Основы платформы Microsoft .NET

Тема:

Введение в интегрированную среду разработки Microsoft Visual Studio .NET 2003

Введение

Интегрированная среда разработки (*Integrated Development Environment*, *IDE*) Microsoft Visual Studio .NET 2003 (MS VS .NET 2003) является последней по времени выпуска версией популярной и широко используемой среды разработки профессионального программного обеспечения (ПО) производства компании Microsoft. Объединяя в своем составе все положительные стороны предыдущих версий, MS V S.NET 2003 обеспечивает возможность использования всех преимуществ современной технологии Microsoft .NET. В числе основных достоинств MS VS .NET 2003, по

достоинству оцененных сообществом профессиональных программистов, можно отметить следующие моменты:

- **Повышение производительности труда разработчиков** - Среда разработки Visual Studio .NET продолжает традиции корпорации Microsoft в области предоставления эффективных инструментальных средств для разработчиков сложного ПО. Обеспечивая среду разработки для всех языков программирования, дополненную набором окон с интуитивно понятными инструментальными средствами, контекстной справкой и автоматизированными механизмами выполнения разнообразных задач разработки, Visual Studio .NET позволяет в сжатые сроки проводить профессиональную разработку программ различного назначения;
- **Поддержка нескольких языков программирования** – В большинстве профессиональных групп разработчиков, как правило, используется несколько языков программирования – для поддержки такой практики в Visual Studio .NET впервые была обеспечена возможность использования сразу нескольких языков в рамках одной и той же среды. Благодаря применению общего конструктора для компонентов, для форматов XML и HTML, а также наличию единого отладчика, Visual Studio .NET предоставляет разработчикам эффективные средства, независимые от языка программирования. Разработчикам ПО при использовании Visual Studio .NET уже не придется ограничиваться одним языком программирования, адаптируя свою рабочую среду к особенностям этого языка. Более того, Visual Studio .NET позволяет программистам многократно использовать уже имеющиеся у них наработки, а также навыки разработчиков, создающих свои программы на разных языках программирования;
- **Единая модель программирования для всех приложений** - При создании приложений ранее разработчикам приходилось использовать различные приемы программирования, которые существенным образом зависели от типа приложения — технологии разработки клиентского программного обеспечения, общедоступных веб-приложений, программного обеспечения для мобильных устройств и бизнес-логики промежуточного уровня значительно различались между собой. Среда разработки Visual Studio .NET решает данную проблему, предоставляя в распоряжение разработчиков единую модель создания приложений всех категорий. Эта интегрированная модель обладает привычным и одновременно интуитивно понятным интерфейсом, позволяя разработчикам использовать свои навыки и знания для эффективного создания широкого спектра приложений,
- **Всесторонняя поддержка жизненного цикла разработк**и - Среда Visual Studio .NET обеспечивает поддержку всего жизненного цикла разработки: начиная с

планирования и проектирования через разработку и тестирование и вплоть до развертывания и последующего управления. Обеспечивая возможность легкого расширения среды разработки посредством включения продуктов независимых разработчиков, Visual Studio .NET предоставляет всестороннюю адаптируемую среду для создания всех приложений, жизненно необходимых для успешной работы современных компаний.

Излагаемый далее учебный материал предназначен для начального знакомства со средой разработки MS VS .NET для всех желающих познакомиться с увлекательным миром профессионального программирования. При этом предполагается, что читатель не знаком с принципами объектно-ориентированного программирования (ООП) и не имеет, соответственно, опыта разработки приложений с графическим интерфейсом пользователя для операционной системы Windows. В ходе изложения материала в разделе будут представлены начальные сведения об ОПП и рассмотрены возможности среды MS VS .NET, достаточные для разработки простых приложений для работы в текстовом режиме ОС Windows (в режиме консоли).

Наиболее эффективно данный учебный материал может быть использован на первых начальных занятиях для студентов младших курсов, осваивающих программирование в рамках учебного курса "Введение в методы программирования" (или других, близких по назначению, курсов).

Описание учебной задачи

В качестве учебной проблемы, на примере которой будут рассматриваться правила разработки программ в среде Visual Studio .NET 2003, будет использоваться задача лабораторной работы 1 учебного практикума по курсу "Введение в методы программирования". Для последовательного изложения необходимых сведений по среде MS VS .NET, разработка программы сортировки будет происходить поэтапно с постепенным нарастанием сложности:

1. *Создание первой программы* в среде MS VS .NET - на этом этапе будет подготовлен вариант программы, в котором массив для сортировки формируется при помощи списка начальных значений, а для сортировки данных используется метод, имеющийся в составе библиотек .NET;

2. *Создание новых методов* в существующем классе программы на C# - на данном этапе разработанная ранее программа будет расширена методом для заполнения

сортируемого массива набором значений, генерируемых при помощи датчика случайных чисел;

3. *Реализация алгоритма пузырьковой сортировки* – на этом этапе будет сформирован полный вариант программы, включающий реализацию алгоритма пузырьковой сортировки и оценку времени работы разных алгоритмов упорядочивания данных.

Начальное знакомство с объектно-ориентированным программированием

Начало работы в среде MS VS .NET для начинающих программистов при разработке первых простых программ наталкивается на определенные логические трудности. Языки программирования платформы MS.NET являются, как правило, объектно-ориентированными, а освоение ООП все-таки целесообразно проводить только после получения некоторого практического опыта разработки алгоритмов и их реализации в виде сравнительно простых программ. Именно на эту начальную стадию изучения программирования и ориентирован данный учебный материал. Как результат, далее будет дана очень краткая характеристика основных понятий объектно-ориентированного программирования, а разработка программа в среде MS V S.NET будет показываться на примере конкретных практических действий:

– Как добавить программный код в метод существующего класса (первый вариант программы сортировки);

– Как добавить новый метод в существующий класс программы (второй вариант программы сортировки);

– Как добавить описание данных в существующий класс программы (полный вариант программы сортировки).

Данных сведений будет достаточно для разработки простых программ в среде MS VS .NET даже без предварительного изучения ООП. Опыт разработки алгоритмов для решения учебных задач даст реальную основу для перехода к успешному освоению объектно-ориентированного программирования.

