Нижегородский государственный университет им. Н.И.Лобачевского

Факультет Вычислительной математики и кибернетики

Введение в интегрированную среду разработки Microsoft Visual Studio .NET 2003

Гергель В.П., профессор
Лабутин Д.Ю., ассистент
Гришагин А.В., аспирант
Кафедра МО ЭВМ

Введение

- Интегрированная среда разработки (Integrated Development Environment, IDE) Microsoft Visual Studio .NET 2003 (MS VS .NET 2003)
  - Последняя по времени выпуска версия популярной и широко используемой среды разработки,
  - Содержит многие дополнительные возможности для эффективной разработки сложного ПО,
  - Обеспечивает возможность использования всех преимуществ современной технологии Microsoft .NET.

Основные достоинства MS VS .NET 2003

- Повышение производительности труда разработчиков,
- Поддержка нескольких языков программирования,
- Единая модель программирования для всех приложений,
- Всесторонняя поддержка жизненного цикла разработки ПО

Используемый подход к изучению MS VS .NET…

- Излагаемый далее учебный материал ориентирован для начального знакомства со средой разработки MS VS .NET
- Предполагается, что обучаемый не знаком с принципами объектно-ориентированного программирования (ООП) и не имеет опыта разработки приложений с графическим интерфейсом пользователя для ОС Windows,
- В ходе обучения будет представлены начальные сведения об ОПП и рассмотрены возможности среды MS VS.NET, достаточные для разработки простых приложений для работы в текстовом режиме (в режиме консоли).

Используемый подход к изучению MS VS .NET

- При изучении среды разработки MS VS в качестве учебного примера будет использоваться задача "Упорядочивание (сортировка) массивов",
- Разработка программы сортировки будет происходить поэтапно с постепенным нарастанием сложности:
  - *Создание первой программы* в среде MS VS .NET (стандартный метод сортировки),
  - *Создание новых методов* в существующем классе программы на C# (разработка генератора исходных данных),
  - *Реализация алгоритма пузырьковой сортировки*

Начальное знакомство с ООП

- Основные понятия *объектно-ориентированного программирования* (*ООП*):
  - В ООП все данные (переменные) и обрабатывающие их процедуры и функции объединяются в *классы,*
  - Переменные класса называются *полями*, а функции и процедуры – *методами* класса,
  - Перед определения класса необходимо дать его описание. По описанию класса можно создать его реализацию – *объект*, в котором для входящих в класс полей будет выделена память

Начальное знакомство с ООП…

```
// Пример класса - Первый вариант учебной программы
using System;
class MainApp {
  public static void Main(string[] args)  {
    // определение массива и его инициализация
    int[] Data = { 9, 3, 7, 5, 6, 4, 8, 1};
    //
    // сортировка значений массива
    Array.Sort(Data);
    //
    // печать отсортированных данных
    Console.WriteLine("Печать отсортированных данных");
    for (int i=0; i<Data.Length; i++)
      Console.WriteLine("Data["+i+"] = " + Data[i]);
  }
}
```

Начальное знакомство с ООП…

- В приведенном примере программы содержится класс с именем **MainApp**, в котором имеется единственный метод **Main**,
- Особая роль метода **Main**
  - С этого метода начинается выполнение программы,
  - Метод **Main** не использует значения полей объектов (на это указывает ключевое слово **static** в описании метода) и, как результат, такой метод может быть вызван по имени класса

Начальное знакомство с ООП…

- Программный код метода **Main** обеспечивает:
  - Создание массива **Data** и его инициализацию при помощи списка начальных значений,

```
// определение массива и его инициализация
int[] Data = { 9, 3, 7, 5, 6, 4, 8, 1};
```

  - Сортировку значений массива **Data**, которая выполняется методом **Sort** класса **Array**

```
// сортировка значений массива
Array.Sort(Data);
```

Начальное знакомство с ООП…

- Класс **Array** является базовым и используется при создании массивов,
- Вызов метода класса осуществляется указанием имени класса, разделителя "." (точки) и, затем, имени метода
- Такой вызов возможен только для методов, описанных с ключевым словом **static** (как метод **Main**)
- В общем случае вместо имени класса должно указываться имя объекта

Начальное знакомство с ООП

- Вывод на экран значений упорядоченного массива осуществляется методом **WriteLine** класса **Console**

