Лабораторная работа № 7

**Сортировка записей**

**Цель:** изучить различные виды сортировки и реализовать их для структуры данных.

## Общее описание

При работе с массивами данных не редко возникает задача их сортировки по возрастанию или убыванию, т.е. упорядочивания. Сортировка представляет собой процесс упорядочения элементов в массиве в порядке возрастания или убывания их значений.

Элементы массива можно сортировать:

* по возрастанию – каждый следующий элемент больше предыдущего – А[1] < А[2] <... < A[N];
* по не убыванию – каждый следующий элемент не меньше предыдущего, то есть больше или равен А[1 ] ≤ А[2] ≤... ≤A[N];
* по убыванию – каждый следующий элемент меньше предыдущего А[1 ] > А[2] > ... > A[N];
* по не возрастанию – каждый следующий элемент не больше предыдущего, то есть меньше или равен А[1] ≥ А[2] ≥ ... ≥ A[N].

Алгоритмы сортировки отличаются друг от друга степенью эффективности, под которой понимается количество сравнений и количество обменов, произведенных в процессе сортировки.

Существует большое количество алгоритмов сортировки, но все они базируются на трех основных:

* сортировка обменом;
* сортировка выбором;
* сортировка вставкой.

## Сортировка обменом или «пузырьковая» сортировка

Сортировка пузырьком – простейший алгоритм сортировки, применяемый чисто для учебных целей. Практического применения этому алгоритму нет, так как он не эффективен, особенно если необходимо отсортировать массив большого размера. К плюсам сортировки пузырьком относится простота реализации алгоритма.

Алгоритм сортировки пузырьком сводится к повторению проходов по элементам сортируемого массива. Проход по элементам массива выполняет внутренний цикл. За каждый проход сравниваются два соседних элемента, и если порядок неверный элементы меняются местами. Внешний цикл будет работать до тех пор, пока массив не будет отсортирован. Таким образом внешний цикл контролирует количество срабатываний внутреннего цикла. Когда при очередном проходе по элементам массива не будет совершено ни одной перестановки, то массив будет считаться отсортированным.

Возьмём массив [5, 1, 4, 2, 8] и отсортируем значения по возрастанию, используя сортировку пузырьком. Выделены те элементы, которые сравниваются на этапах выполнения.

**Первый проход:**

|  |  |  |
| --- | --- | --- |
| **До** | **После** | **Описание шага** |
| **5 1** 4 2 8 | **1 5** 4 2 8 | Здесь алгоритм сравнивает два первых элемента и меняет их местами. |
| 1 **5 4** 2 8 | 1 **4 5** 2 8 | Меняет местами, так как 5 > 4 |
| 1 4 **5 2** 8 | 1 4 **2 5** 8 | Меняет местами, так как 5 > 2 |
| 1 4 2 **5 8** | 1 4 2 **5 8** | Теперь, ввиду того, что элементы стоят на своих местах (8 > 5), алгоритм не меняет их местами. |

**Второй проход:**

|  |  |  |  |
| --- | --- | --- | --- |
| **До** | **После** |  | **Описание шага** |
| **1 4** 2 5 8 | **1 4** 2 5 8 |  |  |
| 1 **4 2** 5 8 | 1 **2 4** 5 8 |  | Меняет местами, так как 4 > 2 |
| 1 2 **4 5** 8 | 1 2 **4 5** 8 |  |  |
| 1 2 4 **5 8** | 1 2 4 **5 8** |  |  |

/\* Пузырьковая сортировка. \*/

*void bubble(char \*items, int count)*

*{*

*register int a, b;*

*register char t;*

*for(a=1; a < count; ++a)*

*for(b=count-1; b >= a; --b) {*

*if(items[b-1] > items[b]) {*

*/\* меняем местами элементы \*/*

*t = items[b-1];*

*items[b-1] = items[b];*

*items[b] = t;*

*}*

*}*

*}*

Здесь ***items*** – имя массива символов (указатель на массив), подлежащий сортировке, a ***count*** – количество элементов в массиве. Работа пузырьковой сортировки выполняется в двух циклах. Если количество элементов массива равно ***count***, внешний цикл приводит к просмотру массива ***count - 1*** раз. Это обеспечивает размещение элементов в правильном порядке к концу выполнения функции даже в самом худшем случае. Все сравнения и обмены выполняются во внутреннем цикле.

Приведем пример программы сортировки обменом. Заполняем символьный массив *items* размерности 10 с клавиатуры и выводим его на экран. Затем сортируем введенный массив, выводя промежуточные этапы сортировки на экран. Завершается программа выводом отсортированного массива.