Итак, дадим краткую характеристику основных понятий *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 использованием класса необходимо дать его описание. По описанию класса можно создать его реализацию – *объект* (иногда еще

говорят – *экземпляр класса*), в котором для входящих в класс полей будет выделена память. В этой памяти можно будет хранить значения полей объекта и выполнять их обработку. По описанию класса можно создавать любое необходимое количество объектов.

В качестве примера класса можно привести программный код, который будет использоваться в первом варианте программы сортировки:

```
// Первый вариант программы сортировки
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]);
   }
}
```

В приведенном примере программы содержится класс с именем **MainApp**, в котором имеется единственный метод **Main**. Следует отметить, что метод **Main** в классах на языке C# имеет особое значение – именно с этого метода начинается выполнение программы (как результат, в программе метод **Main** должен присутствовать хотя бы в одном классе). Кроме того, 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.

Программный код метода **Main** обеспечивает выполнение:

– Создание массива **Data** и его инициализацию при помощи списка начальных значений;

– Сортировку значений массива **Data**, которая выполняется методом **Sort** класса **Array**; класс **Array** является базовым и используется при создании массивов; следует обратить внимание, что 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 же случае вместо имени класса должно указываться имя объекта;

– Вывод на экран значений упорядоченного массива; в соответствии с только что приведенными пояснениями можно понять, что вывод осуществляется методом **WriteLine** класса **Console** и этот метод также описан с ключевым словом **static**; класс Console

отвечает за организацию ввод данных с клавиатуры и вывод информации на экран дисплея в текстовом режиме работы. Следует также обратить внимание, что при выводе значений массива используется поле данных **Length** объекта **Data**. В этом поле хранится количество элементов массива; обращение к полю объекта осуществляется также как к методу объекта, т.е. необходимо указать имя объекта, затем поставить разделитель "." (точку) и далее привести имя требуемого поля.

Разработка программ в среде MS VS .NET

Рассмотрим основные понятия и последовательность действий, необходимых для разработки простой программы в среде MS VS .NET.

Создание проекта

Программы (более часто именуемые *приложениями*), создаваемые в среде разработки MS VS.NET, представляются в виде *проекта*, понимаемого как объединение всех

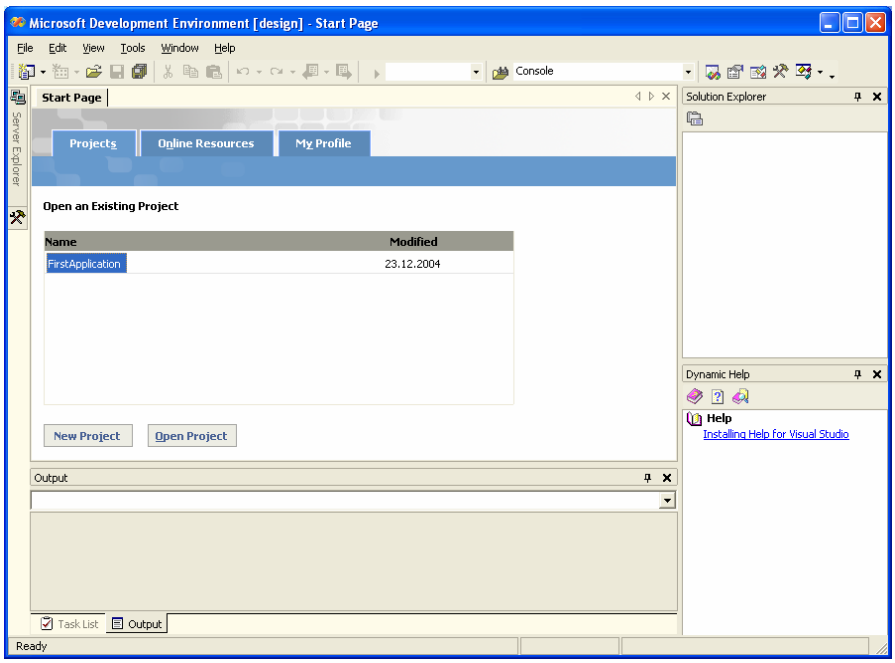

Рис. 1.1. Общий вид среды разработки MS VS.NET после начала работы

необходимых для построения программы файлов. Близкие по назначению проекты могут объединяться в наборы проектов – *решения* (*solutions*). Как результат, при начале разработки программы необходимо создать проект, размещаемый в создаваемое по умолчанию решение.

Для создания проекта необходимо выполнить:

1. Запустите MS VS .NET (это действие может зависеть от настройки параметров системы – в большинстве случаев, для этого необходимо выбрать пункт **Все программы** после нажатия кнопки **Пуск** и выполнить команду **Microsoft Visual Studio .NET 2003**. Общий вид окна среды разработки после начало работы показан на рис.1.1.

2. Для создания нового проекта в диалоговом окне **Начальная страница** (Start Page) необходимо нажать кнопку **New Project**. В появившемся диалоговом окне **New Project** (см. рис. 1.2) нужно выполнить следующие действия:

- В поле **Name** задать имя создаваемого проекта (например, **FirstApplication**),
- В поле **Location** установить папку для размещения файлов проекта (например, **C:\Visual Studio Projects\FirstApp**),
- 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.

