Лекция 1. Основные понятия системного программирования.

Введение

Название «Системное программирование» неразрывно связано с операционными системами компьютеров (ОС). Исторически сложилось так, что первыми большими программами были программы ОС ЭВМ.

Прикладные программы были ограничены объем оставшейся свободной оперативной памяти компьютера и в основном решали вычислительные задачи.

В настоящее время трудно различить, какие программы являются более сложными – прикладные или системные т.к. некоторые прикладные программы фактически являются большими системными проектами и по своей сложности не уступают программам ОС.

В дисциплине «Системное программирование» рассматриваются алгоритмы и решения задач операционной системы.

1.1 Понятие и назначение операционных систем

В понятие компьютера включаются как физические или аппаратные, так и логические или информационные ресурсы.

К физическим ресурсам относятся различные устройства компьютера – процессор, устройства ввода-вывода, память и т.д.

К логическим ресурсам относят данные и программы, по которым работает компьютер.

Все ресурсы компьютера обычно называют системными ресурсами.

Тогда операционную систему можно определить как комплекс программ, которые обеспечивают доступ и управление системными ресурсами компьютера.

Программы, работающие на компьютере под управлением ОС, называют пользовательскими программами.

Пользовательские программы, предназначенные для решения одной задачи, называются приложением.

Если ОС позволяет выполнять только одну пользовательскую программу, то ОС называется однопользовательской или однопрограммной.

Если ОС позволяет одновременно выполнять несколько пользовательских программ, то ее называют многопользовательской или мультипрограммной.

Если ОС может работать только на компьютере с одним процессором, то ее называют однопроцессорной.

Если ОС может работать на компьютере, аппаратный ресурс которого может содержать один или несколько процессоров, то ОС называется мультипроцессорной.

Операционные системы, работающие без участия пользователя в режиме реального времени, называют ОС реального времени. Такие

операционные системы предназначены для управления некоторых технологических процессов.

1.2 Понятие и назначение интерфейса прикладного программирования Win32 API.

Каждая ОС предоставляет пользователю возможность использования практически всех своих системных ресурсов – пользовательский интерфейс, включающий наборы типов, констант, переменных, функций и классов для программирования приложений, работающих под управлением ОС.

Все, работающие в настоящий момент ОС Windows, имеют практически одинаковый интерфейс программирования приложений – Win32 API (Application Programming Interface).

Естественно Win32 API используется не только в прикладном, но и в системном программировании. Условно все функции Win32 API можно подразделить на следующие категории:

- базовые сервисы (Base Services);
- библиотека общих элементов управления (Common Control Library);
- интерфейс графических устройств (Graphics Devices Interface);
- сетевые сервисы (Network Services);
- интерфейс пользователя (Uses Interface);
- управление доступом для Windows (Windows Access Control);
- оболочка Windows (Windows Shell);
- информация о системе Windows (Windows System Information).

В дисциплинах, связанных с разработкой прикладных программ, в основном изучаются функции интерфейса пользователя.

Функции сетевого сервиса изучаются в дисциплинах, связанные с работой локальных вычислительных сетей.

Интерфейс графических устройств интенсивно используется при разработке различных игровых программ.

Назначение и использование остальных функций Win32 API в основном изучается в дисциплине «Системное программирование».

1.3 Понятие объектов и дескрипторов в Windows

Использование функций Win32 API базируется на использовании объектов, при этом необходимо помнить, что «классическое» определение объекта как переменной класса стало применяться после разработки функций Win32 API.

Объектом в Windows называется структура данных, которая содержит системный ресурс. Фактически объект Windows это область памяти, выделенную Windows и содержащую информацию об объекте. Объекты создаются специальными функциями и доступ к объектам возможен только с помощью функций ОС. Win32 API может создавать объекты трех категорий:

– объекты интерфейса пользователя (User);

– объекты интерфейса графических устройств (Graphics Devise Interface);

– объекты ядра операционной системы (Kernel).

Мы будем рассматривать только объекты ядра операционной системы.

При создании объекта ему присваивается имя (идентификатор), которое называется дескриптором (handle). В ОС дескриптор объекта представляет собой запись в таблице дескрипторов, которая содержит адрес объекта и средства для идентификации типа объекта.