*#include "stdio.h"*

*#include "locale"*

*void main()*

*{*

*setlocale (LC\_ALL, "Russian");*

*register int i, j;*

*register char t;*

*char items[10];*

*int count=10;*

*static int l=0;*

***printf ("\nВвод массива items\n");***

*for(i=0; i < count; ++i)*

*{*

*printf ("Введите элемент массива items[%d] = ", i);*

*scanf ("%c", &items[i]);*

*getchar();*

*}*

***printf ("\n Полученный массив items\n\n");***

*for(i=0; i < count; ++i)*

*{*

*printf ("%c\t", items[i]);*

*}*

***printf ("\n\n Процесс сортировки массива items\n");***

*for(i=0; i < count; ++i)*

***{***

*l++;*

*for(j=count-1; j >= i; --j)*

*{*

*if(items[j-1] > items[j])*

***{***

*printf ("\n****Меняем местами элементы****\n ");*

*printf (" items[%d] = %c items[%d] = %c \n", j-1,*

*items[j-1], j, items[j]);*

*/\* меняем местами элементы \*/*

*t = items[j-1];*

*items[j-1] = items[j];*

*items[j] = t;*

*printf ("Результат перестановки\n");*

*printf (" items[%d] = %c items[%d] = %c \n", j-1, items[j-1],*

*j, items[j]);*

***}***

*}*

*printf ("\nРезультат перестановки в %d-м проходе\n", l);*

*for(int ii=0; ii < count; ++ii)*

*{*

*printf ("%c\t", items[ii]);*

*}*

***}***

***printf ("\n\n Отсортированный массив items\n\n");***

*for(i=0; i < count; ++i)*

*{*

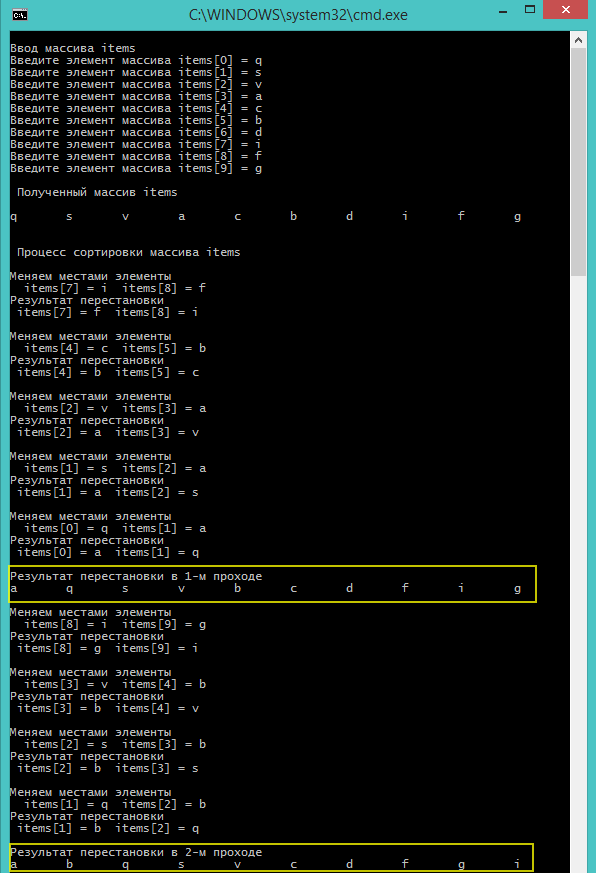
*printf ("%c\t", items[i]);*

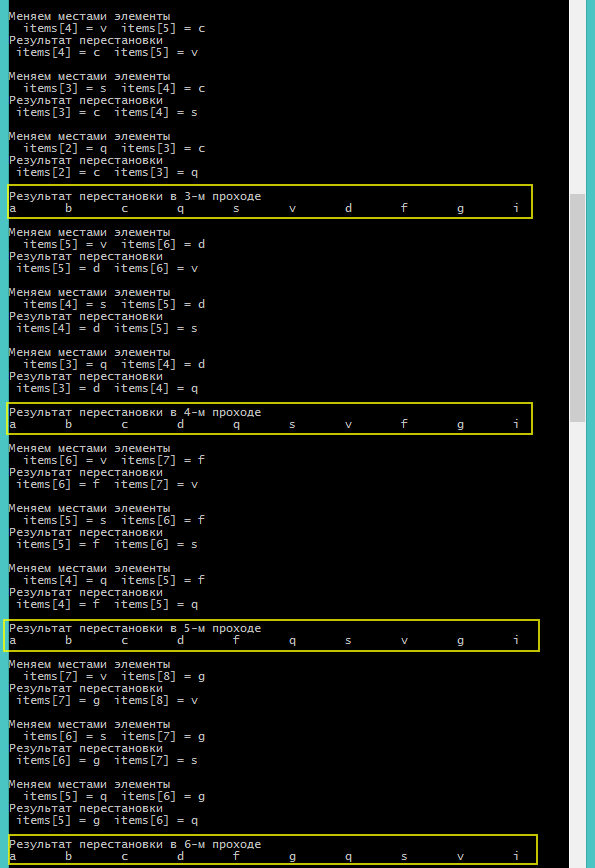
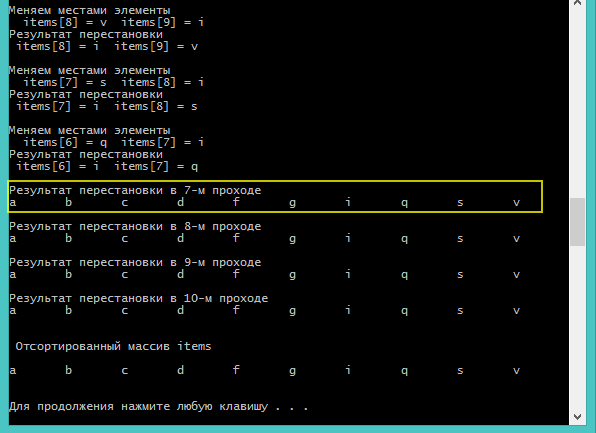
*}*

*printf ("\n\n");*

*}*

Результат работы программы представлен на рисунке 1.



**Рисунок 1 – Результат работы программы**

## Сортировка выбором

Самый простой алгоритм сортировок – это сортировка выбором. Алгоритм сортировки выбором находит в исходном массиве максимальный или минимальный элементы, в зависимости от того как необходимо сортировать массив, по возрастанию или по убыванию. Если массив должен быть отсортирован по возрастанию, то из исходного массива необходимо выбирать минимальные элементы. Если же массив необходимо отсортировать по убыванию, то выбирать следует максимальные элементы.