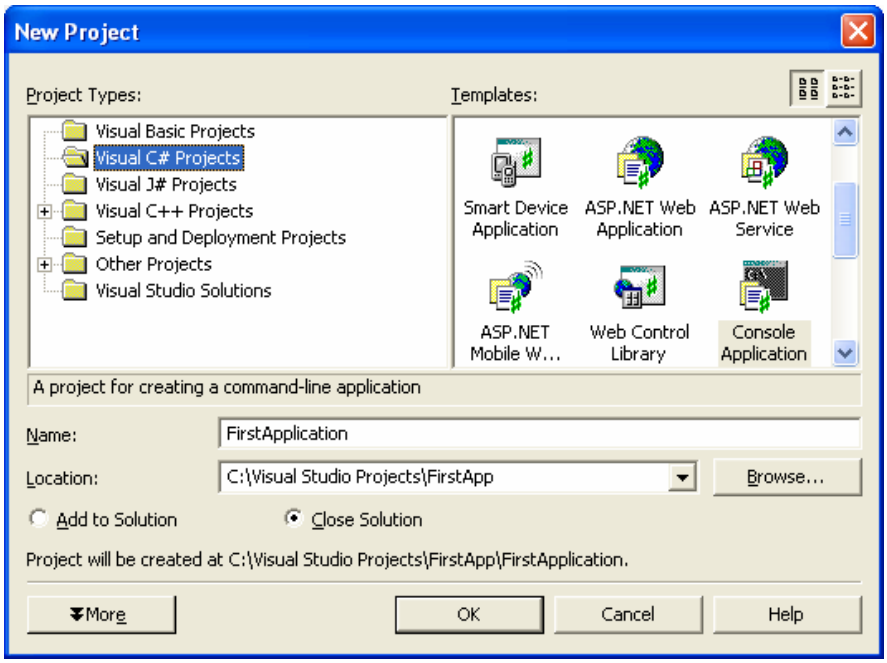

Рис. 1.2. Диалоговое окно создания нового проекта

Следует отметить, что в рассматриваемом примере для размещения создаваемых проектов используется папка **Visual Studio Projects** на диске **C:**. Понятно, что программист может выбирать и другое месторасположение проектов; в качестве рекомендации можно посоветовать использовать некоторый другой рабочий диск (**C:** обычно является системным для операционной системы Windows).

Общая характеристика среды разработки

Общий вид окна среды разработки MS VS.NET после создания проекта показан на рис. 1.3.

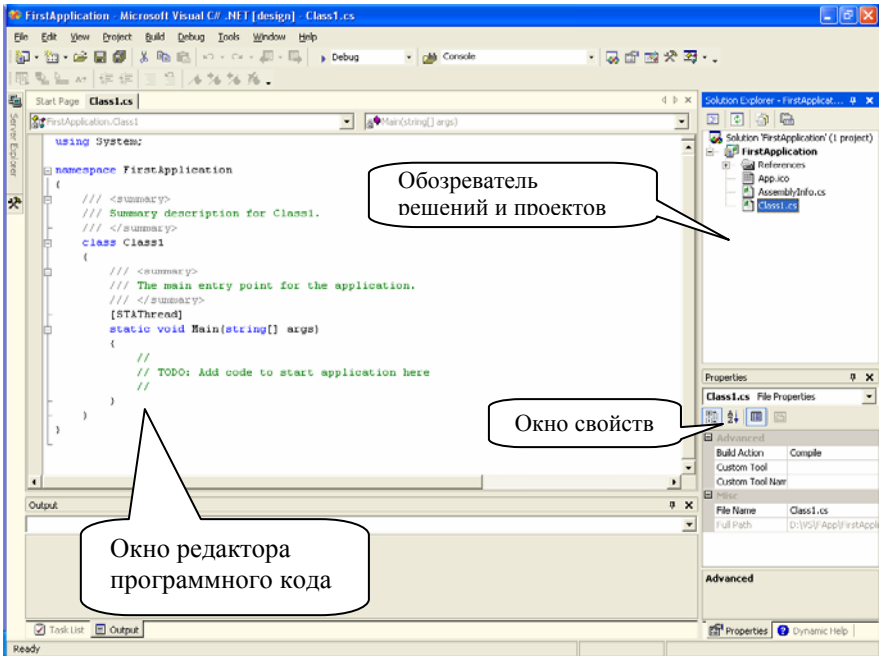

Рис. 1.3. Общий вид окна среды разработки MS VS .NET

Как и другие окна ОС Windows, окно среды разработки содержит строку заголовка, меню и панели инструментов. В рабочей области среды разработки содержится окно *редактора* (см. рис. 1.3) для ввода программного кода, окно *Обозревателя решений и проектов* (*Solution Explorer*) и окно *Обозревателя свойств* (*Properties*) текущего (выбранного) объекта.

Ввод и редактирование программного кода

Основным для начальной работы в среде разработки MS VS .NET является редактор программного кода. Редактор MS VS .NET обеспечивает все стандартные действия, которые доступны для любого другого редактора (набор программного кода, редактирование, копирование, вставка, поиск и т.д.) и, кроме того, обладает большим набором дополнительных возможностей, значительно помогающих разработчикам создавать большие и сложные программные системы. Ориентируясь на начальное знакомство со средой разработки, рассмотрим несколько полезных свойств редактора кода, которые могут оказать заметное практическое содействие программисту при подготовке даже самых простых программ.

Автоматическая проверка правильности текста

Редактор программного кода поддерживает оперативную (в процессе ввода текста) проверку правильности ввода программы – ключевые слова алгоритмического языка опознаются и выделяются (обычно синим) цветом. При этом, если использование ключевых слов происходит неправильно (не соответствует синтаксическим правилам языка программирования) данное ключевое слово будет подчеркиваться красной волнистой линией. Как результат, при наборе программного кода следует внимательно следить за цветовой окраской ключевых слов и выделением синтаксически неправильных конструкций программного кода.

Получение справочной информации

Для получения справочной информации нужно установить текстовый курсор на элемент программы, для которого необходимо наличие справки, и нажать клавишу F1 (следует отметить, что справка будет выдана на английском языке; кроме того, получение справки возможно только в случае, если на компьютере установлена справочная служба MSDN Library – данная служба поставляется при приобретении MS VS .NET). Как правило, получаемая информация содержит всю необходимую для программиста информацию, обеспечивая, тем самым, действенную помощь при разработке программ. В большинстве случаев, справочная информация дополнена примерами практически использования рассматриваемых элементов.

