

Лабораторная работа

7. Работа с Windows-приложениями

Варианты заданий

Выполнить задания 1-2 по предложенным вариантам:

	№ Задания			
	1	2 (см. варианты)		
Вариант 1	1.III			
Вариант 2	1.I			
Вариант 3	1.II			
Вариант 4	1.II			
Вариант 5	1.III			
Вариант 6	1.I			
Вариант 7	1.II			
Вариант 8	1.III			
Вариант 9	1.I			
Вариант 10	1.II			

Задание 1.

I “Работники и фирмы”. Есть N фирм и M работников. Работники хотят устроиться на работу в одну из фирм, а фирмы хотят нанять работников. У тех и других есть свои предпочтения при приеме на работу.

- Для работников: опубликован список *бинарных* свойств, заполняемых при приеме на работу в фирму. Часть из этих свойств относится к положительным, часть - к отрицательным. Требуется из списка кандидатов составить список предпочтительных кандидатов, упорядоченный по степени предпочтения. Предложите разумный алгоритм и реализуйте его.

Указание: список свойств следует задать перечислением, представляющим шкалу. Следует определить класс Candidate, среди полей которого будет поле properties, заданное перечислением.

- Для фирм: опубликован список *бинарных* свойств, характеризующих условия работы в фирме. Часть из этих свойств работник относит к положительным, часть - к отрицательным. Из списка фирм работник хочет составить список предпочтительных фирм, упорядоченный по степени предпочтения. Предложите разумный алгоритм и реализуйте его.

Указание: список свойств следует задать перечислением, представляющим шкалу. Следует определить класс Firm, среди полей которого будет поле properties, заданное перечислением.

Необходимо создать Windows-проект, моделирующий решение задачи распределения работников по фирмам.

II "Женихи и невесты". Есть N женихов и M невест. Каждый хочет найти свою пару. У каждого есть свои предпочтения.

- Девушка хочет найти жениха. Кандидатов достаточно много. Известен список *бинарных* свойств, которые девушка хочет знать о своих женихах. Часть из этих свойств девушка относит к положительным, часть - к отрицательным. Девушка хочет из списка кандидатов составить список предпочтительных кандидатов, упорядоченный по степени предпочтения. Предложите ей разумный алгоритм и реализуйте его.

Указание: список свойств следует задать перечислением, представляющим шкалу. Следует определить класс Groom, среди полей которого будет поле properties, заданное перечислением.

- Юноша хочет найти невесту. Кандидатов достаточно много. Известен список *бинарных* свойств, которые молодой человек хочет знать о претендентках. Часть из этих свойств юноша относит к положительным, часть - к отрицательным. Юноше требуется из списка претенденток составить список предпочтительных невест, упорядоченный по степени предпочтения. Предложите разумный алгоритм и реализуйте его.

Указание: список свойств следует задать перечислением, представляющим шкалу. Следует определить класс Bride, среди полей которого будет поле properties, заданное перечислением.

Необходимо создать Windows-проект, моделирующий решение задачи создания пар. Эту задачу можно рассматривать как вариацию известной задачи "об устойчивом бракосочетании".

III "Абитуриенты и вузы". Есть N университетов и M школьников. Школьники хотят пойти учиться в один из вузов, а вузы хотят набрать хороших студентов. У тех и других есть свои предпочтения.

- Вуз хочет принять n новых студентов. Желающих поступить в вуз достаточно много, заведомо больше, чем N. Опубликован список *бинарных* свойств, заполняемых при поступлении в вуз. Часть из этих свойств относится к положительным, часть - к отрицательным. Требуется из списка кандидатов составить список предпочтительных кандидатов, упорядоченный по степени предпочтения. Предложите разумный алгоритм и реализуйте его.

Указание: список свойств следует задать перечислением, представляющим шкалу. Следует определить класс Abiturient, среди полей которого будет поле properties, заданное перечислением.

- Школьник хочет поступить учиться в один из n вузов. Опубликован список *бинарных* свойств, характеризующих условия учебы в вузе. Часть из этих свойств школьник относит к положительным, часть - к отрицательным. Из списка вузов школьник хочет составить список предпочтительных вузов, упорядоченный по степени предпочтения. Предложите разумный алгоритм и реализуйте его. **Указание:** список свойств следует задать перечислением, представляющим шкалу. Следует определить класс `University`, среди полей которого будет поле `properties`, заданное перечислением.

Необходимо создать Windows-проект, моделирующий решение задачи распределения абитуриентов по вузам.

Задание 2.

Разработать динамическую библиотеку, содержащую класс вычисления функций с помощью разложения в ряд Тейлора на интервале от $x1$ до $x2$ с шагом dx с точностью eps .

Разработать Windows-приложение для расчета требуемых значений. Пользователь вводит в поля на форме значения $x1$, $x2$, dx , eps . Результат сравнить с выполнением аналогичной функции класса **Math**.

