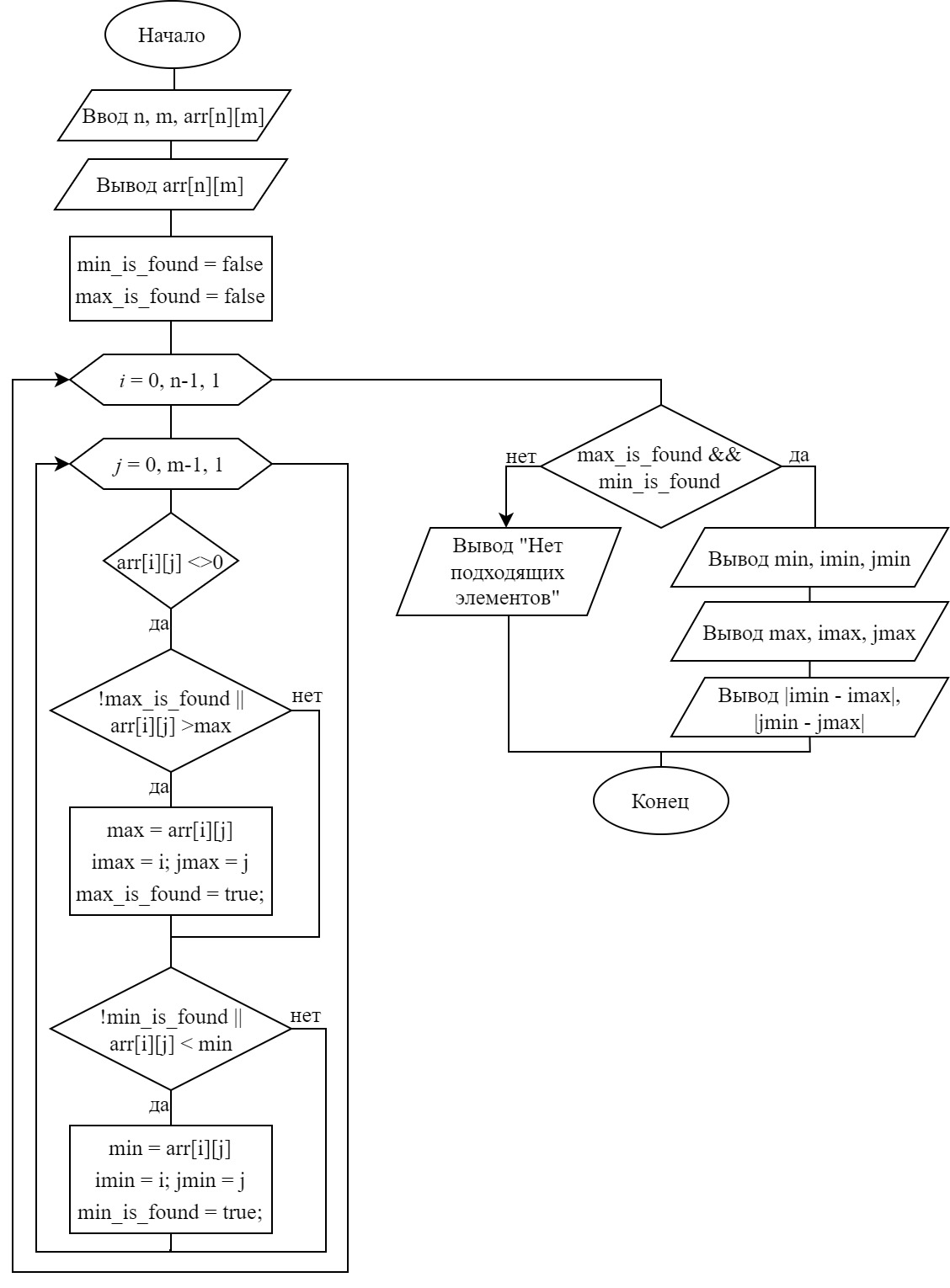


Рисунок 18 – Схема алгоритма программы

**Текст программы**:

#include <stdio.h>

#include <math.h>

int main(int argc, char\* argv[])

{

int arr[8][15], n, m;

int min, imin, jmin;

int max, imax, jmax;

bool min\_is\_found, max\_is\_found;

do{ //контроль выхода за границы размеров

puts("Inputn<=8, m<=15");

scanf("%d %d", &n, &m);