Автоматизированная поддержка набора текста

Для оказания максимального содействия программисту для быстрого и безошибочного набора программного кода в редакторе среды MS VS.NET имеется специальная служба **IntelliSense**, которая обеспечивает:

– Отображение списка методов и полей для классов, структур, пространства имен и других элементов кода – см. рис. 1.4. (вывод списка осуществляется автоматически после ввода имени и последующего за ним одного из разделителей "." (точка), "->" или "::"; выбор нужного варианта может быть выполнен, например, при помощи двойного щелчка мыши на требуемой строке списка или при помощи последовательного нажатия клавиш <Tab> и <Enter>);

– Отображение информации о параметрах для методов и функций – вывод данной информации также осуществляется автоматически после ввода имени метода или функции;

– Отображение краткого описания элементов кода программы (вывод описания происходит при наведении указателя мыши на нужный элемент кода);

– Завершение слов при наборе наименований команд и имен функций (для использования этой возможности следует набрать несколько первых символов вводимого имени и нажать одновременно клавиши <Ctrl> и <Пробел>, выбор нужного варианта, как и ранее, производится при помощи двойного щелчка мыши или клавиш <Tab> и <Enter>);

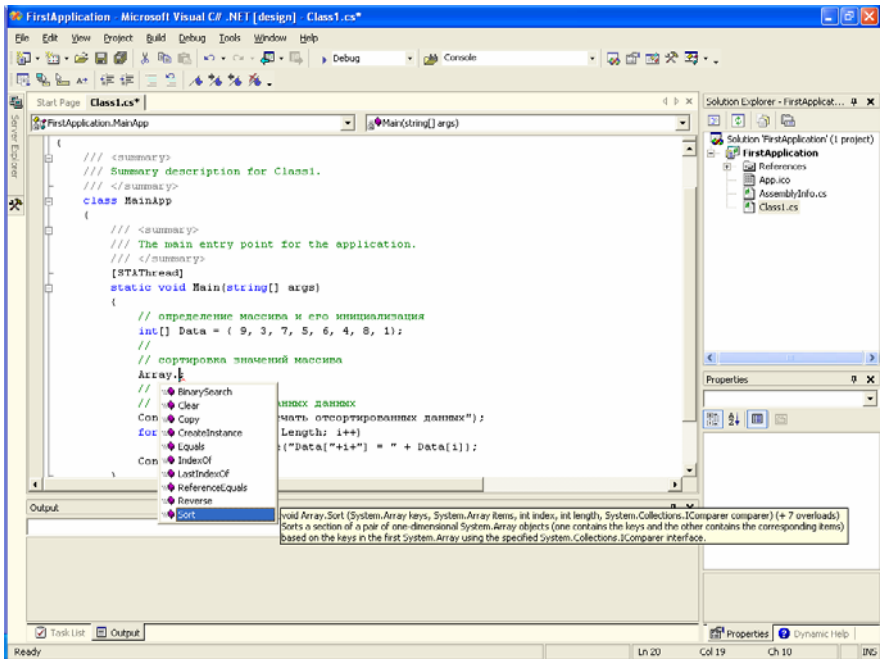

Рис. 1.4. Пример отображения списка методов для массива Data

– Автоматическое сопоставление правильности расстановки скобок (набираемые скобки },],), #endif выделяются более темным цветом вместе с соответствующей открывающейся скобкой).

Следует отметить, что служба **IntelliSense** может быть отключена при соответствующей настройки параметров среды MS VS.NET.

Ввод первого варианта программы

Внимательно проанализируйте программный код, появившийся в редакторе после создания проекта. Как можно увидеть, при создании проекта автоматически генерируется и начальная заготовка (*оболочка*) программы, которая сдержит в себе все необходимые стандартные элементы. Данную заготовку можно скомпилировать и запустить на выполнение – она не содержит ошибок, но при этом не выполняет каких-либо нужных нам действий. Все, что дальше необходимо выполнить – это ввести более подходящее для программы имя класса (например, MainApp) и заменить комментарий

```
// TODO: Add code to start application here
```

необходимым программным кодом (наберите для этого текст метода **Main** из примера в разделе, посвященном краткому описанию ООП).

Построение сборки и запуск ее на выполнение

Процедура построения исполняемой программы

Для выполнения программы, подготовленной на алгоритмическом языке, необходимо осуществить достаточно длинную цепочку технологических действий – программу нужно откомпилировать и убедиться, что в ней отсутствуют синтаксические ошибки, далее программу надо собрать ("слинковать") вместе со всеми используемыми служебными модулями - в результате в рамках платформы MS .NET получается готовая к исполнению *сборка* (assembly) на *промежуточном языке* (Microsoft Intermediate Language, MSIL или просто IL). При запуске на выполнение сборка должна быть переведена с промежуточного языка в *исполняемую программу* в командах компьютера, на котором будет работать сборка – реализацию данного шага выполняют *JIT-компиляторы общей среды выполнения* (Common Language Runtime, CLR) платформы MS .NET (англ. JIT – Just In Time – в нужный момент). Более подробно данная информация рассмотрена в главе Введение в технологию Microsoft .NET.

Запуск сборки на выполнение

Построение сборки (команда **Build** пункта меню **Build**) и запуск ее на выполнение (команда **Start** пункта меню **Debug**) могут быть выполнены раздельно, однако достаточным является и применение одной команды **Start**, т.к. при выполнении этой команды проверяется соответствие имеющейся сборки и программного кода в редакторе и, если после времени построения последнего варианта сборки в программном коде были поведены какие-либо изменения, то автоматически будет вызван JIT-компилятор и сформирован новый вариант сборки. Выполнение команды **Start**, как можно увидеть в пункте меню, можно обеспечить и простым нажатием клавиши **F5**.