По нажатии кнопки «Рассчитать» формируется таблица значений. Таблицу снабдить заголовком и шапкой. Каждая строка таблицы должна содержать значение аргумента, значение функции, значение функции класса **Math**, разница двух значений и количество просуммированных членов ряда. Оценить время работы программы.

Варианты задания:

$$1. \ln \frac{x+1}{x-1} = 2 \sum_{n=0}^{\infty} \frac{1}{(2n+1)x^{2n+1}} = 2\left(\frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \dots\right) \quad |x| > 1$$

$$3. e^{-x} = \sum_{n=0}^{\infty} \frac{(-1)^n x^n}{n!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \dots \quad |x| < \infty$$

$$4. \ln(x+1) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{n+1}}{n+1} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} - \dots \quad -1 < x \leq 1$$

$$5. \ln \frac{1+x}{1-x} = 2 \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1} = 2\left(x + \frac{x^3}{3} + \frac{x^5}{5} + \dots\right) \quad |x| < 1$$

$$6. \ln(1-x) = - \sum_{n=1}^{\infty} \frac{x^n}{n} = -\left(x + \frac{x^2}{2} + \frac{x^3}{3} + \dots\right) \quad -1 \leq x < 1$$

$$7. \operatorname{arctg} x = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1} x^{2n+1}}{2n+1} = \frac{\pi}{2} - x + \frac{x^3}{3} - \frac{x^5}{5} - \dots \quad |x| \leq 1$$

$$8. \operatorname{arctg} x = \frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{(2n+1)x^{2n+1}} = \frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} \dots \quad x > 1$$

$$9. \operatorname{arctg} x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{(2n+1)} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots \quad |x| \leq 1$$

10. $\operatorname{Arth} x = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{2n+1} = x + \frac{x^3}{3} + \frac{x^5}{5} + \frac{x^7}{7} + \dots \quad |x| < 1$
11. $\operatorname{Arth} x = \sum_{n=0}^{\infty} \frac{1}{(2n+1)x^{2n+1}} = \frac{1}{x} + \frac{1}{3x^3} + \frac{1}{5x^5} + \dots \quad |x| > 1$
12. $\operatorname{arctg} x = -\frac{\pi}{2} + \sum_{n=0}^{\infty} \frac{(-1)^{n+1}}{(2n+1)x^{2n+1}} = -\frac{\pi}{2} - \frac{1}{x} + \frac{1}{3x^3} - \frac{1}{5x^5} + \dots \quad x < -1$
13. $e^{-x^2} = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{n!} = 1 - x^2 + \frac{x^4}{2!} - \frac{x^6}{3!} + \frac{x^8}{4!} - \dots \quad |x| < \infty$
14. $\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots \quad |x| < \infty$
15. $\frac{\sin x}{x} = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n+1)!} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \frac{x^6}{7!} - \dots \quad |x| < \infty$
16. $\ln x = 2 \sum_{n=0}^{\infty} \frac{(x-1)^{2n+1}}{(2n+1)(x+1)^{2n+1}} = 2 \left(\frac{x-1}{x+1} + \frac{(x-1)^3}{3(x+1)^3} + \frac{(x-1)^5}{5(x+1)^5} + \dots \right) \quad x > 0$
17. $\ln x = \sum_{n=0}^{\infty} \frac{(-1)^n (x-1)^{n+1}}{(n+1)} = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} + \dots \quad 0 < x \leq 2$
18. $\ln x = \sum_{n=0}^{\infty} \frac{(x-1)^{n+1}}{(n+1)(x+1)^{n+1}} = \frac{x-1}{x} + \frac{(x-1)^2}{2x^2} + \frac{(x-1)^3}{3x^3} + \dots \quad x > \frac{1}{2}$
19. $\arcsin x = x + \sum_{n=1}^{\infty} \frac{1 \cdot 3 \cdot \dots \cdot (2n-1) \cdot x^{2n+1}}{2 \cdot 4 \cdot \dots \cdot 2n \cdot (2n+1)} = x + \frac{x^3}{2 \cdot 3} + \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} + \frac{1 \cdot 3 \cdot 5 \cdot x^7}{2 \cdot 4 \cdot 6 \cdot 7} + \frac{1 \cdot 3 \cdot 5 \cdot 7 \cdot x^9}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9} \dots \quad |x| < 1$
20. $\arccos x = \frac{\pi}{2} - \left(x + \sum_{n=1}^{\infty} \frac{1 \cdot 3 \cdot \dots \cdot (2n-1) \cdot x^{2n+1}}{2 \cdot 4 \cdot \dots \cdot 2n \cdot (2n+1)} \right) =$
 $= \frac{\pi}{2} - \left(x + \frac{x^3}{2 \cdot 3} + \frac{1 \cdot 3 \cdot x^5}{2 \cdot 4 \cdot 5} + \frac{1 \cdot 3 \cdot 5 \cdot x^7}{2 \cdot 4 \cdot 6 \cdot 7} + \frac{1 \cdot 3 \cdot 5 \cdot 7 \cdot x^9}{2 \cdot 4 \cdot 6 \cdot 8 \cdot 9} \dots \right) \quad |x| < 1$