Допустим необходимо отсортировать массив по возрастанию. В исходном массиве находим минимальный элемент, меняем его местами с первым элементом массива. Уже из всех элементов массива один элемент стоит на своём месте. Теперь будем рассматривать не отсортированную часть массива, то есть все элементы массива, кроме первого. В неотсортированной части массива опять ищем минимальный элемент. Найденный минимальный элемент меняем местами со вторым элементом массива и т. д. Таким образом, суть алгоритма сортировки выбором сводится к многократному поиску минимального (максимального) элементов в неотсортированной части массива. Отсортируем массив из семи чисел согласно алгоритму «Сортировка выбором».

**Исходный массив: 3 3 7 1 2 5 0**

1. Итак, находим минимальный элемент в массиве. 0 – минимальный элемент;
2. Меняем местами минимальный и первый элементы массива. Текущий массив: 0 3 7 1 2 5 3;
3. Находим минимальный элемент в неотсортированной части массива. 1 – минимальный элемент;
4. Меняем местами минимальный и первый элементы массива. Текущий массив: 0 1 7 3 2 5 3;
5. min = 2;
6. Текущий массив: 0 1 2 3 7 5 3;
7. min = 3;
8. Текущий массив: 0 1 2 3 7 5 3 в массиве ничего не поменялось, так как 3 стоит на своём месте;
9. min = 3;
10. Конечный вид массива: 0 1 2 3 3 5 7 – массив отсортирован.

*void choicesSort(int\* arrayPtr, int length\_array) // сортировка выбором*

*{*

*for (int repeat\_counter = 0; repeat\_counter < length\_array; repeat\_counter++)*

*{*

*int temp = arrayPtr[0]; // временная переменная для хранения*

*// значения перестановки*

*for (int element\_counter = repeat\_counter + 1; element\_counter < length\_array; element\_counter++)*

*{*

*if (arrayPtr[repeat\_counter] > arrayPtr[element\_counter])*

*{*

*temp = arrayPtr[repeat\_counter];*

*arrayPtr[repeat\_counter] = arrayPtr[element\_counter];*

*arrayPtr[element\_counter] = temp;*

*}*

*}*

*}*

*}*

Приведем пример программы сортировки выбором. Заполняем целочисленный массив *arrayPt* размерности 10 (*length\_array*) с клавиатуры и выводим его на экран. Затем сортируем введенный массив, выводя промежуточные этапы сортировки на экран. Завершается программа выводом отсортированного массива.