При запуске на выполнение подготовленной на предшествующих шагах программы могут возникнуть две различные ситуации:

– Программа подготовлена правильно, в этом случае запуск сборки произойдет без обнаружения ошибок, на экране дисплея мелькнет окно вывода результатов и практически моментально исчезнет. Для наблюдения итогов выполнения программы окно вывода надо задержать – это можно обеспечить, например, при помощи вызова процедуры ввода

```
// приостановка окна вывода
  Console.ReadLine();
```

перед завершением метода **Main**. В этом случае при переходе на вызов метода **ReadLine** выполнение программы будет приостановлено и мы получим возможность рассмотрения результатов вывода программы – см. рис. 1.5. Для продолжения работы программы

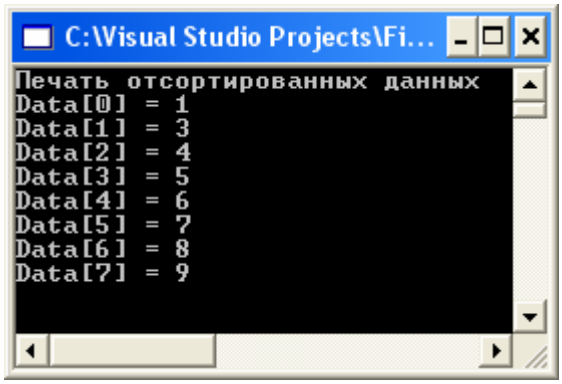

Рис. 1.5. Окно вывода результатов программы

достаточно нажать клавишу **Enter** (отметим еще раз, что метод ввода **ReadLine** используется в данном примере только для организации приостановки окна вывода, а не для реального ввода данных);

– Другая ситуация возникает при обнаружении ошибок при построении сборки – в этом случае, естественно, выполнение сборки невозможно и для ее подготовки необходимо найти и исправить все имеющие ошибки в программном коде программы.

Обработка синтаксических ошибок

При обнаружении синтаксических ошибок, компилятор в диалоговом окне **Microsoft Development Environment** выводит сообщение

```
There were build errors. Continue ?
```

для ответа на которое следует нажать кнопку **Нет**. В результате компиляция программы завершается, в окне **Output** выводится сообщение

```
Build: 0 succeeded, 1 failed, 0 skipped
```

и для каждой обнаруженной ошибки в окне **Task List** приводится ее краткое описание. Так, например, если в правильной программе нашего учебного примера убрать символ ";" в операторе **using System**, сообщение об ошибке имеет вид (см. рис. 1.6):

```
; expected
```

Сообщение об ошибке можно выделить и, нажав клавишу **F1**, получить справочную информацию по допущенной ошибке. Нажав клавишу **Enter** (или выполнив двойной щелчок мыши) можно перейти в окно редактора на строчку с оператором, в котором была

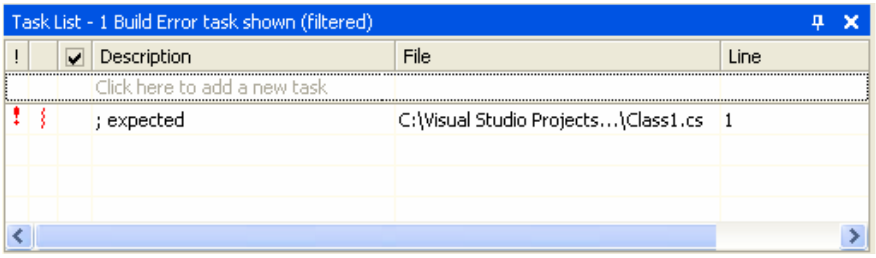

Рис. 1.6. Сообщение об ошибке в окне Task List

обнаружена ошибка. Следует отметить, что "правильное понимание" выдаваемых компилятором сообщений требует определенной практики (если бы компилятор мог абсолютно точно выделять ошибочные ситуации в программе, то тогда исправление ошибок могло бы происходить автоматически). Так, например, если в нашей правильной программе в методе **Main** удалить открывающую фигурную скобку, то в окне **Task List** будет выведено 13 (!) сообщений об ошибках.

Поэтапная разработка программы

Разработка программного обеспечения является достаточно сложной профессиональной деятельностью. Понимание постановки задачи, поиск и разработка алгоритмов решения, проектирование структуры программы, программная реализация, поиск и исправление ошибок, доказательство правильности работы созданной программы – и это лишь "верхушка айсберга" программирования. Успешное разрешение проблем на всех перечисленных этапах разработки программных систем, безусловно, требует длительной подготовки и обширной практики. Здесь же в рамках данного руководства в качестве начального шага в нужном направлении дадим "базовую" рекомендацию – все выполняемые действия при создании программ в максимальной степени должны быть понятными и простыми. В числе основных способов достижения подобной цели – *последовательная (поэтапная) разработка*, когда на первом этапе реализации создается некоторая простая версия разрабатываемой программы, которая затем последовательно расширяется вплоть до получения полного (изначально определенного) варианта. Такой подход обладает множеством преимуществ – в частности, работающий вариант программы появляется на самых начальных этапах разработки, расширения программы на каждом этапе реализации являются небольшими по размеру, что существенно снижает трудоемкость поиска и исправления ошибок и т.д. В полном объеме методика разработки программ является предметом рассмотрения *технологии программирования* (для начального освоения технологических аспектов разработки программ могут быть рекомендованы, например, учебные издания [9-10]; более систематическое изложение вопросов технологии программирования излагается в работе [11]).

В рамках данного учебного руководства, как отмечалось и ранее, выполним расширение первоначального простого варианта программы для автоматического заполнения сортируемого массива набором значений, генерируемых при помощи датчика случайных чисел, а также выполним реализацию одного из методов упорядочивания данных - алгоритма пузырьковой сортировки.

Добавление нового метода класса

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

```
// генератор данных
static void ValsGenerator(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);
}
```

Дадим краткие пояснения для метода **DataGenerator**:

– Метод описан как **static** – как результат, метод может быть вызван по имени класса;

– Метод имеет входной параметр – массив **Vals**, который и должен быть заполнен генерируемым набором значений;

– Для генерации значений используется объект класса **Random** (следует обратить внимание, как происходит создание этого объекта);

– Для генерации следующего случайного значения используется метод **Next**; параметр метода задает для датчика случайных чисел максимально-возможное генерируемое значение (минимально-возможное значение равно нулю).

Набор метода **DataGenerator** целесообразно выполнить перед методом **Main**.

