*1)* ***Тестирование****программного обеспечения (Software Testing) —****проверка****соответствия реальных и ожидаемых результатов поведения****программы****, проводимая на конечном наборе****тестов****, выбранном определённым образом. Цель****тестирования****—****проверка****соответствия ПО предъявляемым требованиям, обеспечение уверенности в качестве ПО, поиск очевидных ошибок в программном обеспечении, которые должны быть выявлены до того, как их обнаружат пользователи****программы****.*

* ***2)***

***Отладка****- это процесс локализации и исправления ошибок в****программе****. Принципы****отладки****.*

*3)  Шестой****этап****–****тестирование****программы. Различается два вида****тестирования****: автономное и комплексное. При автономном****тестированию****подвергаются отдельные****программные****модули, из которых состоит****программный****комплекс. Комплексное****тестирование****заключается в проверке всего****программного****комплекса.*

*4) Структурный подход базируется на том, что известна структура и алгоритмы тестируемого ПО, («стеклянный ящик»). Тесты строят так, чтобы проверить правильность реализации заданной логики в коде программы. Функциональный подход основывается на том, что структура ПО не известна («черный ящик»), тесты строят, опираясь на функциональные спецификации.*

*5)* ***Метод******покрытия******операторов****. Целью этого****метода******тестирования****является выполнение каждого****оператора****программы хотя бы один раз. Если для****тестирования****задать значения переменных А = 2, B = 0, Х=3,****будет****реализован путь асе, т. е. каждый****оператор****программы выполнится один раз (рис. Л****5****.1, а). Но если внести в алгоритм ошибки — заменить в первом условии and на or, а во втором Х> 1 на Х< 1*

* *6)**Более сильный****метод******тестирования****известен как****покрытие******решений****(****покрытие****переходов). Согласно данному****методу****должно****быть****написано достаточное число****тестов****, такое, что каждое направление перехода должно****быть****реализовано по крайней мере один раз.****Покрытие******решений****обычно удовлетворяет критерию****покрытия****операторов.*

*7)* ***Покрытие****решения или****покрытие****Ветвления — это****метод******тестирования****, целью которого является обеспечение того, чтобы каждое из возможных ответвлений от каждой точки принятия решения выполнялось хотя бы один раз, и, таким образом, обеспечение выполнения всего достижимого кода. То****есть****каждое решение принимается по-разному, верно и ложно.*

*8)* ***Комбинаторные******методы****построения****тестов****основаны на разделении каждого тестового воздействия на ряд элементов и построении****тестов****как всевозможных комбинаций полученных элементов, объединяемых по определенным правилам.****Комбинаторные******методы****дают более высокую полноту****покрытия****, чем вероятностные, и при этом требуют ненамного больше ресурсов.*

*9)* ***Эквивалентное******разбиение****– это****метод****тестирования «черного ящика». Идея тестирования по****методу******разбиения****классов эквивалентности состоит в том, чтобы исключить набор входных данных, которые заставляют систему вести себя одинаково и давать одинаковый результат при тестировании программы.*

*10)* ***Анализ******граничных******значений****заключается в получении тестовых вариантов, которые анализируют****граничные******значения****. Данный способ тестирования дополняет способ разбиения на эквивалентности. Основные отличия****анализа******граничных******значений****от разбиения на эквивалентности: Тестовые варианты создаются для проверки только границ классов эквивалентностей.*

*11)* ***Анализ******причинно****-****следственных******связей****– это структурированный****метод****, применяемый для определения возможных причин нежелательного события или проблемы.*

*Программа*

using System;

namespace labwork

{

class Program

{

static void Main(string[] args)

{

string str1 = "Исходный массив";//

string str2 = "Массив сумм положительных элементов и их количества";//

Console.Write("Введите размерность матрицы N = ");//

int n = int.Parse(Console.ReadLine());//

int[,] mas1 = new int[n,n];//

CreateArray(mas1);//

PrintArray(mas1, str1);//

Console.Write("Введите М = ");//

int m = int.Parse(Console.ReadLine());//

int[,] mas2 = new int[2,m];//

EditArray(mas1, mas2, m);//

PrintArray(mas2, str2);//

FindMax(mas2, n);//

Console.ReadKey();//

}

//Заполнение массива случайными целыми числами от -20 до 20

private static void CreateArray(int[,] mas)

{

Random rand = new Random();//

for (int i = 0; i < mas.GetLength(0); ++i)//

{

for (int j = 0; j < mas.GetLength(1); ++j)//

{

mas[i,j] = rand.Next(-20,20);//

}

}

}

//Вывод массива

private static void PrintArray(int[,] mas, string str)

{

Console.WriteLine(str + ":");//

for (int i = 0; i < mas.GetLength(0); ++i)//

{

for (int j = 0; j < mas.GetLength(1); ++j)//

{

Console.Write(mas[i, j] + " ");//

}

Console.WriteLine();//

}

}

//Находим сумму и количество положительных элементов в последних М строках матрицы

private static void EditArray(int[,]mas1, int[,]mas2, int m)

{

int x = 0;//

for (int i = mas1.GetLength(0) - m; i < mas1.GetLength(0); ++i)//

{

int sum = 0;//

int count = 0;//

for (int j = 0; j < mas1.GetLength(1); ++j)//

{

if (mas1[i, j] > 0)//

{

sum += mas1[i, j];//

++count;//

}

}

mas2[0, x] = sum;//

mas2[1, x] = count;//

++x;//

}

}

//Определяем, в какой из M строк сумма элементов максимальна

private static void FindMax(int[,]mas, int n)

{

int max = mas[0, 0];//

int number = 0;//

for (int i = 0; i < mas.GetLength(1); ++i)//

{

if(mas[0,i] > max) //

{

max = mas[0, i];//

number = i;//

}

}

Console.WriteLine("Сумма элементов максимальна в {0} строке исходной матрицы", number + n - mas.GetLength(1));

}

}

}