} while ((n>8) || (m>15));

printf("Input matrix %dx%d:\n", n, m);

for(int i=0; i<n; i++) {

for(int j=0; j<m; j++) scanf("%d", &arr[i][j]);

}

printf("\n");

puts("Inputedmatrix:"); // выводисходнойматрицы

for(int i=0; i<n; i++) {

for(int j=0; j<m; j++) printf("%4d", arr[i][j]);

printf("\n");

}

min\_is\_found = false; // обнуляемфлаги

max\_is\_found = false;

for(int i=0; i<n; i++)

for(int j=0; j<m; j++)

if (arr[i][j] != 0) { // ищемненулевыеэлементы

//записываем в maxи minпервый ненулевой элемент

//дальше – начинаем сравнивать с остальными

if(!max\_is\_found || (arr[i][j] >max)) {

max = arr[i][j];

imax = i;

jmax = j;

max\_is\_found = true;

}

if (!min\_is\_found || (arr[i][j] < min)) {

min = arr[i][j];

imin = i;

jmin = j;

min\_is\_found = true;

}

}

if (max\_is\_found&&min\_is\_found) {

//еслибылхотябыодинненулевойэлемент

printf("Minimum = %d, (i = %d, j = %d)\n", min, imin, jmin);

printf("Maximum = %d, (i = %d, j = %d)\n", max, imax, jmax);

printf("|imin - imax| = %d\n", abs(imin-imax));

printf("|jmin - jmax| = %d\n", abs(jmin-jmax));

}

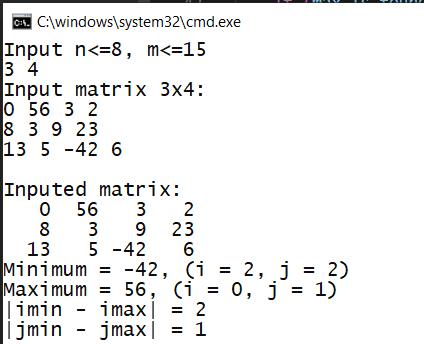
else

printf("There are no appropriate elements in the matrix\n");

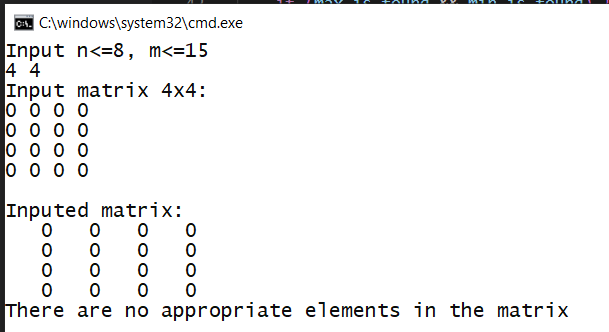
return 0;

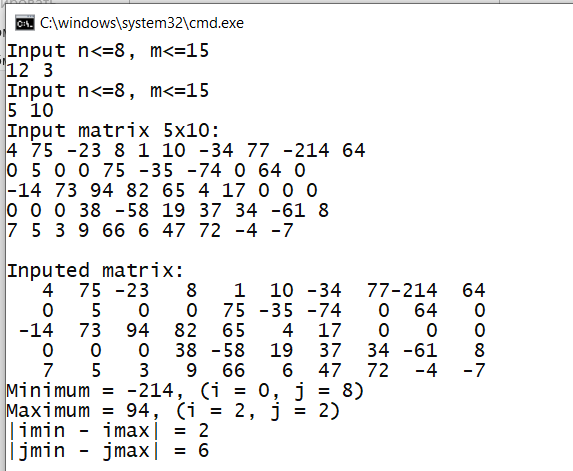
}

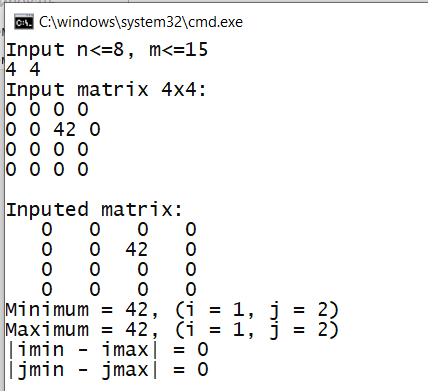
**Результаты работы программы**:



**Тесты**:







**Часть 3. Применение подпрограмм при обработке массивов.**

**Задание**. При решении задания необходимо использовать не менее -х подпрограмм, а также построить и представить в отчете структурную схему программы.