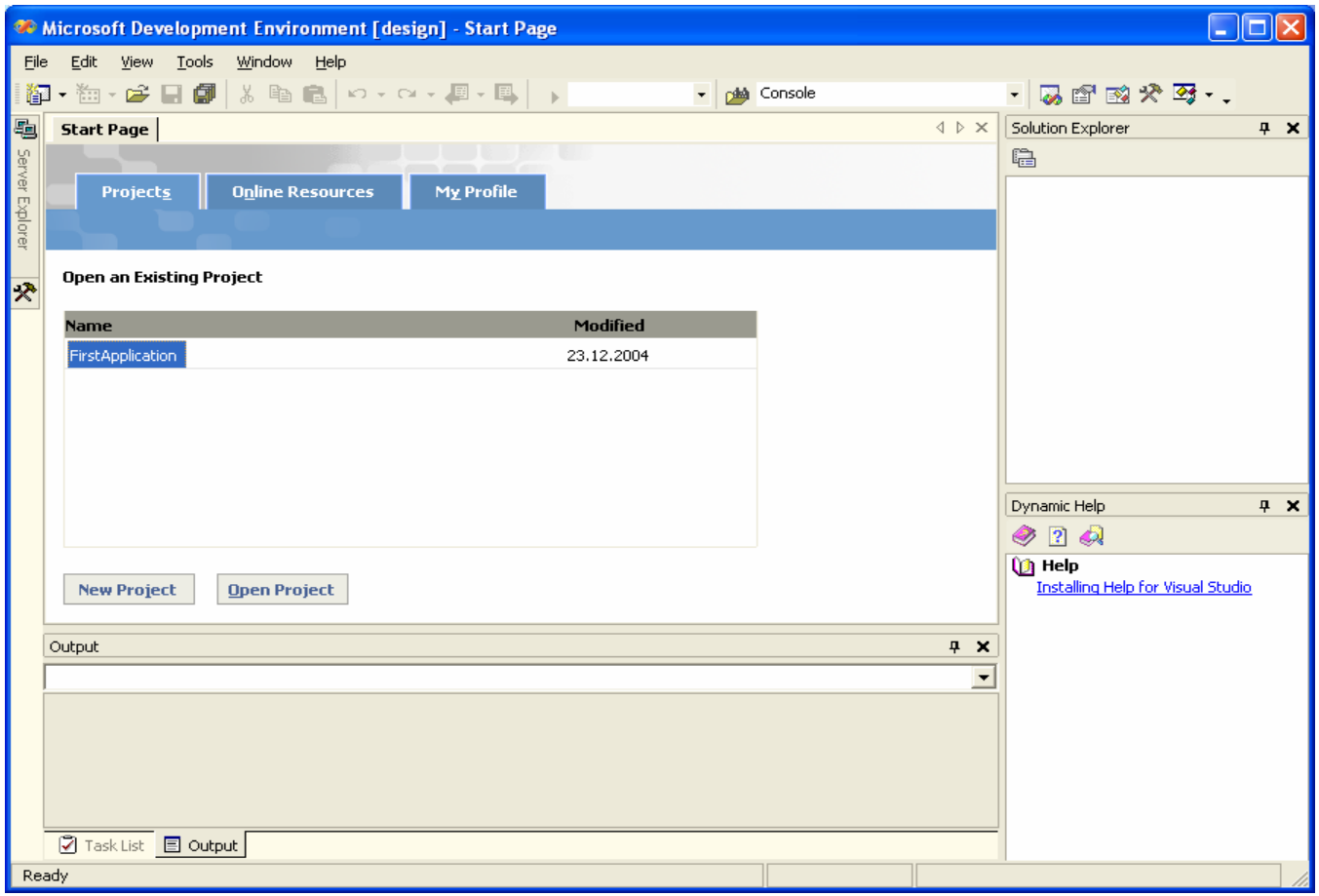
```
// печать отсортированных данных
Console.WriteLine("Печать отсортированных данных");
for (int i=0; i<Data.Length; i++)
   Console.WriteLine("Data["+i+"] = " + Data[i]);
```

- Класс **Console** отвечает за организацию ввода данных с клавиатуры и вывода информации на экран дисплея в текстовом режиме работы
- При выводе значений массива используется поле данных **Length** объекта **Data,** в котором хранится количество элементов массива

Разработка в MS VS .NET: Создание проекта…

- Программы (*приложения*), создаваемые в MS VS.NET, представляются в виде *проекта*, понимаемого как объединение всех необходимых для построения программы файлов,
- Близкие по назначению проекты могут объединяться в наборы проектов – *решения* (*solutions*)