*#include "stdio.h"*

*#include "locale"*

*void main() // сортировка выбором*

*{*

*setlocale (LC\_ALL, "Russian");*

*register int i, j;*

*int arrayPtr[10]; // Задаем целочисленный массив*

*int length\_array=10; // Размер массива*

*int minValueIndex; // индекс минимального элемента*

*static int l=0;*

*int temp; // Временная переменная для хранения значения перестановки*

***printf ("\nВвод массива arrayPtr\n");***

*for(i=0; i < length\_array; ++i)*

*{*

*printf ("Введите элемент массива arrayPtr[%d] = ", i);*

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

*}*

*printf ("\n Полученный массив arrayPtr\n\n");*

*for(i=0; i < length\_array; ++i)*

*{*

*printf ("%d\t", arrayPtr[i]);*

*}*

***printf ("\n\n Процесс сортировки массива arrayPtr\n");***

*for(i = 0; i < length\_array; i++)*

*{*

*l++; // Счетчик циклов сортировки*

*minValueIndex = i; // Найдем минимальный элемент на*

*// промежутке индексов [i+1; length\_array].*

*// Изначально его индекс равен i*

*// Переберем оставшиеся элементы промежутка*

*for (j = i+1; j < length\_array; j++)*

*{*

*// Если элемент в позиции j меньше*

*// элемента в позиции minValueIndex,*

*// то необходимо обновить значение индекса*

*if (arrayPtr[j] < arrayPtr[minValueIndex])*

*{*

*minValueIndex = j;*

*}*

*}*

*printf ("\nМинимальный элемент arrayPtr[%d] = %d \n",*

*minValueIndex, arrayPtr[minValueIndex]);*

*// Меняем текущий элемент с минимальным*

*printf ("\nМеняем местами элементы\n");*

*printf (" arrayPtr[%d] = %d arrayPtr[%d] = %d \n", i,*

*arrayPtr[i],minValueIndex, arrayPtr[minValueIndex]);*

*temp = arrayPtr[i];*

*arrayPtr[i] = arrayPtr[minValueIndex];*

*arrayPtr[minValueIndex] = temp;*

*printf (" arrayPtr[%d] = %d arrayPtr[%d] = %d \n", i,*

*arrayPtr[i],minValueIndex, arrayPtr[minValueIndex]);*

*printf ("\nРезультат перестановки в %d-м проходе\n", l);*

*for(int ii=0; ii < length\_array; ++ii)*

*{*

*printf ("%d\t", arrayPtr[ii]);*

*}*

*}*

***printf ("\n\n Отсортированный массив arrayPtr\n\n");***

*for(i=0; i < length\_array; ++i)*

*{*

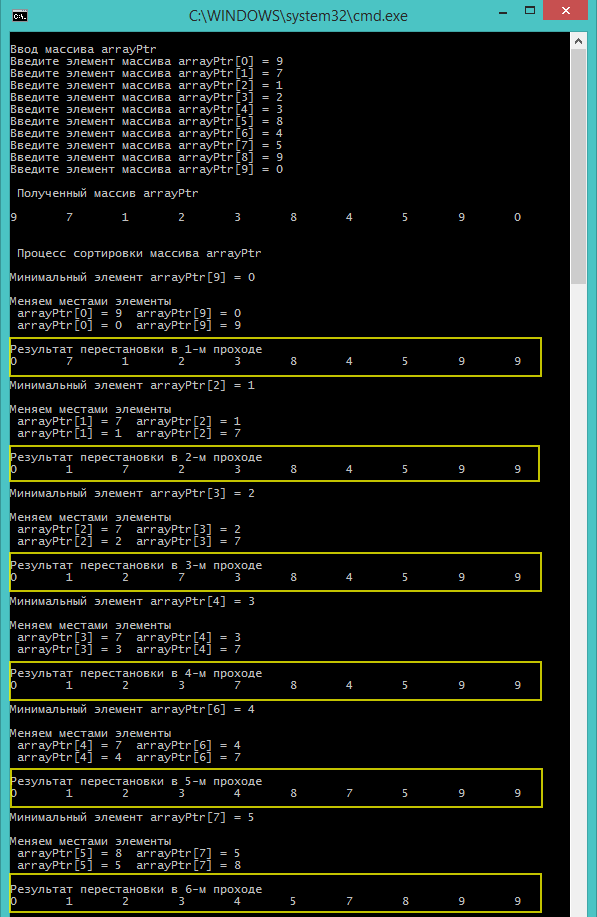
*printf ("%d\t", arrayPtr[i]);*

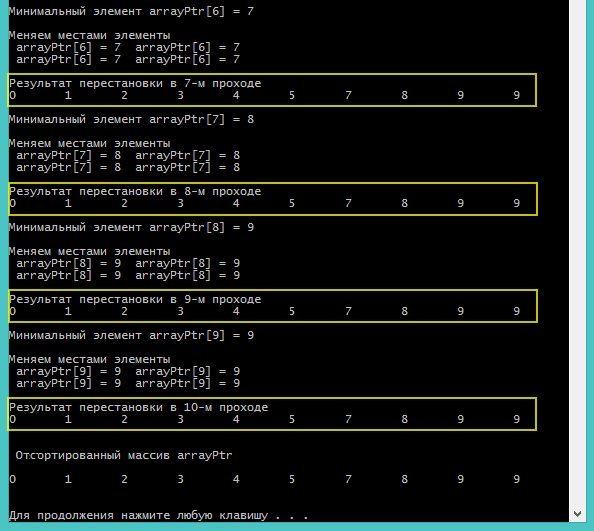
*}*

*printf ("\n\n");*

*}*

Результат работы программы представлен на рисунке 2.





**Рисунок 2 – Результат работы программы**

## Сортировка вставкой

При сортировке вставками массив разбивается на две области: упорядоченную и неупорядоченную. Изначально весь массив является неупорядоченной областью. При первом проходе первый элемент из неупорядоченной области изымается и помещается в правильном положении в упорядоченной области.

На каждом проходе размер упорядоченной области возрастает на 1, а размер неупорядоченной области сокращается на 1.

Основной цикл работает в интервале от 1 до N-1. На j-й итерации элемент [i] вставлен в правильное положение в упорядоченной области. Это сделано путем сдвига всех элементов упорядоченной области, которые больше, чем [i], на одну позицию вправо. [i] вставляется в интервал между теми элементами, которые меньше [i], и теми, которые больше [i].

В отличие от пузырьковой сортировки и сортировки посредством выбора, количество сравнений в сортировке вставками зависит от изначальной упорядоченности списка. Если список уже отсортирован, количество сравнений равно *n*–1; в противном случае его производительность является величиной порядка *n*2.

*Сортировка вставками* имеет большую вычислительную сложность. Поэтому она эффективна на небольших наборах данных. Рекомендуется использовать этот метод на наборах размером до десятков элементов. Сортировка вставками эффективна на последовательностях с данными, которые уже частично отсортированы.

Вообще говоря, в худших случаях сортировка вставками настолько же плоха, как и пузырьковая сортировка и сортировка посредством выбора, а в среднем она лишь немного лучше. Тем не менее, у сортировки вставками есть два преимущества. Во-первых, ее поведение естественно. Другими словами, она работает меньше всего, когда массив уже упорядочен, и больше всего, когда массив отсортирован в обратном порядке. Поэтому сортировка вставками – идеальный алгоритм для почти упорядоченных списков. Второе преимущество заключается в том, что данный алгоритм не меняет порядок одинаковых ключей. Это значит, что если список отсортирован по двум ключам, то после сортировки вставками он останется упорядоченным по обоим.

*int i, j, temp;*

*for (i = 1; i < size; i++)*

*{*

*temp = array[i];*

*for (j = i - 1; j >= 0; j--)*

*{*

*if (array[j] < temp)*

*break;*

*array[j + 1] = array[j];*

*array[j] = temp;*

*}*

*}*

Приведем пример программы, которая считывает с клавиатуры последовательность элементов, сортирует их *методом вставок* и выводит отсортированный массив на экран.

В программе рассматривается алгоритм метода *сортировки вставками* на примере сортировки по возрастанию. Первый элемент в массиве образует уже отсортированную последовательность. Сравниваем второй элемент с первым. Если порядок между ними нарушен, то первый элемент передвигается на одну позицию вправо. Теперь отсортированный массив состоит из двух элементов.

Далее, в течении каждой итерации, берем следующий элемент (третий, четвертый и т.д.) и сравниваем его поочередно с другими элементами в уже отсортированном списке, *начиная с конца* этого списка. Если порядок между сравниваемыми элементами нарушен, то меняем их местами, если нет, то “вставка” нового элемента закончена, переходим к следующему.