При наличии метода для генерации значении 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);
```

После подготовки программы следует выполнить несколько экспериментов для проверки правильности работы программы. Важно отметить, что использование генератора позволяет выполнить программу при нескольких различающих наборах исходных данных.

Подготовка полного варианта программы

Расширим создаваемую учебную программу собственно реализованным методом упорядочивания данных - выберем для этого хорошо известный и сравнительно простой алгоритм пузырьковой сортировки. Метод основывается на базовой операции "сравнить и переставить" (compare-exchange), состоящей в сравнении той или иной пары значений из сортируемого набора данных и перестановки этих значений, если их порядок не соответствует условиям сортировки

```
// операция "сравнить и переставить"
```

```
if ( a[i] > a[j] ) {
  temp = a[i];
  a[i] = a[j];
  a[j] = temp;
}
```

Последовательно применив данную процедуру сравнения всех соседних элементов в результате прохода по не упорядочиваемому набору данных в последнем (верхнем) элементе массива оказывается максимальное значение ("всплывание пузырька"); далее для продолжения сортировки этот уже упорядоченный элемент может быть отброшен и действия алгоритма следует повторить

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

В результате реализация алгоритма пузырьковой сортировки может быть выполнена следующим образом:

```
// 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;
  }
}
```

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

```
// создание копии массива
int[] Data2 = new int[N];
Data.CopyTo(Data2,0);
```

Как видно из примера, создание копии обеспечивается методом **CopyTo** (значение 0 указывает, что копирование необходимо выполнить с нулевого элемента массива). Разместить данный код следует сразу после заполнения исходного массива (после вызова метода **DataGenerator**).

Для использования пузырьковой сортировки можно продублировать уже имеющийся в Main программный код (заменив массив Data на массив Data2)

```
// сортировка при помощи 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]);
```

Полный вариант нашей учебной программы подготовлен. Далее необходимо, как и ранее, построить сборку и выполнить несколько экспериментов для проверки правильности работы алгоритма сортировки. Здесь следует сделать важное замечание – сравнение результатов работы программы в большинстве случаев является достаточно трудоемким занятием. Проверить визуально правильность работы можно только для небольших массивов данных и для ограниченного набора контрольных примеров. Крайне желательно пытаться автоматизировать процесс проверки результатов – так, для проверки совпадения результатов сортировки можно подготовить специальный программный код:

```
// автоматическая проверка результатов сортировки
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).

Выполним еще один дополнительный шаг – определим время, которое затрачивается выбранными методами сортировки на выполнение. Окружим для этого вызовы методов сортировки операторами получения системного времени

```
// сортировка значений массива
long time;
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");
```

Поскольку быстродействие современных компьютеров является высоким, для получения различимых диапазонов времени количество сортируемых данных должно быть достаточно большим (например, N=50000 – при таком количестве значений

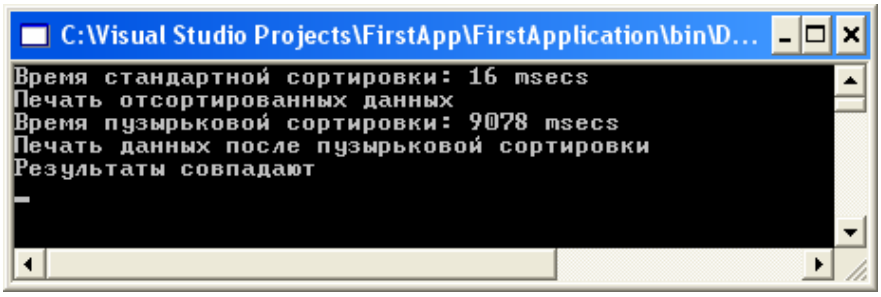

Рис. 1.7. Время выполнения методов сортировки

операторы печати содержимого массивов нужно удалить или закомментировать). В этом случае, на компьютере, на котором происходило выполнение примера, были получены результаты – см. рис. 1.7 (алгоритм пузырьковой сортировки работает медленнее более чем в 500 раз по сравнению со стандартным методом сортировки из библиотеки платформы . NET.

Тестирование и отладка программ

Практические навыки эффективного тестирования программ, умение быстро определять причины самых сложных ошибок составляют основу профессии программиста. Из опыта выполнения реальных проектов разработки сложного программного обеспечения следует, что до половины общего времени работ занимают как раз этапы тестирования и отладки программ. Безусловно, полное изучение данных вопросов требует и достаточно большого объема учебного времени, и наличия определенного практического опыта разработки сложных программ. С другой стороны, поскольку этот материал рассчитан на студентов младших курсов, начинающих осваивать программирование, вопросы тестирования и отладки также должны быть рассмотрены (пусть хотя бы на самом начальном уровне).

Подготовка тестовых заданий

После подготовки программы и исправления синтаксических ошибок (что после небольшого периода практических занятий выполняется достаточно быстро) наступает *этап тестирования*. Под *тестовым заданием* (или просто *тестом*) обычно понимается набор исходных данных, при использовании которых в программе должны получиться заранее определенные результаты. Проблеме тестирования посвящено достаточно большое количество работ (см., например, [7, 8]), здесь же отметим ряд основных принципов тестирования:

1. Тесты должны подготавливаться на начальных этапах разработки программ (в идеальном случае, на этапе постановки задачи),

2. Успешность выполнения теста (т.е. когда результаты выполнения программы совпадают с прогнозируемыми) не являются доказательством правильности программы, т.к. тест проверяет только вполне конкретные условия работы программы; полное (исчерпывающее) тестирование обычно нереализуемо из-за практически неограниченного множества различных вариантов исходных данных – с другой стороны, хорошо

подготовленный комплект тестов может проверить основные режимы работы программы и выявить большинство имеющихся ошибок в программе,

3. 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 (хороший тест – это тест, выявляющий наличие ошибки в программе (!)),

4. Тест должен быть достаточно небольшим (по объему исходных данных), быстро выполняемым и желательно должен иметь значения не только результирующих, но и промежуточных данных,

5. Тест желательно должен иметь средства автоматической проверки результатов выполнения (как, например, в учебном примере результаты алгоритма пузырьковой сортировки можно было сравнить с результатами метода стандартной сортировки).