В Win32 API дескрипторы имеют тип HANDLE.

При обращении к объекту необходимо указывать его дескриптор.

По окончании работ с объектом его необходимо закрывать.

1.4 Понятие динамически подключаемых библиотек

«Динамически подключаемые библиотеки (dynamic-link libraries, DLL) — краеугольный камень операционной системы Windows, начиная с самой первой её версии. В DLL содержатся все функции функций Win32 API. Три самые важные DLL: Kernel32.dll (управление памятью, процессами и потоками), User32.dll (поддержка пользовательского интерфейса, в том числе функции, связанные с созданием окон и передачей сообщений) и GDI32.dll (графика и вывод текста).

В Windows есть и другие DLL, функции которых предназначены для более специализированных задач. Например, в AdvAPI32.dll содержатся функции для защиты объектов, работы с реестром и регистрации событий, в ComDlg32.dll — стандартные диалоговые окна (вроде FileOpen и FileSave), а ComCtl32.dll поддерживает стандартные элементы управления.

Зачем нужны динамически подключаемые библиотеки? Вот лишь некоторые из причин, по которым нужно применять DLL:

– расширение функциональности приложения. DLL можно загружать в адресное пространство процесса динамически, что позволяет приложению, определив, какие действия от него требуются, подгружать нужный код. Поэтому одна компания, создав какое-то приложение, может предусмотреть расширение его функциональности за счет DLL от других компаний.

– возможность использования разных языков программирования. У Вас есть выбор, на каком языке писать ту или иную часть приложения;

– более простое управление проектом. Если в процессе разработки программного продукта отдельные его модули создаются разными группами, то при использовании DLL таким проектом управлять гораздо проще.

– экономия памяти. Если одну и ту же DLL использует несколько приложений, в оперативной памяти может храниться только один её экземпляр, доступный этим приложениям.

– разделение ресурсов. DLL могут содержать такие ресурсы, как шаблоны диалоговых окон, строки, значки и битовые карты (растровые изображения). Эти ресурсы доступны любым программам.

– решение проблем, связанных с особенностями различных платформ. В разных версиях Windows содержатся разные наборы функций. Зачастую разработчикам нужны новые функции, существующие в той версии системы, которой они пользуются. Если Ваша версия Windows не поддерживает эти

функции, Вам не удастся запустить такое приложение: загрузчик попросту откажется его запускать. Но если эти функции будут находиться в отдельной библиотеке (DLL), то Вы загрузите программу даже в более ранних версиях Windows.» (Джеффри Рихтер Windows для профессионалов, стр. 476-477).

1.5 Разработка DLL библиотека для некоторых задач

Разработаем учебную DLL библиотеку для задач, связанных с некоторой обработкой целых чисел в одномерном массиве. Это чисто учебная задача, в которой рассматривается технология создания DLL библиотеки. Одновременно рассмотрены вопросы использование DLL библиотеки для работы в консольном приложении языка C# .

Пусть наша DLL библиотека будет содержать только три метода работы с массивом целых чисел:

- поиск максимального значения;
- сортировку в порядке убывания;
- вычисления общей суммы всех значений массива.

Запустим Visual Studio 2008 и перейдем к созданию проекта. В качестве типа проекта укажем тип "Class Library". Переопределим поля окна создания DLL библиотеки в соответствии с рисунком 1.1.