*#include "stdio.h"*

*#include "locale"*

*void main() // сортировка выбором*

*{*

*setlocale (LC\_ALL, "Russian");*

*register int i, j;*

*int arrayPtr[10]; // определение массива - выделение памяти*

*// под массив*

*int length\_array=10; // размер массива*

*static int l=0;*

*int temp; // временная переменная для хранения*

*// значения перестановки*

*printf ("\nВвод массива arrayPtr\n");*

*for(i=0; i < length\_array; ++i)*

*{*

*printf ("Введите элемент массива arrayPtr[%d] = ", i);*

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

*}*

*printf ("\n Полученный массив arrayPtr\n\n");*

*for(i=0; i < length\_array; ++i)*

*{*

*printf ("%d\t", arrayPtr[i]);*

*}*

*printf ("\n\n Процесс сортировки массива arrayPtr\n");*

*for (i = 0; i < length\_array; i++)*

*{*

*l++; // определяет номер прохода сортировки*

*temp = arrayPtr[i];*

*for (j = i - 1; j >= 0; j--)*

*{*

*if (arrayPtr[j] < temp)*

*break;*

*printf ("\nМеняем элементы arrayPtr[%d] = %d на arrayPtr[%d] = %d*

*\n", j+1, arrayPtr[j+1],j, arrayPtr[j]);*

*arrayPtr[j + 1] = arrayPtr[j];*

*arrayPtr[j] = temp;*

*}*

*printf ("\nРезультат перестановки в %d-м проходе\n", l);*

*for(int ii=0; ii < length\_array; ++ii)*

*{*

*printf ("%d\t", arrayPtr[ii]);*

*}*

*}*

*printf ("\n\n Отсортированный массив arrayPtr\n\n");*

*for(i=0; i < length\_array; ++i)*

*{*

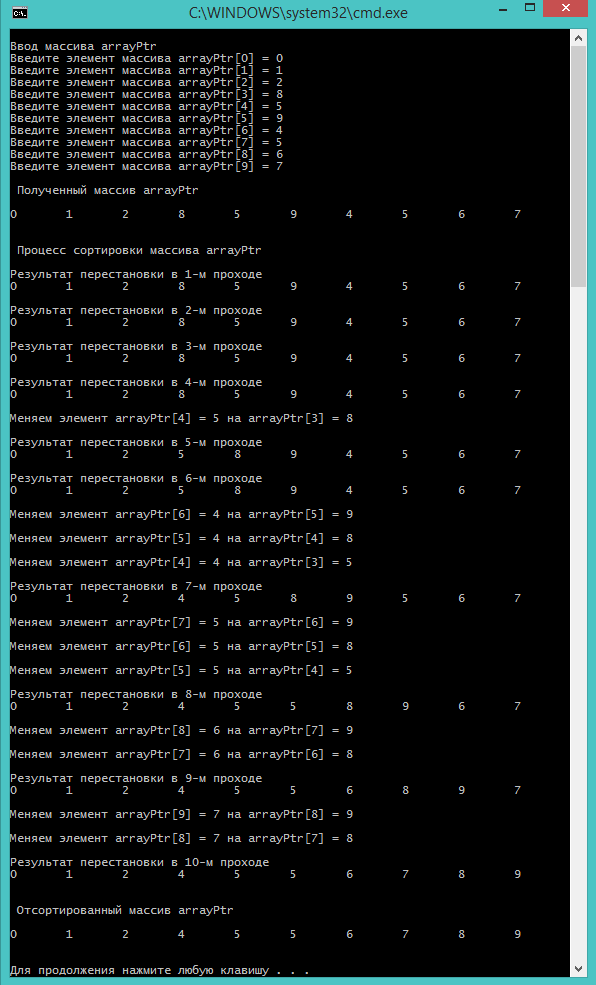
*printf ("%d\t", arrayPtr[i]);*

*}*

*printf ("\n\n");*

*}*

Результат работы программы представлен на рисунке 3.



**Рисунок 3 – Результат выполнения программы**

Пример сортировки массива размерности 3х3.

*#include <iostream>*

*#include "stdio.h"*

*#include "locale"*

*void main() // сортировка вставкой*

*{*

*setlocale (LC\_ALL, "Russian");*

*register int i, j, i1;*

*int arrayPtr[3][3]; // определение массива - выделение памяти*

*// под массив*

*int length\_array=3; // размер массива*

*static int l1=0;*

*int temp; // временная переменная для хранения*

*// значения перестановки*

*printf ("\nВвод массива arrayPtr\n");*

*for(i=0; i < length\_array; ++i)*

*{*

*for(j=0; j < length\_array; ++j)*

*{*

*printf ("Введите элемент массива arrayPtr[%d][%d] = ", i,j);*

*scanf ("%d", &arrayPtr[i][j]);*

*}*

*}*

*printf ("\n Полученный массив arrayPtr\n\n");*

*for(i=0; i < length\_array; ++i)*

*{*

*for(j=0; j < length\_array; ++j)*

*{*

*printf ("%d\t", arrayPtr[i][j]);*

*}*

*printf ("\n");*

*}*

*printf ("\n\n Процесс сортировки массива arrayPtr\n");*

*for (i = 1; i < 9; i++) // кол-во строк\*кол-во столбцов = 3\*3*

*{*

*printf ("\n Цикл i = %d \n", i);*

*temp = arrayPtr[0][i];*

*printf ("\n\ttemp = %d\n", temp);*

*for (i1 = i-1; i1 >=0; i1--)*

*{*

*printf ("\n\tВнутренняя сортировка i1 = %d \n", i1);*

*l1++;// определяет номер прохода сортировки*

*if ((arrayPtr[0][i1]) < temp)*

*{*

*printf ("\nПереход к следующему элементу\n");*

*break;*

*}*

*printf ("\nМеняем элементы \n");*

*printf ("\n arrayPtr[%d][%d] = %d на arrayPtr[[%d][%d] = %d \n", 0, i1+1,arrayPtr[0][i1+1],0,i1, arrayPtr[0][i1]);*

