Лабораторная работа №3. Эффективные алгоритмы обработки массивов

Часть I. Обработка одномерных массивов

1. Создайте новое решение **Лабораторная работа 3** и проект **Example1** в нём.
2. В главной функции определите одномерный массив:

```
const long size=10000; //количество элементов в массиве
const int value=100;   //диапазон значений
int array_1[size];     //одномерный массив
```

3. Перед описанием главной функции объявите прототип функции инициализации элементов массива случайными числами:

```
//заполнение массива случайными числами
void SetRandomValues(int *m, long n, int val);
```

4. После описания главной функции разместите реализацию функции инициализации массива:

```
//инициализация элементов массива случайными значениями
//m – массив, n – количество элементов в массиве, val – диапазон значений
void SetRandomValues(int *m, long n, int val)
{
   srand(time(0));           //инициализация ГСЧ
   for (int i=0; i<n; i++)
   {
         m[i]=rand()%val+1; //получение случайного числа в диапазоне от 1 до val
   }
}
```

5. По аналогии создайте функцию для вывода значений элементов массива на экран:

```
//вывод массива на экран
void PrintArray(const int *m, long n)
{
   for (int i=0; i<n; i++)
   {
         cout<<m[i]<<' ';
   }
   cout<<endl;
}
```

6. В главной функции программы выполните инициализацию массива и вывод его значений на экран.
7. Запустите программу и проверьте правильность ее работы.
8. Дополните программу функцией поиска максимального элемента:

```
//поиск максимального элемента в массиве
int MaxValue(const int *m, long n)
{
   int count=0;          //счетчик сравнений с положительным результатом
   int max=m[0];
   for (int i=1; i<n; i++)
   {
         if (m[i]>max)
         {
               max=m[i];
               count++;  //обновление счетчика
         }
   }
   cout<<"Кол-во сравнений: "<<count<<endl;    //вывод значения счетчика на экран
```

 © Екатерина Бурова, 2018.

```
    return max;
}
```

9. Дополните программу функцией сортировки массива по возрастанию методом простого обмена (Пузырьковый метод):

```
//сортировка массива по возрастанию методом "Пузырек"
void BubbleSortAsc(int *m, long n)
{
    for (int i=0; i<n; i++)
    {
        for (int j=0; j<n-1; j++)
        {
            if (m[j]>m[j+1])
            {
                int buf = m[j];
                m[j] = m[j+1];
                m[j+1] = buf;
            }
        }
    }
}
```

10. В главной функции выполните поиск максимального элемента, отсортируйте массив, а затем снова найдите максимальный элемент.
11. Запустите программу и проверьте результат ее работы.
12. Самостоятельно реализуйте сортировку по убыванию.
13. В главной функции выполните поиск максимального элемента в отсортированном по убыванию массиве.
14. Проведите серию испытаний программы и заполните таблицу 1 полученными данными:

Таблица 1. Поиск минимального элемента в массиве

№ теста	**Количество сравнений с положительным результатом при поиске максимального элемента**		
	После инициализации	**После сортировки по возрастанию**	**После сортировки по убыванию**
1			
2			
…			
5			
Среднее значение			

Проведите анализ полученных результатов и сделайте вывод.

Часть II. Обработка двумерных массивов

1. Создайте новое консольное приложение **Example2** в проекте **Лабораторная работа 3**.
2. В главной функции объявите следующие переменные:

```
const int rows = 100;        //количество строк
const int colls = 100;       //количество столбцов
short array_2[colls][rows]; //двумерный массив
unsigned int startTime = 0; //время начала
unsigned int endTime = 0;    //время завершения
unsigned int leadTime = 0;  //время выполнения
```

3. Опишите функцию заполнения массива случайными значениями:

```
//заполнение двумерного массива случайными значениями
//m – массив, colCount – количество столбцов, rowCount – количество строк
void SetRandomValues(short *m, int colCount, int rowCount)
{
   srand(time(0));
   for (int i=0; i<rowCount; i++)
         for (int j=0; j<colCount; j++)
               m[j+i*colCount]=rand()%100;
}
```