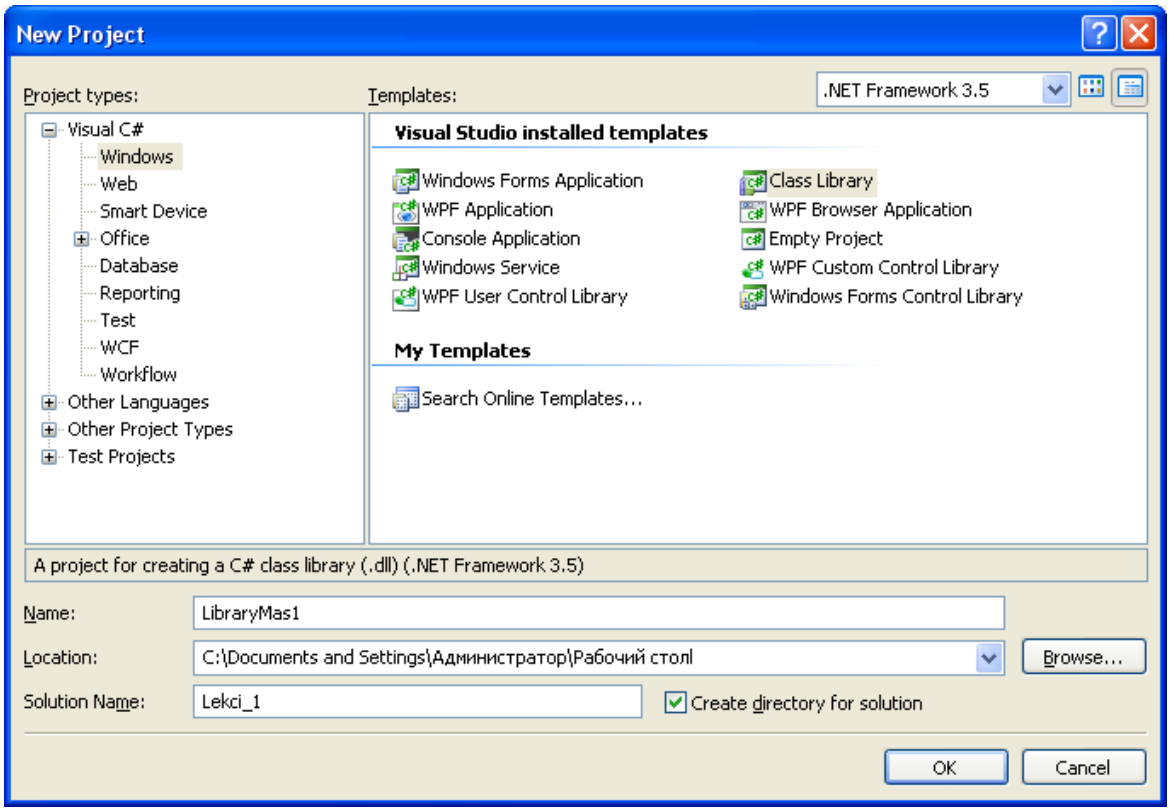


Рисунок 1.1 – Создание DLL библиотеки

В поле Name задается имя строящейся DLL. Пусть имя создаваемой DLL будет LibraryMas1.

В поле Location указывается путь к папке, где будет храниться Решение, содержащее проект. Укажем рабочий стол компьютера.

В окне Solition Name задается имя «Решения». Предлагается использовать имя Lekci_1, указывающее на то, что все проекты первой лекции вложены в одно «Решение».

В поле Solition Name выбран элемент "Create directory for solution", указывающий, что нужно создать новую папку для размещения «Решение».

После подтверждения типа проекта нажатием кнопки ОК получим следующий код:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LibraryMas1
{
    public class Class1
    {
    }
}
```

Изменим имя "Class1" на имя "MyMas" для этого в окне кода проекта выделяем имя изменяемого идентификатора, затем в главном меню выбираем пункт Refactor и подпункт Rename. В открывшемся окне указываем новое имя. Тогда будут показаны все места, требующие переименования идентификатора. В нашем случае будет только одна замена, но, в общем случае, замен может быть много, так что автоматическая замена всех вхождений крайне полезна.

Наполним создаваемую DLL библиотеку кодом трех методов в соответствии с условием задания.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LibraryMas1
{
    public class MyMas
    {
        public static int maxMas(ref int[] masi, int n)
        {
            int max = masi[0];
            for (int i = 1; i < n; i++)
                if (max < masi[i]) max = masi[i];
            return max;
        }
        public static void sortMas(ref int[] masi, int n)
        {
         int b = 0;
         for (int i = 0; i < n - 1; i++)
          for (int j = i + 1; j < n; j++)
```

```
            if (masi[i] < masi[j])
             { b = masi[i]; masi[i] = masi[j]; masi[j] = b; }
        }
        public static int symMas(ref int[] masi, int n)
        {
            int sym = 0;
            for (int i = 0; i < n; i++)
                sym=sym+masi[i];
            return sym;
        }
    }
}
```

На заключительном этапе создания DLL библиотеки необходимо выполнить два действия.