Пользователь вводит массив целых чисел до тех пор, пока не введет нулевых элемента – все нули являются элементами последовательности (изначально число элементов последовательности неизвестно, но гарантируется, что их будет не больше ). Необходимо вывести на экран элемент, индекс которого равен количеству отрицательных элементов введенной последовательности. После этого необходимо найти и вывести элементы последовательности, которые больше найденного по индексу. Если таких элементов нет, то вывести соответствующее сообщение.

Исходя из анализа задания, для решения задачи необходимо выполнить структурную декомпозицию и разбить программу на подпрограммы:

* Ввод массива.
* Вывод массива.
* Подсчет количества отрицательных элементов.
* Поиск и печать элементов, больших заданного значения.

Во всех задачах можно использовать поэлементную обработку массива.

Подпрограммы вывода массива и поиска определенных элементов организуем как процедуры, подпрограммы ввода массива и подсчета кол-ва отрицательных элементов – как функции, так как они должны возвращать скалярные значения. Структурная схема показана на рисунке 19.



Рисунок 19 – Структурная схема программы

Схемы алгоритмов подпрограмм приведены на рисунке 20, основной программы – на рисунке 21.

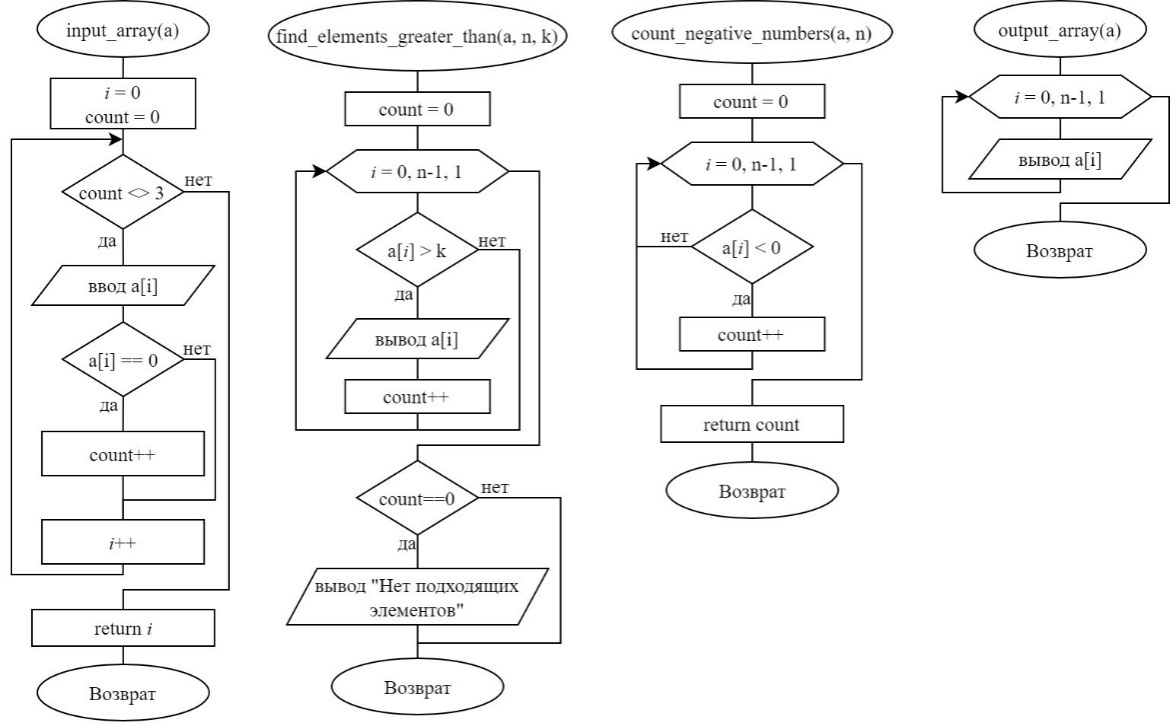


Рисунок 20 – Схемы алгоритмов подпрограмм

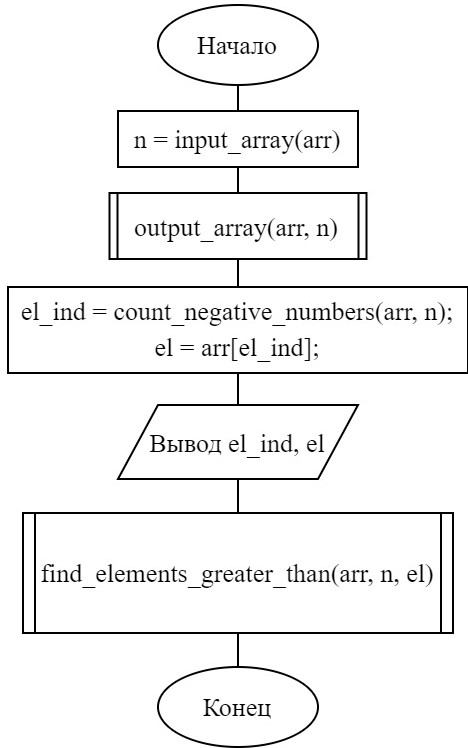


Рисунок 21 – Схема алгоритма основной программы