*arrayPtr[0][i1+1] = arrayPtr[0][i1];*

*arrayPtr[0][i1] = temp;*

*}*

*printf ("\nРезультат перестановки в %d-м проходе\n", l1);*

*for(int ii=0; ii < length\_array; ++ii)*

*{*

*for(int ii1=0; ii1 < length\_array; ++ii1)*

*{*

*printf ("%d\t", arrayPtr[ii][ii1]);*

*}*

*printf ("\n");*

*}*

*}*

*printf ("\n\n Отсортированный массив arrayPtr\n\n");*

*for(i=0; i < length\_array; ++i)*

*{*

*for(j=0; j < length\_array; ++j)*

*{*

*printf ("%d\t", arrayPtr[i][j]);*

*}*

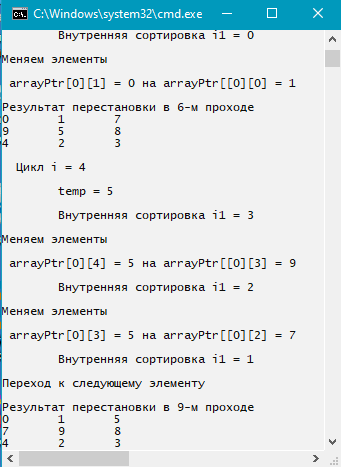
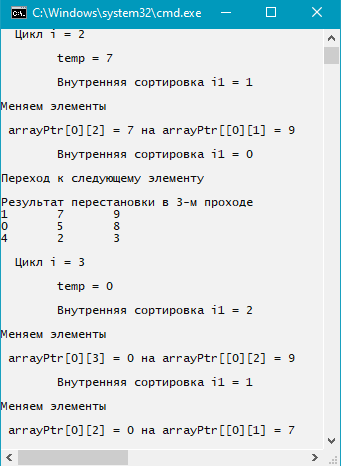
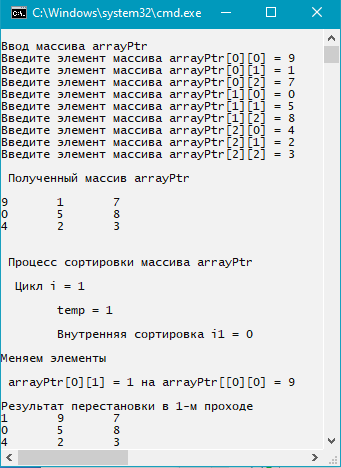
*printf ("\n");*

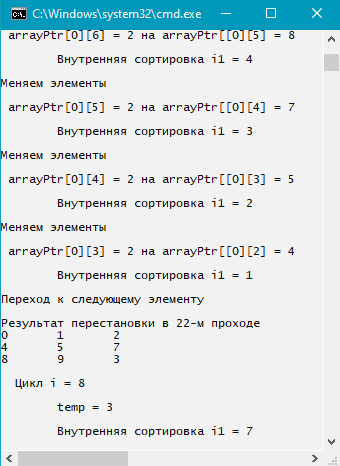
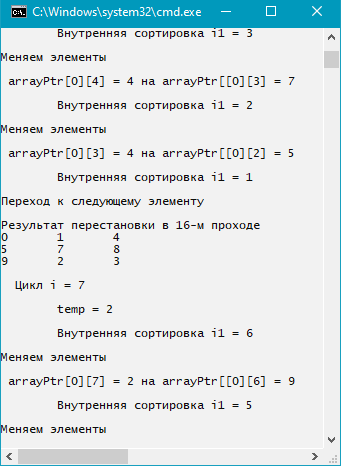
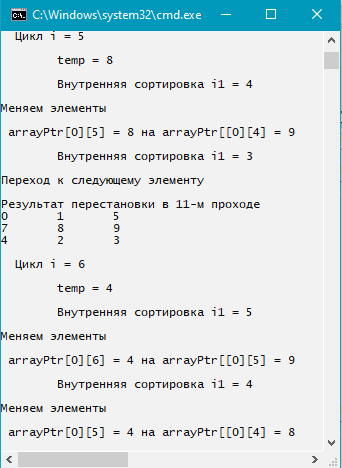
*}*

*printf ("\n\n");*

*}*

Результат работы программы представлен на рисунке 4.





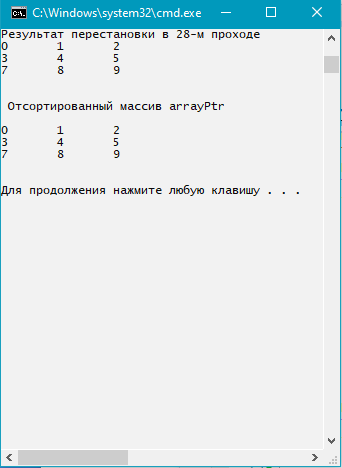
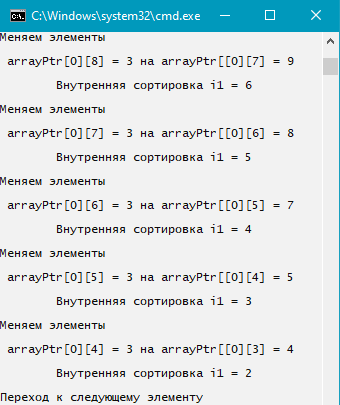


Рисунок 4 – результат выполнения программы

Изменим код программы, используя указатели для получения адреса элементов массива:

*for (i1 = i - 1; i1 >= 0; i1--)*

*{*

*printf("\n\tВнутренняя сортировка i1 = %d \n", i1);*

*l1++;// определяет номер прохода сортировки*

*if ((arrayPtr[0][i1]) < temp)*

*{*

*printf("\nПереход к следующему элементу\n");*

*break;*

*}*

*printf("\nМеняем элементы \n");*

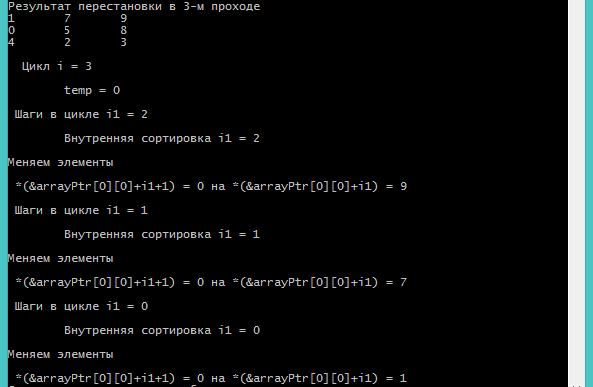
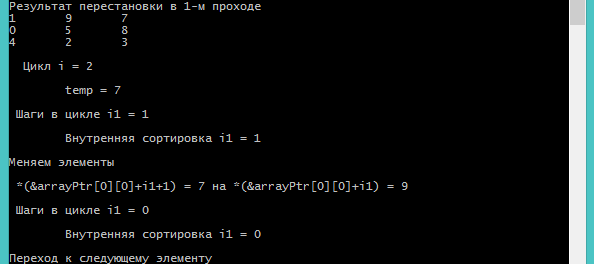
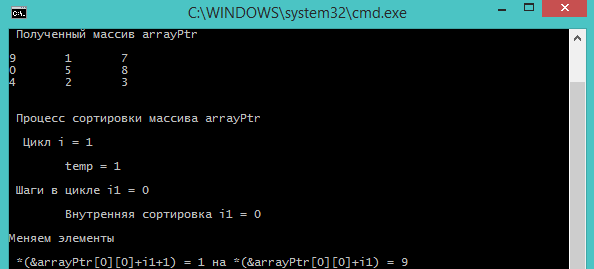
*printf("\n \*(&arrayPtr[0][0] + i1 + 1) = %d на \*(&arrayPtr[0][0] + i1) = %d \n", 0, i1 + 1, \*(&arrayPtr[0][0] + i1 + 1), 0, i1, \*(&arrayPtr[0][0] + i1));*

*\*(&arrayPtr[0][0] + i1 + 1) = \*(&arrayPtr[0][0] + i1);*

*\*(&arrayPtr[0][0] + i1) = temp;*

*}*

Результат работы программы представлен на рисунке 5.



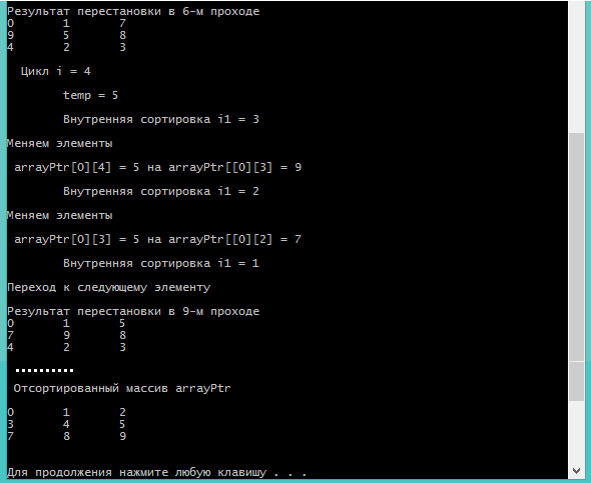


Рисунок 5 – результат выполнения программы

## Сортировка Шелла

Описание алгоритма “Сортировка Шелла”.

Этот метод сортировки Д. Шелл предложил в 1959 г. Он использует минимум памяти и показывает высокие скорости при сортировке. По сути в методе Шелла применяются сравнения и перестановки элементов аналогичные методу вставок, но при этом порядок сравниваемых элементов совершенно другой.

Идея сортировки методом Шелла заключается в сравнение разделенных на группы элементов последовательности, находящихся друг от друга на некотором расстоянии step. Изначально это расстояние равно ***step = N/2***, где N – общее число элементов. На первом шаге каждая группа включает в себя два элемента расположенных друг от друга на расстоянии N/2; они сравниваются между собой, и, в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние step сокращается на ***step/2***, и количество групп, соответственно, уменьшается. Постепенно расстояние между элементами уменьшается, и на ***step =1*** проход по массиву происходит в последний раз, выполняется обычная сортировка вставкой.

До сих пор продолжает обсуждаться вопрос выбора шага сортировки step. Шелл предложил такую последовательность: N/2, N/4, N/8 …, где N – количество элементов в сортируемом массиве.

Сортировка Шелла требует около log2N проходов для упорядочивания последовательности длиной N. Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Рассмотрим пример сортировки массива методом Шелла.

Дан массив: **A[14] = {32, 95, 16, 82, 24, 66, 35, 19, 75, 54, 40, 43, 93, 68**}.

В качестве значений шага выбираем **step = {5, 3,1}.**

**step = 5 (по 3 элемента)**

На первом шаге сортируются подмассивы ***Ai,j*** массива ***A***, составленные из всех элементов ***A***, различающихся на 5 позиций, то есть подсписки:

A5,1 = {**32, 66, 40**}; A5,2 = {**95, 35, 43**}; A5,3 = {**16, 19, 93**};

A5,4 = {**82, 75, 68**}; A5,5 = {**24, 54**}.

После сортировки:

A[14] = {**32**, **95**, **16**, **82**, **24**, **66**, **35**, **19**, **75**, **54**, **40**, **43**, **93**, **68**}получаем

A[14] = {**32**, **35**, **16**, **68**, **24**, **40**, **43**, **19**, **75**, **54**, **66**, **95**, **93**, **82**}.

В полученном списке на втором шаге вновь сортируются подсписки из отстоящих на 3 позиции элементов.