Методы поиска ошибок (отладки)

Итак, признаком наличия ошибки в программе является неправильное выполнение теста (во время теста выполнение программы завершается аварийно или результаты выполнения программы не совпадают с прогнозируемыми результатами теста). Процесс выявления причин обнаруженной ошибки, определение места (локализация) ошибки в программе и исправление ошибочно реализованного программного кода обычно называется *отладкой*. Как правило, при отладке существует некоторая предварительная стадия, во время которой программист выдвигает те или иные предложения о причинах ошибочной работы и проводит визуальный анализ (*инспекцию*) программного кода – к сожалению, данной формой отладки многие программисты пренебрегают, хотя эффективность такого способа отладки является достаточно высокой. Если при инспекции кода выявить ошибки не удается, далее наступает основной способ отладки – *отладочное выполнение программы* (или *трассировка*), 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. Тем самым, задача трассировки – обнаружения информационных признаков проявления ошибки.

Рассмотрим далее возможности среды MS VS .NET для обеспечения трассировки программ при поиске и исправления ошибок.

Пошаговое выполнение программы

Для выполнения программы в пошаговом режиме (в режиме трассировки) используются четыре команды, которые доступны из меню **Debug**, панели инструментов **Debug** и клавиш быстрого вызова:

– 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),

– 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 строки останова и позицией курсора (в зависимости от настроек параметров среды MS VS .NET данная команда может отсутствовать в пункте меню **Debug**).

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 установки контрольной точки. В ходе выполнения программы при попадании на контрольную точку происходит останов; для продолжения работы необходимо выполнить команду **Continue** пункта меню **Debug**.

Вид окна среды разработки для учебного примера в момент останова после

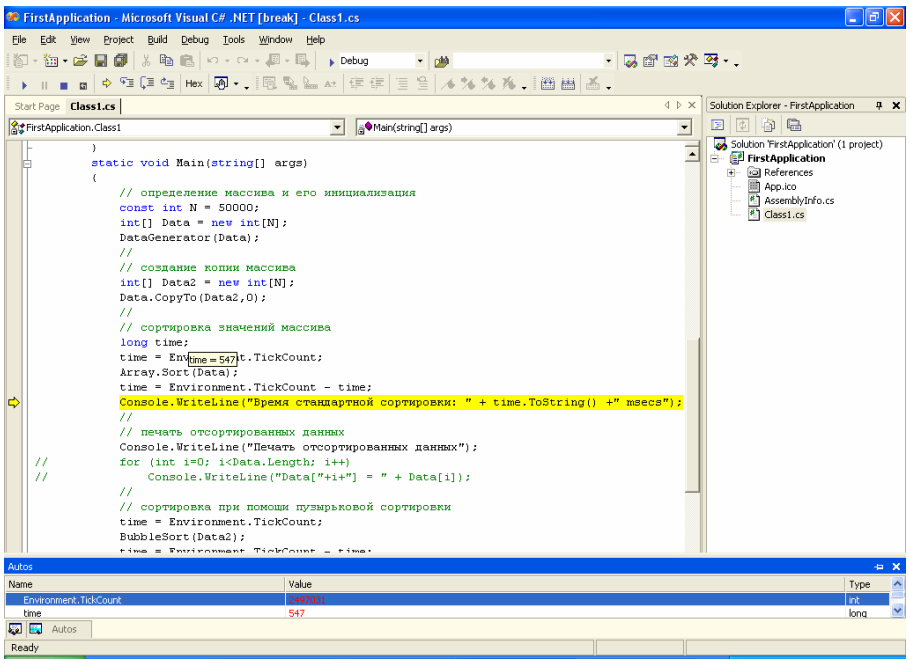

Рис. 1.8. Вид окна среды разработки в момент остановки выполнения программы

выполнения стандартного метода сортировки показан на рис. 1. 8.

Наблюдение значений переменных

Для наблюдения значений переменных в момент останова выполнения программы достаточно расположить указатель мыши на имени переменной – в результате значение переменной появится в виде всплывающей подсказки (пример высветки значения переменной показан на рис. 1.8).

Дополнительная возможность для наблюдения значений переменных состоит в использовании специальных окон наблюдения:

– Окно **Autos** отображает значения всех переменных, используемых в текущей и предшествующей строках точки останова программы; в окне отображаются названия переменных, их тип и значения; окно **Autos** обычно располагается в нижней левой части экрана (см. рис. 1.8) 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);

– Окна **Watch** (таких окон в момент выполнения 4) отличаются тем, что 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** (такая же команда может иметься в пункте меню **Debug**, ее наличие в меню зависит от настроек параметров среды MS VS .NET). Удобный способ добавления переменных в окна наблюдения состоит в использовании техники "Взять и перенести" (выделить имя переменной, нажать левую кнопку мыши и, не отпуская ее, переместить указатель мыши в окно наблюдения, после чего отпустить конку мыши). Для удаления переменных из окна наблюдения достаточно выделить соответствующую строку и нажать клавишу <**Delete**>;

– Близким по назначению к окнам **Watch** является окно **Quick Watch**, которое дополнительно 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, а также окно **Call Stack**, в котором отображается последовательность вызова методов, приведшая к обращению к текущему исполняемому методу.

Пример выполнения отладки

Перечисленных набор средств среды разработки MS VS .NET достаточно для организации быстрой и эффективной отладки. Для успешного освоения этих средств необходима понимание принципов отладки и практика по их использованию для поиска и исправления ошибок при разработке достаточно сложных программ.

Приведем пример проведения процесса отладки с использованием нашей учебной программы. Внесем ошибку в программу – заменим оператор

```
Vals[j+1] = temp;
```

в методе пузырьковой сортировки **BubbleSort** на оператор

```
Vals[j] = temp;
```

(такой прием, конечно, дает достаточно слабое представление о процедуре отладки – больший эффект можно получить, если внесение ошибки произвольного вида (!) будет выполнено кем то другим).