Разработка в MS VS .NET: Создание проекта…



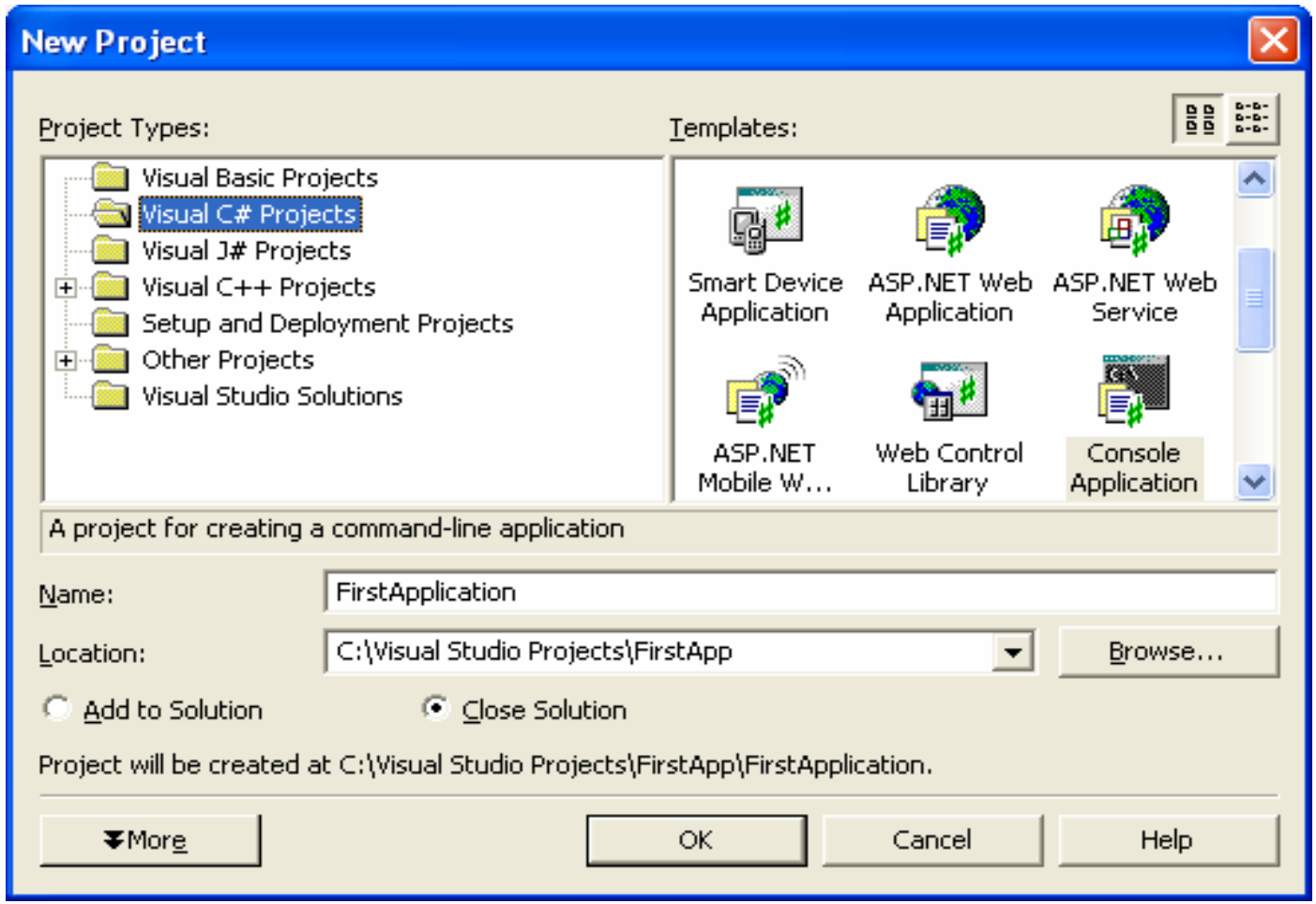
Общий вид среды разработки MS VS .NET (начало работы)

Создание нового проекта

- Запустите MS VS .NET,
- В диалоговом окне **Начальная страница** (Start Page) необходимо нажать кнопку **New Project**.
  В окне **New Project** нужно выполнить:
  - В поле **Name** задать имя создаваемого проекта,
  - В поле **Location** установить папку для размещения файлов проекта,
  - В области **Project Types** выбрать вариант **Visual C# Projects**,
  - В области **Templates** выбрать вариант **Console Application.**