Во-первых, необходимо переименовав имя файла «Class1.cs» в имя «MyMas.cs». Так как имя файла, хранящего класс, и имя класса должны совпадать. Переименование имени файла делается непосредственно в окне проектов Solition Explorer. После выделения имени файла нажимаем правую кнопку мышки и выбираем команду Rename. Вводим новое имя – изменения происходят во всех элементах проекта.

Во-вторых, необходимо скомпилировать созданную DLL библиотеку для чего в Главном меню среды выберем пункт Build -> Build Solition. В результате успешной компиляции будет построен файл с расширением dll.

1.6 Использование созданной DLL библиотеки

Рассмотрим чисто учебную программу для работы с массивом, в которой используем созданную DLL библиотеку. Программа предназначена для решения следующей задачи.

Задача 1.1 Сформировать массив из 20 случайных целых чисел в диапазоне от минус 50 до 50. Напечатать его. Найти и напечатать сумму всех элементов массива. Выполнить сдвиг значений массива на 1 разряд влево. Выполнить сортировку элементов массива в порядке убывания. Предусмотреть меню программы.

Один из вариантов использования созданной DLL библиотеки предполагает создание приложения в одном с DLL «Решении». Для этого обычным способом открываем проект созданной ранее DLL библиотеке и в нем создаем проект консольного приложение в соответствии с рисунком 1.2.

В поле Name оставляем значение по умолчанию.

В поле Location оставляем значение по умолчанию.

В поле Solition выбираем значение "Add to Solution", указывающий, что создаваемое приложение будет добавлено в существующее «Решение».

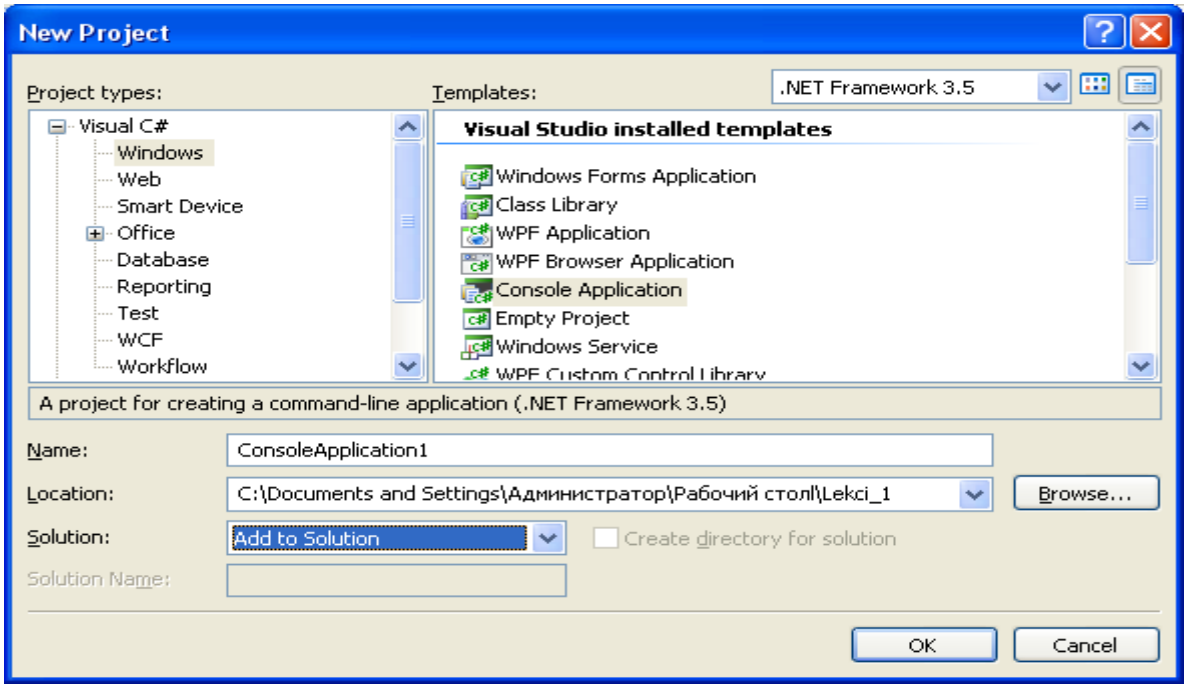


Рисунок 1.2 – Создание консольного приложения

Заготовка кода консольного приложения представлена на рисунке 1.3.