Далее, на языке С\С++, в любой визуальной среде реализуем разработанные алгоритмы.

**Текст программы:**

#include <stdio.h>

//пп ввода массива, возвращается кол-во элементов

int input\_array(int a[]) {

int i = 0;

puts("Input less then 15 numbers:");

for (int zero\_count = 0; zero\_count != 3; ++i) {

scanf("%d", &a[i]);

if (a[i] == 0) ++zero\_count;

else zero\_count = 0;

}

return i;

}

//пп выводамассива

void output\_array(int a[], int n) {

for (int i = 0; i<n; ++i) printf("%d ", a[i]);

printf("\n");

}

//пп поиска и печати элементов, больших k

void find\_elements\_greater\_than(int a[], int n, int k) {

int count = 0;

printf("Array elements greater then %d:\n", k);

for (int i = 0; i< n; ++i)

if (a[i] > k) {

printf("%d ", a[i]);

++count;

}

if (count == 0) puts("There are no such elements in the array");

}

//пп подсчета кол-ва отрицательных элементов

int count\_negative\_numbers(int a[], int n) {

int count = 0;

for (int i = 0; i< n; ++i)

if (a[i] < 0) ++count;

return count;

}

int main() {

int arr[15], n, el, el\_ind;

n = input\_array(arr);

puts("Inputed array:");

output\_array(arr, n);

el\_ind = count\_negative\_numbers(arr, n);

el = arr[el\_ind];

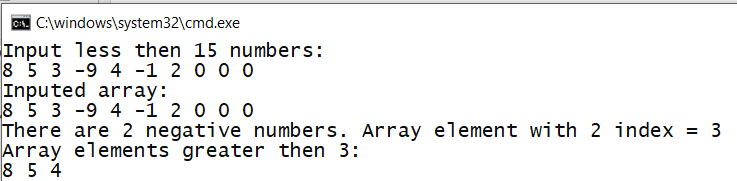
printf("There are %d negative numbers. Array element with %d index = %d\n", el\_ind, el\_ind, el);

find\_elements\_greater\_than(arr, n, el);

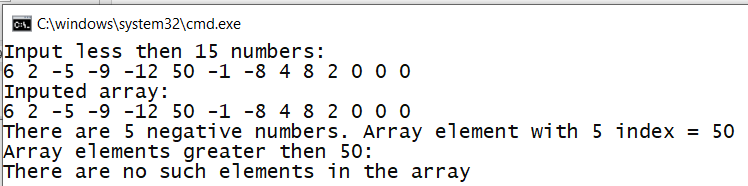
return 0;