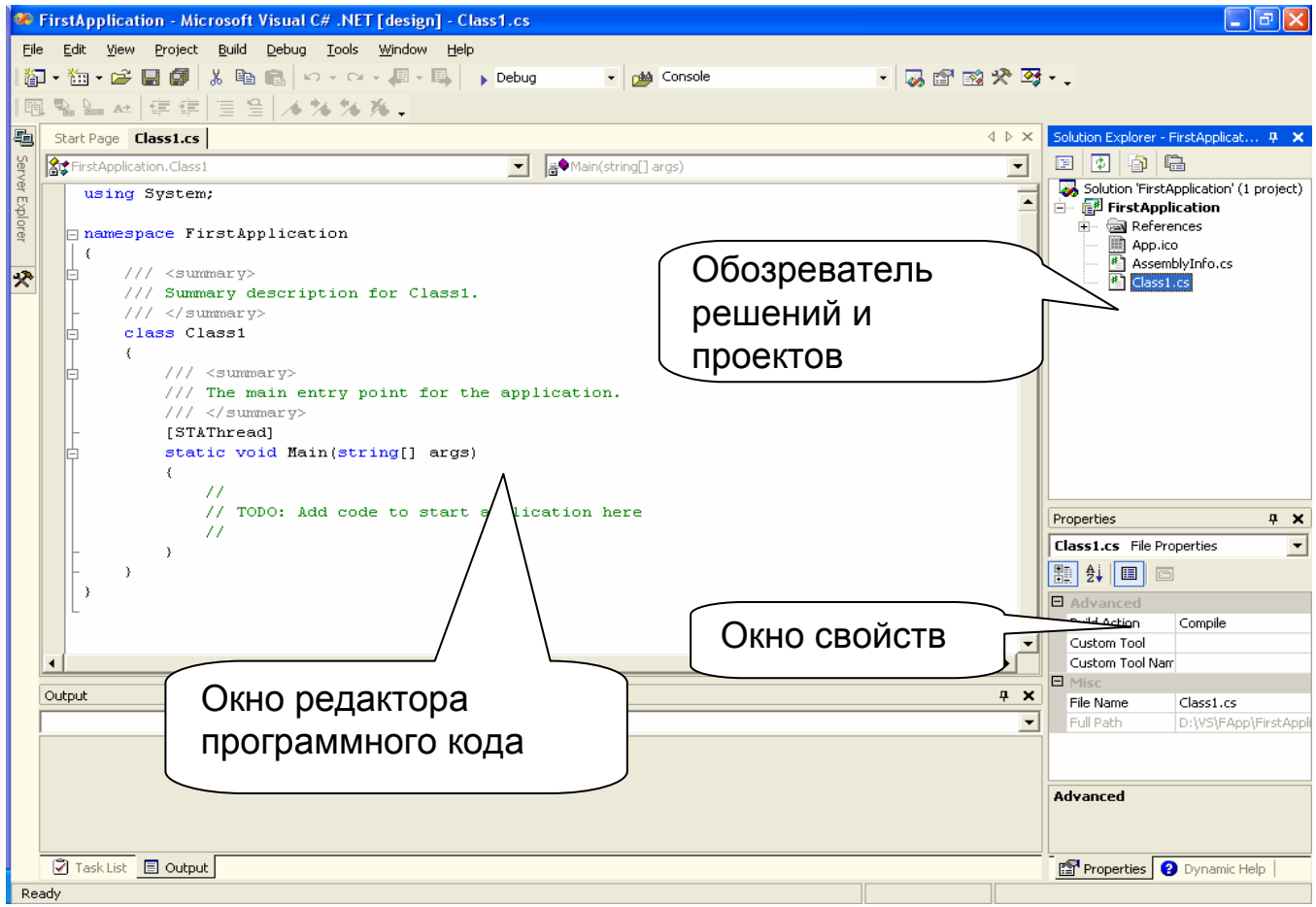
По завершении всех перечисленных действий необходимо нажать кнопку **OK**.

Разработка в MS VS .NET: Создание проекта



Диалоговое окно создания нового проекта

Общая характеристика среды разработки…



Общий вид окна среды разработки MS VS .NET

Общая характеристика среды разработки

- Как и другие программы ОС Windows, окно среды разработки содержит строку заголовка, меню и панели инструментов.
- В рабочей области среды разработки содержатся:
  - Окно *редактора* для ввода программного кода,
  - Окно *Обозревателя решений и проектов* (*Solution Explorer*),
  - Окно *Обозревателя свойств* (*Properties*) текущего (выбранного) объекта.

Разработка в MS VS .NET: Редактор кода…

- Редактор MS VS .NET обеспечивает поддержку всех стандартных действий, необходимых для подготовки программного кода – Ввод, Редактирование, Копирование, Вставка, Поиск и др.
- Редактор обладает большим набором дополнительных возможностей, значительно помогающих разработчикам создавать большие и сложные программные системы.

Разработка в MS VS .NET: Редактор кода…

- Редактор программного кода поддерживает оперативную проверку правильности ввода программы:
  - Ключевые слова алгоритмического языка опознаются и выделяются (обычно синим) цветом,
  - Если использование ключевых слов происходит неправильно, то данное ключевое слово будет подчеркиваться красной волнистой линией.
- При необходимости получения справочной информации нужно установить текстовый курсор на элемент программы, для которого требуется справка, и нажать клавишу F1

Разработка в MS VS .NET: Редактор кода

- Для возможности более быстрого и безошибочного набора программного кода редактор содержит службу **IntelliSense**, которая обеспечивает:
  - Отображение списка методов и полей для классов, структур, пространства имен и других элементов кода,
  - Отображение информации о параметрах для методов и функций,
  - Отображение краткого описания элементов кода программы,
  - Завершение слов при наборе наименований команд и имен функций,
  - Автоматическое сопоставление правильности расстановки скобок

Разработка в MS VS .NET: Ввод программы

- При создании проекта автоматически генерируется начальная заготовка (*оболочка*) программы, которая сдержит в себе все необходимые стандартные элементы кода
- Для завершения ввода текста программы необходимо:
  - ввести подходящее для разрабатываемого приложения имя класса (например, MainApp) и
  - заменить комментарий
    // TODO: Add code to start application here
    необходимым программным кодом

Разработка в MS VS .NET: Построение сборки

- Для выполнения программы, подготовленной на алгоритмическом языке, необходимо осуществить следующие действия:
  - Откомпилировать программу,
  - Собрать ("слинковать") вместе все модули программы при помощи специального редактора связей - получается готовая к исполнению *сборка* (assembly) на *промежуточном языке (Microsoft Intermediate Language, MSIL или просто IL),*
  - перевести сборку с промежуточного языка в *исполняемую программу* в командах компьютера - реализацию данного шага выполняют *JIT-компиляторы* MS .NET (англ. JIT – Just In Time – в нужный момент)