4. Далее реализуйте функцию, которая вычисляет сумму столбцов, выполняя обход построчно (смотри рисунок 1):

```
//построчное вычисление суммы элементов столбцов
//results – массив для записи сумм столбцов
void SumColls_1(const short *m, const int colCount, int rowCount, int *results)
{
   for (int i=0; i<rowCount; i++)
   {
         for (int j=0; j<colCount; j++)
         {
               results[j]+=m[j+i*colCount];
         }
   }
```

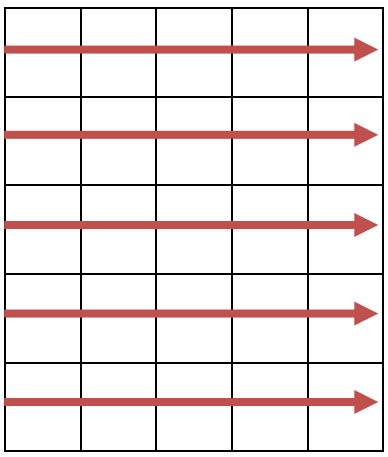

Рисунок 1. Обход массива по строкам

5. В главной функции выполните инициализацию массива.
6. Там же объявите одномерный массив для записи сумм столбцов:

```
int sums_1[colls]={0};
```

7. Вычислите суммы, используя созданную ранее функцию:

```
startTime=clock();                          //время начала выполнения функции
```

 © Екатерина Бурова, 2018.

```
SumColls_1((short *)array_2,colls,rows,sums_1);
endTime=clock();                              //время завершения функции
leadTime=(endTime-startTime);                 //итоговое время выполнения
cout<<"Время выполения функции (мс): "<<leadTime<<endl;
```

8. Реализуйте функцию нахождения сумм элементов в столбцах, которая будет выполнять обход массива по столбцам (смотри рисунок 2):

```
//вычисление суммы элементов столбцов с обходом по столбцам
//results – массив для записи сумм столбцов
void SumColls_2(const short *m, const int colCount, int rowCount, int results[])
{
   for (int j=0; j<colCount; j++)
   {
         for (int i=0; i<rowCount; i++)
         {
               results[j]+=m[j+i*colCount];
         }
   }
}
```

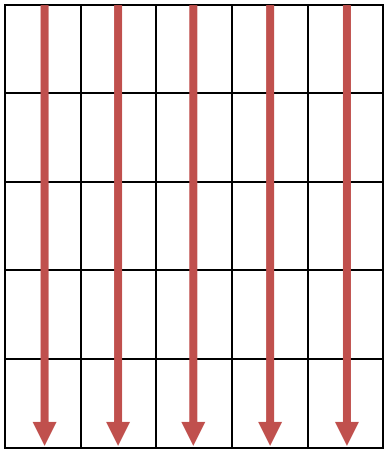

Рисунок 2. Обход массива по столбцам

9. В главной функции реализуйте подсчет сумм вторым способом и определите время его выполнения.
10. Проведите серию испытаний программы и заполните таблицу 2 полученными значениями:

Таблица 2. Обработка двумерного массива

№	Размер массива			Время нахождения сумм элементов столбцов	
	Количество столбцов	Количество строк	Количество элементов в массиве	При обходе по строкам	При обходе по столбцам
1	100	100			
2	500	500			
3	1000	100			
4	100	1000			
5	10	10000			
6	10000	10			

Выполните анализ полученных данных и сделайте вывод.

Задания для самостоятельной работы

1. Разработайте приложение, демонстрирующее работу различных методов сортировки и сравнение их эффективности по количеству выполненных операций сравнения и перестановки элементов. В качестве тестовых данных используйте целочисленный одномерный массив. По результатам выполнения задания заполните таблицу 3.

Таблица 3. Сравнение эффективности алгоритмов сортировки

№	Метод сортировки	Количество операций		
		Заполненный случайными числами	**Отсортированный по возрастанию**	**Отсортированный по убыванию**
1	Обменом по возрастанию (пузырьковая)			
2	Вставками по возрастанию			
3	Выбором по возрастанию			