Выполнение программы с внесенной ошибкой приведет к тому, что результирующий массив не будет являться отсортированным – возможная схема отладки может состоять в следующем:

– Можно попытаться проверить, наблюдается ли ошибочный эффект при меньшем размере сортируемого набора данных (меньший объем информации упростит проведение трассировки программы); установим для этого значения константы N=3 и повторим выполнение программы – массив по прежнему остается неотсортированным;

– Попытаемся определить причину ошибки – проблема может состоять или в неправильной работе алгоритма сортировки или ошибочными являются операторы вывода значений массива; однако вывод результатов стандартной сортировки сработал правильно и, кроме того, автоматическая проверка результатов сортировки тоже подтвердила, что результат пузырьковой сортировки является неправильным; как результат, можно сделать вывод, что ошибки содержатся в методе пузырьковой сортировки **BubbleSort**;

– Установим контрольную точку на внутреннем операторе цикла в методе **BubbleSort** (см. рис. 1.9) и запустим программу на выполнение; после останова добавим массив **Vals** в окно наблюдения **Watch 1** и запомним исходное значение сортируемого массива;

– 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 и, как результат, можно сделать вывод, что внешняя итерация алгоритма сортировки работает неправильно;

– 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 значений из рис. 1.9); 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 перестанавливаются) – 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**; для исправления ошибки вносим необходимые изменения, выполняем программу и достигаем требуемого результата (!!!) – программа начинает работать правильно.

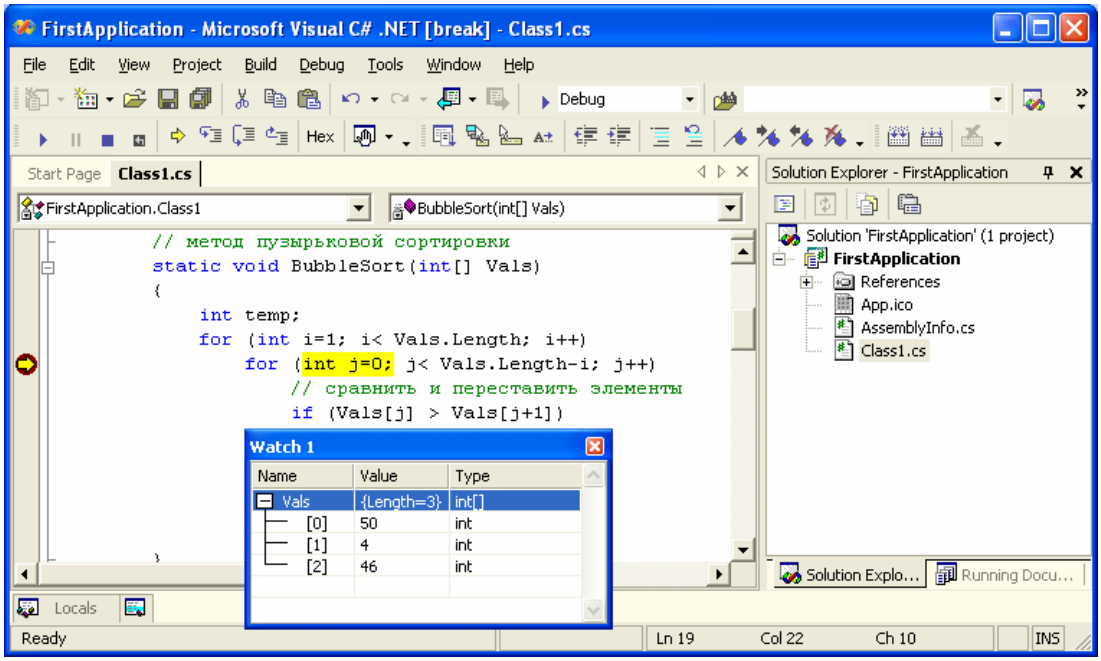

Рис. 1.9. Состояние упорядочиваемого массива перед началом сортировки

Рекомендации по дальнейшему освоению MS VS .NET

Итак, начальное знакомство со средой разработки Microsoft Visual Studio .NET выполнено. Надеемся, что с использованием данного руководства под руководством опытного наставника начало работы в MS VS .NET окажется не только успешным, но и интересным. Несмотря на скромную задачу начального ознакомления со средой, рассмотренных средств вполне достаточно для эффективной разработки сложных программ, работающих в режиме консоли. Безусловно, можно быть довольным достигнутым результатом, но надо и четко понимать, что самое главное впереди. Мир среды разработки MS VS .NET многогранен и является достаточным как для начинающих программистов, так и для профессиональных разработчиков сложного ПО. За рамками начального знакомства осталось многое – прежде всего создание Windows приложений с полноценным графическим интерфейсом, разработка программ для сети Интернет, реализация Web-сервисов,… - все перечислить просто невозможно. Важно отметить также, что помимо использования уже имеющихся возможностей, среда разработки может быть расширена и дополнена непосредственно самим программистом для учета особенностей создаваемого им ПО – 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 – эти и многие другие способы развития среды разработки могут значительно повысить уровень интеллектуальной поддержки деятельности программиста со стороны MS VS .NET.

Существует достаточно много учебных изданий, которые могут помочь в дальнейшем изучении среды разработки MS VS .NET – так, могут быть рекомендованы работы [1-5]. Вопросы, связанные с возможностями расширения среды, в достаточно полной степени рассмотрены в замечательном руководстве [6]. Освоение MS VS .NET, конечно, потребует определенных усилий, но что может остановить настоящего профессионала в стремлении достичь уровня мастерства.

Литература

1. Гарнаев А. Самоучитель Visual Studio .NET 2003. – СПб.: БХВ-Петербург, 2003.

2. Пономарев В. Программирование на C++/C# в Visual Studio .NET 2003. – СПб.: БХВ-Петербург, 2004.

4. Фалчер С. Программирование на Microsoft Visual Studio .NET. - – М.: Русская редакция, 2003.

5. Постолит А. Visual Studio .NET: разработка приложений баз данных. - СПб.: БХВ-Петербург, 2003.

6. Джонсон Б., Скибо К., Янг М. Основы Microsoft Visual Studio .NET. – М.: Русская редакция, 2003.

7. Тамре Л. Введение в тестирование программного обеспечения. - Издательство: Вильямс, 2003.

8. Дастин Э., Рэшка Д., Пол Д. Автоматизированное тестирование программного обеспечения. – М.: Лори, 2003

9. Иванова Г.С. Технология программирования. - М.: МГТУ, 2002.

10. Орлов С.А. Технологии разработки программного обеспечения. - СПб.: Питер, 2002.

11. Соммервил И. Инженерия программного обеспечения. - М.: Вильямс, 2002