Разработка в MS VS .NET: Выполнение сборки

- Построение сборки - команда **Build** пункта меню **Build**,
- Запуск сборки на выполнение - команда **Start** пункта меню **Debug** (или клавиша **F5**),
- Построение сборки и сразу же запуск ее на выполнение - команда **Start** (построение сборки будет выполняться только при наличии изменений в коде программы после времени построения последнего варианта сборки)

Разработка в MS VS .NET: Обработка ошибок

- Построение сборки происходит только при отсутствии синтаксических ошибок в программе,
- При обнаружении ошибок компилятор в диалоговом окне **Microsoft Development Environment** выводит сообщение

  There were build errors. Continue ?
- В результате компиляция программы завершается, в окне **Output** выводится сообщение

  Build: 0 succeeded, 1 failed, 0 skipped

Поэтапная разработка программы

- Разработка программного обеспечения – это сложная профессиональная деятельность
- *Последовательная (поэтапная) разработка*

  На первом этапе реализации создается простая версия разрабатываемой программы, которая затем – на последующих этапах разработки - расширяется вплоть до получения полного варианта

Разработка в MS VS .NET: Добавление метода…

- Создадим метод **DataGenerator** для генерации сортируемого набора значений при помощи датчика случайных чисел:

```
// генератор данных
static void DataGenerator(int[] Vals) {
// Random - класс для генерации случайных чисел
   Random aRand = new Random();
// заполнение массива
    for (int i=0; i<Vals.Length; i++)
      Vals[i] = aRand.Next(100);
}
```

Разработка в MS VS .NET: Добавление метода…

- Метод **DataGenerator** описан как **static** - как результат, метод может быть вызван с использованием имени класса,
- Метод имеет входной параметр - массив **Vals**, который и должен быть заполнен генерируемым набором значений,
- Для генерации значений используется объект класса **Random**,
- Для генерации следующего случайного значения используется метод **Next**

Разработка в MS VS .NET: Добавление метода

- Текст метода **DataGenerator** целесообразно разместить перед текстом метода **Main,**
- При наличии метода для генерации значений участок программного кода по формированию массива должен быть заменен на следующий фрагмент:

```
// определение массива и его инициализация
const int N = 10;
int[] Data = new int[N];
DataGenerator(Data);
```

Разработка в MS VS .NET: Реализация алгоритма сортировки…

- На следующем этапе разработки проведем реализацию алгоритма пузырьковой сортировки,
- Метод основывается на базовой операции "сравнить и переставить" (compare-exchange)

```
// операция "сравнить и переставить"
if ( a[i] > a[j] ) {
   temp = a[i];
   a[i] = a[j];
   a[j] = temp;
}
```

Разработка в MS VS .NET: Реализация алгоритма сортировки…

- Применив процедуру "сравнить и переставить" для пар соседних элементов при последовательном проходе по упорядочиваемому набору можно обеспечить перемещение максимального значения данных в последний (верхний) элемент массива ("всплывание пузырька"),
- Для продолжения сортировки уже упорядоченный элемент может быть отброшен и действия алгоритма повторяются

```
// пузырьковая сортировка
for ( i=1; i<n; i++ )
  for ( j=0; j<n-i; j++ )
    <сравнить и переставить элементы (a[j],a[j+1])>
}
```

```
// метод пузырьковой сортировки
static void BubbleSort(int[] Vals) {
  double temp;
  for (int i=1; i<Vals.Length; i++)
    for (int j=0; j<Vals.Length-i; j++)
      // сравнить и переставить элементы
      if (Vals[j] > Vals[j+1]) {
        temp = Vals[j];
        Vals[j]  = Vals[j+1];
        Vals[j+1] = temp;
```

Разработка в MS VS .NET: Реализация алгоритма сортировки…

- В методе **Main** сохраним сортировку стандартным методом (для последующего сравнения результатов работы),
- Как результат, для исходного массива необходимо создать копию упорядочиваемых данных

```
// создание копии массива
int[] Data2 = new int[N];
Data.CopyTo(Data2,0);
// элементы массива Data копируются в массив
// Data2 начиная с индекса 0
```

- Добавим программный код в метод **Main** для сортировки и печати значений массива **Data2**

```
// вызов метода пузырьковой сортировки
BubbleSort(Data2);
//
// печать отсортированных данных
Console.WriteLine("Печать данных после пузырьковой
   сортировки");
for (int i=0; i<Data2.Length; i++)
  Console.WriteLine("Data2["+i+"] = " + Data2[i]);
```

Разработка в MS VS .NET: Проверка результатов

- Для проверки результатов пузырьковой сортировки можно подготовить автоматическую процедуру контроля

```
bool IsEqual = true;
for (int i=0; i<Data2.Length; i++)
   if ( Data[i] != Data2[i] ) IsEqual = false;
if ( IsEqual )
    Console.WriteLine("Результаты совпадают");
else
    Console.WriteLine("Ошибки в пузырьковой сортировке");
```

- При организации такой проверки делается предположение, что метод стандартной сортировки работает правильно

Разработка в MS VS .NET: Оценка эффективности...