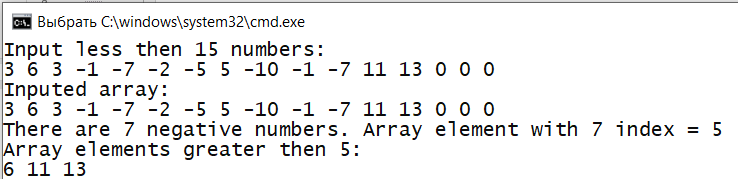
}

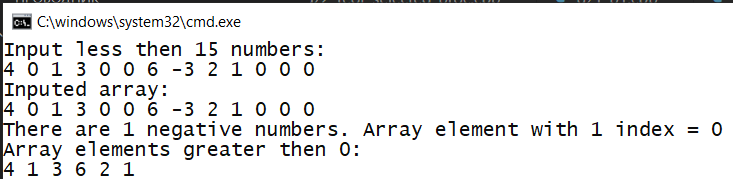
**Результаты работы программы:**

****

**Тесты:**

****

****

****

# Литература

Г.С.Иванова, Т.Н. Ничушкина, Р.С. Самарев. Средства процедурного программирования MicrosoftVisual С++ 2008. Учебное пособие. – М.: Издательство МГТУ им. Н.Э. Баумана, 2011.

Г.С. Иванова. Программирование. Учебник для ВУЗов. – М.: Кнорус, 2013.- 432 с.

Г.С. Иванова. Технология программирования. Учебник для ВУЗов. – М.: Кнорус, 2013. – 336 с.

Т.Н. Ничушкина, В.В. Гуренко. Разработка алгоритмов простейших программ. Электронное учебное издание. -М.: МГТУ им. Н.Э. Баумана, 2014.

Подбельский В.В. Язык С++: Учеб. пособие. – М.: Финансы и статистика, 2006.

# Приложение А. Пример титульного листа ДЗ

|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **ФУНДАМЕНТАЛЬНЫЕ НАУКИ**

КАФЕДРА **ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И МАТЕМАТИЧЕСКАЯ ФИЗИКА (фн11)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **02.03.01 МАТЕМАТИКА И КОМПЬЮТЕРНЫЕ НАУКИ**

**Отчет**

|  |  |
| --- | --- |
| **по домашнему заданию №** | 2 |

**Название:**

**Разработка программ с использованием сложных структурных типов данных**

***Часть 2.1.* Обработка одномерных массивов.**

***Часть 2.2.*Обработкаматриц.**

***Часть 2.3.* Применение подпрограмм при обработке массивов.**

**Дисциплина: Информатика**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ФН11-13Б |  |  | В.П.Малышев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Т.Н.Ничушкина |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2024

# Приложение Б. Примеры вариантов задания

**Домашние задания. Вариант 27**

**Задание 2**

***Часть 1. Обработка одномерных массивов.***

Упорядочить массив А(30) по неубыванию, используя метод сортировки обменом (метод пузырька): последовательно сравниваются пары соседних элементов и, если первый больше второго, то они меняются местами и т.д., пока наибольший элемент не окажется последним. Процедура повторяется пока массив не окажется отсортированным.

***Часть 2. Обработка матриц.***

Дана целочисленная матрица **A(n,m),(n>=8, m<=15).** Выяснить, есть ли в ней ненулевые элементы, и если есть, определить разность индексов наименьшего и наибольшего из них. Вывести на экран исходную матрицу, найденные элементы и всю информацию о них, а также найденную разность. Если искомых элементов нет, выдать сообщение.

***Часть 3. Применение подпрограмм при обработке массивов.***

Сформировать одномерный массив целых чисел. В полученном массиве определить сумму нечетных элементов массива, среднее арифметическое его элементов, кратных двум и произведение его элементов, кратных 7. Вывести на экран исходный массив и всю полученную информацию.

**Домашние задания. Вариант 28**

**Задание 2**

***Часть 1. Обработка одномерных массивов.***

Упорядочить массив А(n) (n<=50) по невозрастанию абсолютной величины элементов, используя метод сортировки «пузырьком».

***Часть 2. Обработка матриц.***

Дана целочисленная матрица **A(n,m),(n<=8,m<=15).** Выяснить, есть ли в ней нулевые элементы, и если есть, определить разность индексов первого и последнего из них. Вывести на экран исходную матрицу и индексы искомых элементов. Если нулевых элементов нет – вывести соответствующее сообщение.

***Часть 3. Применение подпрограмм при обработке массивов.***

Сформировать одномерный массив целых чисел. В полученном массиве определить сумму четных элементов массива, среднее арифметическое его элементов, кратных трем и произведение его отрицательных элементов. Вывести на экран исходный массив и всю полученную информацию.