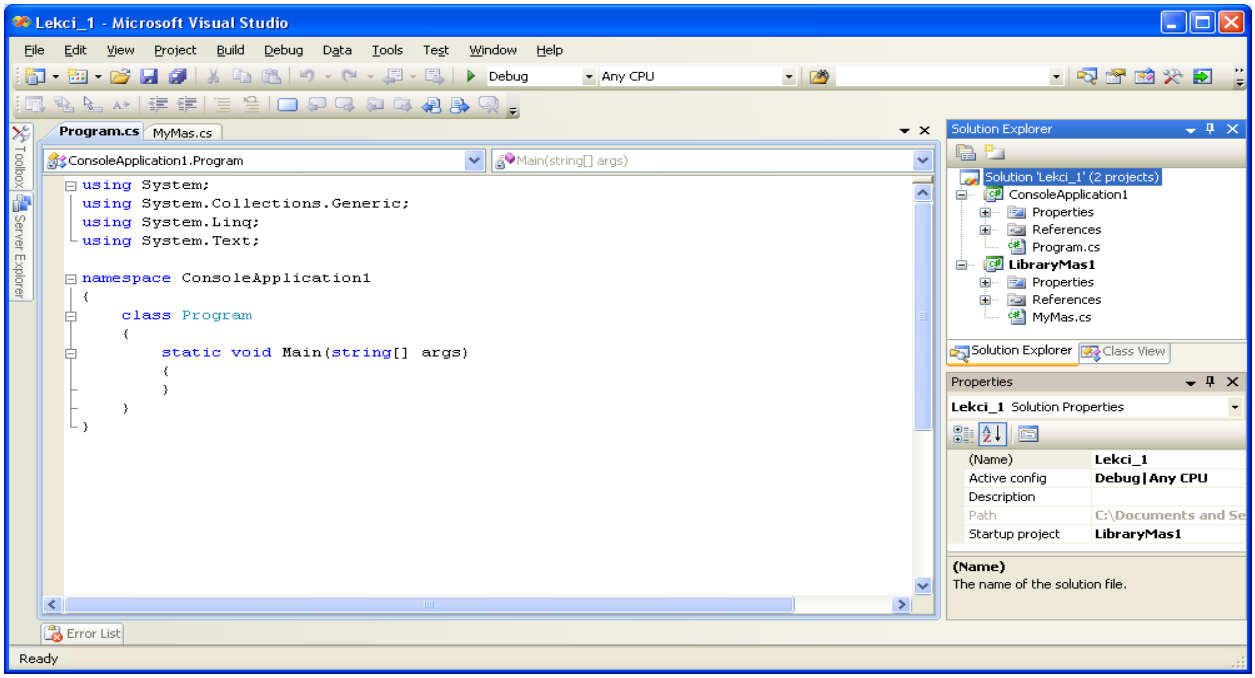
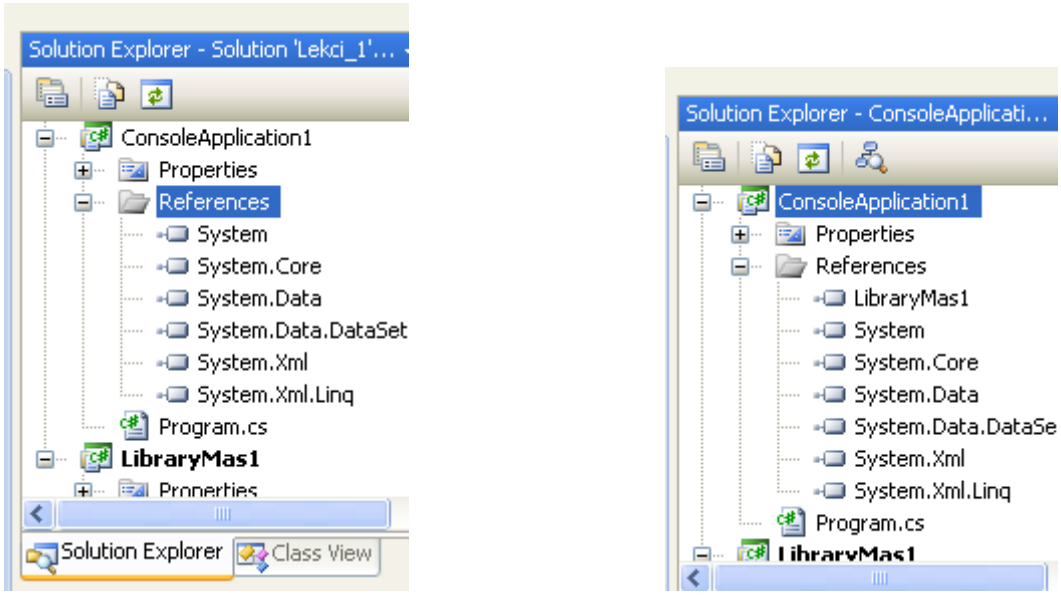