- Для оценки эффективности используемых методов сортировки добавим в программу операторы для получения времени работы алгоритмов

```
// сортировка значений массива
long time = Environment.TickCount;
Array.Sort(Data);
time = Environment.TickCount - time;
Console.WriteLine("Время стандартной сортировки: "
   + time.ToString() +" msecs");
//
// сортировка при помощи пузырьковой сортировки
time = Environment.TickCount;
BubbleSort(Data2);
time = Environment.TickCount - time;
Console.WriteLine("Время пузырьковой сортировки: "
  + time.ToString() +" msecs");
```

Разработка в MS VS .NET: Оценка эффективности

- При проведении эксперимента при N=50000 были получены результаты

  Время стандартной сортировки: 16 msecs

  Время пузырьковой сортировки: 9078 msecs

т.е. алгоритм пузырьковой сортировки работает медленнее более чем в 500 раз по сравнению со стандартным методом сортировки из библиотеки платформы MS .NET.

Разработка в MS VS .NET: Тестирование и отладка

- Тестирование и отладка программ – это неотъемлемая часть разработки ПО,
- Тестирование и отладка могут занимать до 50% (а иногда и больше) от общего времени разработки,
- Опыт в тестировании и выявлении ошибок является значимым профессиональным качеством программиста

Разработка в MS VS .NET: Тестирование…

Под *тестовым заданием* (или просто *тестом*) обычно понимается набор исходных данных, при использовании которых в программе должны получиться заранее определенные результаты.

- Тесты должны подготавливаться на начальных этапах разработки программ,
- Успешность выполнения теста не является доказательством правильности программы,
- Подготавливаемые тесты должны проверять основные варианты выполнения программы и должны быть в максимальной степени направлены на выявление ошибочных ситуаций в реализованных алгоритмах решения поставленной задачи

Разработка в MS VS .NET: Тестирование

- Тест должен быть:
  - Достаточно небольшим (по объему исходных данных),
  - Быстро выполняемым,
  - Желательно должен иметь значения не только результирующих, но и промежуточных данных,
- Тест желательно должен иметь средства автоматической проверки результатов выполнения

Разработка в MS VS .NET: Отладка…

- Признаком наличия ошибки в программе является неправильное выполнение теста,
- Процесс выявления причин обнаруженной ошибки, определение места (локализация) ошибки в программе и исправление ошибочно реализованного программного кода называется *отладкой*

Разработка в MS VS .NET: Отладка…

- Один из эффективных способов отладки является визуальный анализ (*инспекция*) программного кода
- *Отладочное выполнение программы* (или *трассировка*) – метод поиска ошибок, в ходе которого работа программы может быть приостановлена для просмотра значений тех или иных переменных программы с целью обнаружения ситуаций, когда эти значения не соответствуют предполагаемых величинам

Разработка в MS VS .NET: Отладка…

Для обеспечения трассировки программ в MS VS .NET имеется:

- Команда **Step Info** (клавиша **F11**) обеспечивает последовательное, строка за строкой, выполнение программного кода программы (включая содержимое вызываемых методов),
- Команда **Step Over** (клавиша **F10**) обеспечивает, как и предшествующая команда **Step Info**, последовательное выполнение программы, но при этом вызов методов рассматривается как один неделимый шаг (т.е. без перехода внутрь вызываемых методов),

Разработка в MS VS .NET: Отладка…

Для обеспечения трассировки программ в MS VS .NET имеется: (продолжение)

- Команда **Step Out** (клавиша **Shift+F11**) обеспечивает выполнение всех оставшихся строк программного кода текущего выполняемого метода без останова, позволяя выполнить быстрый переход в последнюю точку вызова,
- Команда **Run to Cursor** (клавиша **Ctrl+F10**) обеспечивает выполнение без останова программного кода между текущей строкой останова и позицией курсора

Разработка в MS VS .NET: Отладка…

Удобным средством указания точек останова процесса выполнения программы является использование *контрольных точек* (breakpoints):

- Для определения контрольной точки необходимо щелкнуть мышкой на вертикальной полосе слева от нужной строки программного кода,
- Повторный щелчок отменяет установку контрольной точки,
- В ходе выполнения программы при попадании на контрольную точку происходит останов,
- Для продолжения работы необходимо выполнить команду **Continue** пункта меню **Debug**

Разработка в MS VS .NET: Отладка…

- Для наблюдения значений переменных достаточно расположить указатель мыши на имени переменной – в результате значение переменной появится в виде всплывающей подсказки
- Дополнительная возможность для наблюдения значений переменных состоит в использовании специальных окон наблюдения **Autos, Locals, Watch, Quick Watch**