**step = 3 (по 5 элементов)**

На первом шаге сортируются подмассивы ***Ai,j*** массива ***A***, составленные из всех элементов ***A***, различающихся на 5 позиций, то есть подсписки:

A3,1 = {**32, 68, 43, 54, 93**}; A3,2 = {**35, 24, 19, 66, 82**}; A3,3 = {**16, 40, 75, 95**};.

После сортировки:

A[14] = {**32**, **35**, **16**, **68**, **24**, **40**, **43**, **19**, **75**, **54**, **66**, **95**, **93**, **82**}получаем

A[14] = {**32**, **19**, **16**, **43**, **24**, **40**, **54**, **35**, **75**, **68**, **66**, **95**, **93**, **82**}.

**step = 1 (все элементы)**

Процесс завершается обычной сортировкой вставками получившегося списка.

A[14] = {16, 19, 24, 32, 35, 40, 43, 54, 66, 68, 75, 82, 93, 95}.

Рассмотрим пример программной реализации алгоритма сортировки Шелла.

В программе массив заполняется случайными числами с помощью функции **rand** из стандартной библиотеки. Она генерирует целое число в диапазоне от 0 до **RAND\_MAX** (символическая константа, равная 32767).

Так как при каждом запуске программы будут генерироваться одни и те же случайные числа функцией rand, то для того чтобы числа были разными нужно использовать библиотечную функцию **srand**, которая задает начальное число для функции **rand**. Для получения начального числа считываются показания часов с помощью функции **time,** которая возвращает текущее время в секундах.

**ПРИМЕР**

**Создание структуры Студенты.**

#include<stdio.h>

#include<string>

#include<iostream>

#include <windows.h>

using namespace std;

//объявление структуры

typedef struct student {

int rno;

char name[20]; //объявления члена структуры

char surname[20];

char second\_name[20];

struct subject { //структура внутри струкуры

int scode;

char name[20];

int mark;

}sub[3]; //объявление массива в структуре

int total;

float per;

}student;

**Пример функции сортировки**

void sort\_on\_screen() {

student\* s, s1;

FILE\* fp;

fp = fopen("mystudents1.txt", "r");

fseek(fp, 0, SEEK\_END);

int n = ftell(fp) / sizeof(student);

int i, j;

rewind(fp);

s = (student\*)calloc(n, sizeof(student));

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

fread(&s[i], sizeof(student), 1, fp);

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

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

if (s[i].total < s[j].total) {

s1 = s[i];

s[i] = s[j];

s[j] = s1;

}

}

}

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

printf("\n%-5d%-20s%-20s%-20s", s[i].rno, s[i].name, s[i].surname, s[i].second\_name);

for (j = 0; j < 3; j++) {

printf("%4d", s[i].sub[j].mark);

}

printf("%5d%7.2f", s[i].total, s[i].per);

}

fclose(fp);

}

void sort\_in\_file() {

student\* s, s1;

FILE\* fp;

fp = fopen("mystudents1.txt", "r");

fseek(fp, 0, SEEK\_END);

int n = ftell(fp) / sizeof(student);

int i, j;

rewind(fp);

s = (student\*)calloc(n, sizeof(student));

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

fread(&s[i], sizeof(student), 1, fp);

fclose(fp);

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

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

if (s[i].total < s[j].total) {

s1 = s[i];

s[i] = s[j];

s[j] = s1;

}

}

}

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

printf("\n%-5d%-20s%-20s%-20s", s[i].rno, s[i].name, s[i].surname, s[i].second\_name);

for (j = 0; j < 3; j++) {

printf("%4d", s[i].sub[j].mark);

}

printf("%5d%7.2f", s[i].total, s[i].per);

}

fp = fopen("mystudents1.txt", "w");

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

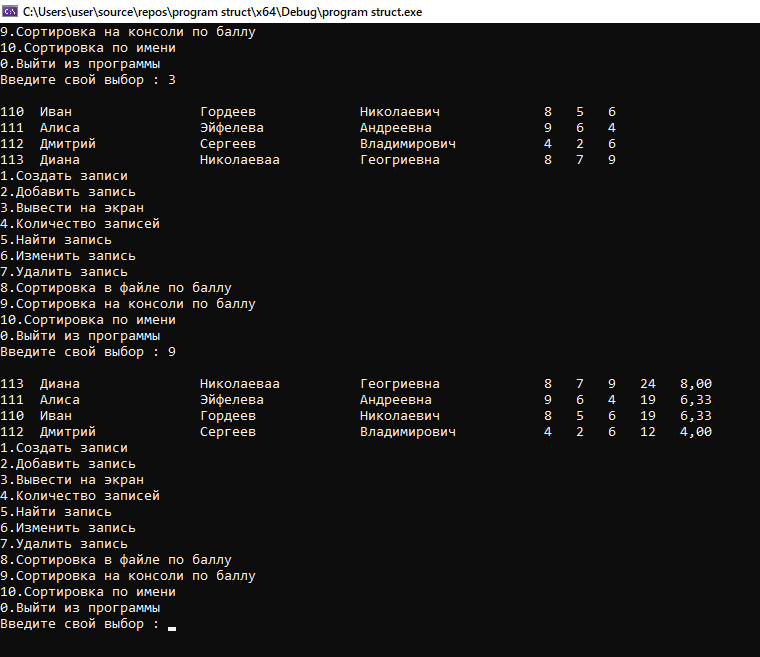
fwrite(&s[i], sizeof(student), 1, fp);

}

fclose(fp);

}

Результат работы программы:



**ЗАДАНИЕ**

Создайте структуру Меню столовой с полями Название блюда, Цена, Вес (г), Тип (Суп, Гарнир, Десерт, Горячее, Закуска и др.). Реализуйте сортировку 2 различными способами (по Цене и по Типу блюда). Добавьте функции редактирования записей и удаления записей. Результаты вывести на консоль.