Рисунок 1.3 – Начальное окно создания консольного приложения

Обратите внимание, что окно «Solition Explorer» содержит два проекта.

Оба проекта находятся в одном решении, но не связаны друг с другом. Это легко проверить, если в окне «Solition Explorer» папку «References» - смотрите рисунок 1.4 – а.



a b

Рисунок 1.4 – Связи консольного приложения

Отсутствие связей делает невозможным использования методов DLL библиотеки в нашем консольном проекте.

Для организации связи консольного приложения с DLL (добавить ссылку на проект с DLL LibraryMas1) необходимо в окне «Solition Explorer» выбрать имя консольного приложения (ConsoleApplication1) и из контекстного меню, появляющегося при щелчке правой кнопки, выберем пункт меню "Add Reference". В открывшемся окне добавления ссылок выберем вкладку "Projects". Поскольку проект LibraryMas1 включен в «Решение», то он автоматически появится в открывшемся окне. Подтверждаем установку связи нажатием кнопки ОК. Ссылка на DLL LibraryMas1 появится в папке "References" консольного приложения. На рисунке 1.4 – b видно, что ссылка на DLL LibraryMas1 появилась вверху списка ссылок. Теперь проекты связаны и из консольного проекта доступны методы, предоставляемые DLL.

Ссылку на DLL LibraryMas1 можно установить, используя режим Project-> Add Reference.

Если ссылку нужно установить на проект, не включенный в «Решение», то в окне добавления ссылок нужно задать путь к проекту.

В соответствии с условиями задачи разрабатываем код программы.

Исходный код программы:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
```

```
 class Program
 {
  public static void sozd(ref int[] ma)
  {
   Random rnd = new Random();
   for (int i = 0; i < 20; i++)
    ma[i] = rnd.Next() % 101 - 50;
   Console.WriteLine("Массив создан !!");
  }
  public static void zadvig(ref int[] ma)
  {
   int k;
   for (int i = 0; i < 19; i++)
   {
    k = ma[i]; ma[i] = ma[i + 1]; ma[i + 1] = k;
   }
   Console.WriteLine("Сдвиг массива на 1 разряд выполнен !");
  }
  public static void prinmas(int[] ma)
  {
   for (int i = 0; i < 20; i++)
    Console.Write(" {0}", ma[i]);
   Console.WriteLine();
  }
  static void Main()
  {
   int[] a = new int[20];
   int k = 0;
   int s = 0,n = 20;
   string buf;
   while (k < 6)
   {
    Console.WriteLine("1 - Создать массив 20 чисел");
    Console.WriteLine("2 - Напечатать массив");
    Console.WriteLine("3 - Найти и напечатать сумму всех
элементов массива");
    Console.WriteLine("4 - Выполнить сдвиг значений массива на 1
разряд влево");
    Console.WriteLine("5 - Выполнить сортировку элементов
массива в порядке убывания");
    Console.WriteLine("6 - Выход из программы");
    Console.WriteLine("Введите пункт меню программы");
    buf = Console.ReadLine();
    k = Convert.ToInt32(buf);
    switch (k)
    {
     case 1: sozd(ref a); break;
     case 2: prinmas(a); break;
     case 3:
      {
       s = LibraryMas1.MyMas.symMas(ref a, n);
       Console.WriteLine("Сумма элементов массива = {0}",s);
      }; break;
```