Разработка в MS VS .NET: Отладка…

- Окно наблюдения **Autos:**
  - Отображает значения всех переменных, используемых в текущей и предшествующей строках точки останова программы,
  - Отображает названия переменных, их тип и значения,
  - Обычно располагается в нижней левой части экрана и для его высветки необходимо щелкнуть мышью на ярлычке с названием окна
- Окно **Locals** отличается от предшествующего окна **Autos** тем, что отображает значения всех переменных текущей области видимости (т.е. переменных текущего выполняемого метода или его локального блока)

Разработка в MS VS .NET: Отладка…

- Окна наблюдения **Watch:**
  - Таких окон в момент выполнения 4,
  - Состав отображаемых в них переменных может формироваться непосредственно программистом,
  - Для высветки нужного окна нужно последовательно выполнить команды **Debug\Windows\Watch\Watch <N>**, где N есть номер высвечиваемого окна,
  - Для добавления переменной в окно для наблюдения нужно указать мышкой необходимую переменную, нажать правую кнопку мыши и появившемся контекстном меню выполнить команду **Add Watch**

Разработка в MS VS .NET: Отладка

- Окно **Quick Watch:**
  - Позволяет изменять значения наблюдаемых переменных,
  - Для высветки окна необходимо выделить нужную переменную и выполнить команду **Quick Watch** пункта меню **Debug**
- Кроме перечисленных окон, может быть использовано:
  - Окно **this** для наблюдения за значениями полей объекта, метод которого выполняется в текущий момент времени,
  - Окно **Call Stack**, в котором отображается последовательность вызова методов, приведшая к обращению к текущему исполняемому методу

Разработка в MS VS .NET: Пример отладки…

- Внесем ошибку в программу – заменим оператор

  Vals[j+1] = temp;

  в методе пузырьковой сортировки **BubbleSort** на оператор

  Vals[j] = temp;

- Выполнение программы с внесенной ошибкой приведет к тому, что результирующий массив не будет являться отсортированным

Разработка в MS VS .NET: Пример отладки…

- Проверим, наблюдается ли ошибочный эффект при меньшем размере сортируемого набора данных,
- Установим для этого значения константы **N=3** и повторим выполнение программы – массив по прежнему остается не отсортированным,
- Возможные причины ошибки - проблема может состоять:
  - В неправильной работе алгоритма сортировки,
  - В наличии ошибки в операторах вывода значений массива

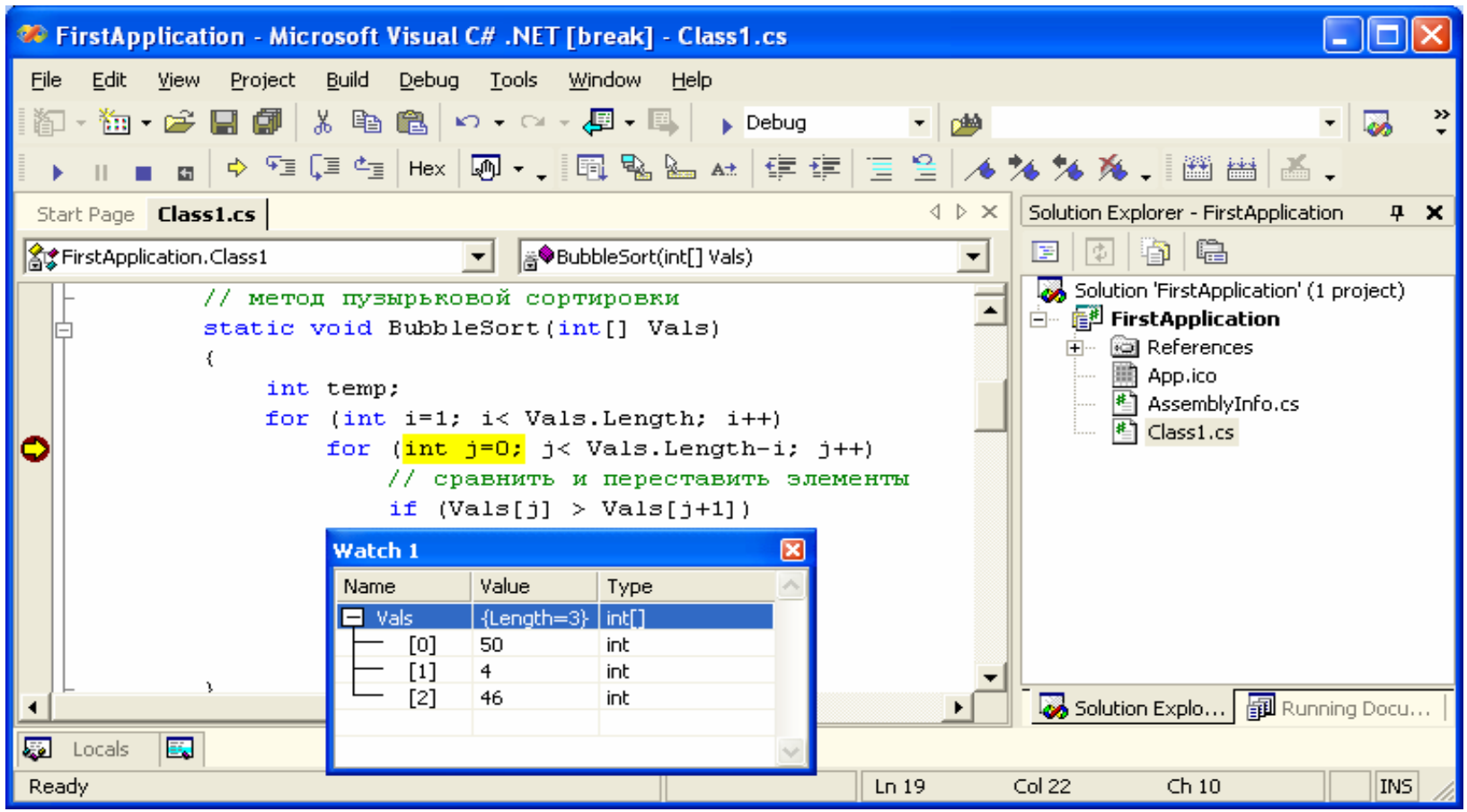
Разработка в MS VS .NET: Пример отладки…