```
            case 4: zadvig(ref a); break;
            case 5: LibraryMas1.MyMas.sortMas(ref a,n); break;
            default: break;
          }
        }
      }
    }
  }
```

Если попытаться запустить проект на выполнение, то появится сообщение, что не назначен запускаемый проект. Это возможно когда в «Решении» несколько проектов. Чтобы определить первый запускаемый проект необходимо находиться в окне редактора этого проекта и задать следующую команду: Project -> Set as StartUp Project.
Работа программы:

1 - Создать массив 20 чисел
2 - Напечатать массив
3 - Найти и напечатать сумму всех элементов массива
4 - Выполнить сдвиг значений массива на 1 разряд влево
5 - Выполнить сортировку элементов массива в порядке убывания
6 - Выход из программы
Введите пункт меню программы
1
Массив создан !!
1 - Создать массив 20 чисел
2 - Напечатать массив
3 - Найти и напечатать сумму всех элементов массива
4 - Выполнить сдвиг значений массива на 1 разряд влево
5 - Выполнить сортировку элементов массива в порядке убывания
6 - Выход из программы
Введите пункт меню программы
2
-36 -2 -3 35 -10 28 -32 11 -21 -32 -35 13 6 3 -45 32 -27 -5 -36 15
1 - Создать массив 20 чисел
2 - Напечатать массив
3 - Найти и напечатать сумму всех элементов массива
4 - Выполнить сдвиг значений массива на 1 разряд влево
5 - Выполнить сортировку элементов массива в порядке убывания
6 - Выход из программы
Введите пункт меню программы
3
Сумма элементов массива = -141
1 - Создать массив 20 чисел
2 - Напечатать массив
3 - Найти и напечатать сумму всех элементов массива

4 - Выполнить сдвиг значений массива на 1 разряд влево
5 - Выполнить сортировку элементов массива в порядке убывания
6 - Выход из программы
Введите пункт меню программы
4
Сдвиг массива на 1 разряд выполнен !
1 - Создать массив 20 чисел
2 - Напечатать массив
3 - Найти и напечатать сумму всех элементов массива
4 - Выполнить сдвиг значений массива на 1 разряд влево
5 - Выполнить сортировку элементов массива в порядке убывания
6 - Выход из программы
Введите пункт меню программы
2
-2 -3 35 -10 28 -32 11 -21 -32 -35 13 6 3 -45 32 -27 -5 -36 15 -36
1 - Создать массив 20 чисел
2 - Напечатать массив
3 - Найти и напечатать сумму всех элементов массива
4 - Выполнить сдвиг значений массива на 1 разряд влево
5 - Выполнить сортировку элементов массива в порядке убывания
6 - Выход из программы
Введите пункт меню программы
5
1 - Создать массив 20 чисел
2 - Напечатать массив
3 - Найти и напечатать сумму всех элементов массива
4 - Выполнить сдвиг значений массива на 1 разряд влево
5 - Выполнить сортировку элементов массива в порядке убывания
6 - Выход из программы
Введите пункт меню программы
2
35 32 28 15 13 11 6 3 -2 -3 -5 -10 -21 -27 -32 -32 -35 -36 -36 -45
1 - Создать массив 20 чисел
2 - Напечатать массив
3 - Найти и напечатать сумму всех элементов массива
4 - Выполнить сдвиг значений массива на 1 разряд влево
5 - Выполнить сортировку элементов массива в порядке убывания
6 - Выход из программы
Введите пункт меню программы