- Вывод результатов стандартной сортировки сработал правильно,
- Автоматическая проверка результатов сортировки подтвердила, что результат пузырьковой сортировки является неправильным,
- Как результат, можно сделать вывод, что ошибки содержатся в методе пузырьковой сортировки **BubbleSort**

Разработка в MS VS .NET: Пример отладки…

- Установим контрольную точку на внутреннем операторе цикла в методе **BubbleSort** и запустим программу на выполнение,
- После останова добавим массив **Vals** в окно наблюдения **Watch 1** и запомним исходное значение сортируемого массива

Разработка в MS VS .NET: Пример отладки…



Состояние упорядочиваемого массива перед началом сортировки

Разработка в MS VS .NET: Пример отладки…

- Выполнение внешней итерации алгоритма пузырьковой сортировки должно привести к "всплыванию" максимального значения в последний элемент массива,
- Выполним команду **Continue** пункта меню **Debug** – результат выполнения не привел к желаемому эффекту, состояние сортируемого массива не изменилось,
- Как результат, можно сделать вывод, что внешняя итерация алгоритма сортировки работает неправильно

Разработка в MS VS .NET: Пример отладки…

- Поскольку сортируемый массив состоит только из трех элементов, следующая итерация алгоритма сортировки должна оказаться последней,
- В ходе ее выполнения значения двух первых элементов должно поменяться местами (на примере рассматриваемого набора значений)

Разработка в MS VS .NET: Пример отладки…

- Выполним трассировку – нажмем дважды клавишу **F10** и перейдем на операторы перестановки значений, т.е. сравнение пары значений выполняется корректно,
- Однако последующее выполнение операторов перестановки не приводит к нужному результату (значения не переставляются),
- Отсюда следует, что ошибка содержится в алгоритме перестановки пары значений,
- Анализ данного участка программного кода позволяет определить, что индекс элемента массива в последнем операторе должен быть **j+1**

Разработка в MS VS .NET: Пример отладки

- Для исправления ошибки:
  - Вносим необходимые изменения,
  - Выполняем программу и достигаем требуемого результата (!!!)
- Программа начинает работать правильно

Рекомендации по дальнейшему изучению MS VS .NET

- За рамками начального знакомства осталось:
  - создание Windows приложений с полноценным графическим интерфейсом,
  - разработка программ для сети Интернет,
  - реализация Web-сервисов и др.
- Среда разработки может быть расширена и дополнена самим программистом:
  - запоминание стандартно выполняемых действий в виде макросов,
  - подключение к среде разработки новых операций (надстроек) после их предварительной программной реализации,
  - расширение состава мастеров для автоматизированного выполнения сложных работ и т.п.

Литература

- Гарнаев А. Самоучитель Visual Studio .NET 2003. – СПб.: БХВ-Петербург, 2003.
- Пономарев В. Программирование на C++/C# в Visual Studio .NET 2003. – СПб.: БХВ-Петербург, 2004.
- Джонсон Б., Скибо К., Янг М. Основы Microsoft Visual Studio .NET. – М.: Русская редакция, 2003.

О проекте

Целью проекта является создание образовательного комплекса **CS101 "Основы программирования"**, предусматриваемый рекомендациями Computing Curricula 2001 Международных организаций IEEE-CS и ACM, для обучения на начальном этапе подготовки специалистов в области информатики, вычислительной техники и информационных технологий.

Образовательный комплекс включает **учебный курс "Введение в методы программирования" и лабораторный практикум "Методы разработки программного обеспечения на основе технологии Microsoft .NET"**, что позволяет органично сочетать фундаментальное образование в области программирования и практическое обучение методам разработки масштабного программного обеспечения на основе технологии Microsoft .NET. непосредственно на начальном этапе подготовки специалистов.

Проект выполнялся в Нижегородском государственном университете им. Н.И. Лобачевского на кафедре математического обеспечения ЭВМ факультета вычислительной математики и кибернетики (http://www.software.unn.ac.ru). Выполнение проекта осуществлялось при поддержке компании Microsoft.