Методические указания

Оглавление

7. Работа с Windows-приложениями	1
Варианты заданий	1
Методические указания	4
7.1. Шкалы	5
Пример работы со свойствами объектов класса Person	7
Пример. Класс Job	10
7.2. Windows-приложения	15
7.2.1. Постановка задачи	15
7.2.2. Создание DLL - проекта типа "Class Library"	15
Вычисление конечных и бесконечных сумм	18
7.2.3. Windows- проект	21

7.2.4.	Связывание с DLL	22
7.2.5.	Построение интерфейса формы	23
7.2.6.	Как оценить время работы метода	26
7.2.7.	Предварительные сведения о делегатах - функциональном типе данных	26
7.2.8.	Класс TimeValue	27
7.3.	Ввод-вывод массивов	28
7.3.1.	Ввод-вывод массивов в Windows-приложениях	29
7.3.2.	Организация ввода-вывода двумерных массивов	32
7.3.3.	Элемент управления DataGridView и отображение массивов	34
	Справка по объекту DataGridView.	40
	Объект для отображения табличной информации DataGridView	40
	Настройка свойств столбцов в DataGridView	41

7.1. Шкалы

Предположим, что проектируется некоторый содержательный класс, задающий описание множества объектов, например, класс Car, описывающий автомобили, или класс Employee, представляющий описание служащих некоторой фирмы. Для объектов этого класса зададим набор из n **бинарных свойств**. Бинарность свойства означает, что свойство может принимать только два значения. Служащий может владеть или не владеть иностранными языками, автомобиль может быть легковым или не быть таковым. В таких ситуациях набор таких свойств удобно представить перечислением, заданным в виде шкалы, а в соответствующий содержательный класс следует включить поле, тип которого задан этим перечислением.

Перечисление, содержащее n элементов, будем называть **шкалой**, если отображение задано для каждого элемента перечисления и элемент с индексом k отображается в число 2^k .

Каждый объект перечисления, заданного шкалой, представляется целым числом в диапазоне $[0, 2^n - 1]$. Это число следует рассматривать как число *в двоичной системе счисления* – набор из n битов (разрядов), каждый из которых описывает соответствующее свойство объекта. Единица в разряде указывает, что объект обладает данным свойством; ноль означает отсутствие свойства.

Для перечислений, задающих шкалу, разработана специальная реализация метода ToString, выполняющая разбор численного значения и печатающая в качестве результата метода все идентификаторы, задающие свойства объекта перечисления. Для включения этой реализации метода ToString класс перечисления должен быть задан с атрибутом класса [Flags] (флажки). Этот атрибут сообщает компилятору, что перечисление является шкалой и для него необходимо использовать специальную версию метода ToString.

Для объектов рассматриваемого нами класса Person можно ввести в рассмотрение набор из трех свойств – доброта, ум, богатство. Определим соответствующее перечисление как шкалу (перечисление задается вне класса):

```
[Flags]
public enum Dream_Properties
{
    умный = 1,    //0001
    добрый = 2,   //0010
    богатый = 4   //0100
}
```

Класс Person зададим в следующем виде:

```
public class Person
{
    public Dream_Properties Properties;

    string fam = "";

    public string Fam
    {
        set { if (fam == "") fam = value; }
        get { return (fam); }
    }

    public Person() { }
    public Person(string fam)
    {
        this.fam = fam;
    }
}
```

Добавим в класс Person соответствующее поле:

```
public Dream_Properties properties;
```

Шкалы позволяют просто и эффективно реализовать запросы, позволяющие отобрать среди множества объектов те, которые обладают нужным набором свойств. Достигается это за счет того, что над объектами перечисления, заданного шкалой, определены логические операции, выполняемые над соответствующими парами битов, так что одна операция выполняется над всеми свойствами, входящими в шкалу.

Пусть для определенности у нас есть некоторый объект `pattern`, принадлежащий перечислению `Dream_Properties` и представляющий образец поиска, и объект `person` класса `Person`, одно из полей которого является объектом данного перечисления. Рассмотрим различные запросы на соответствие объекта `person` заданному образцу и реализацию запросов с использованием логических операций.

Запрос на поиск элемента, соответствующего образцу

Как проверить, что объект `person` обладает всеми свойствами, указанными в образце?

```
(person.Properties & pattern) == pattern
```

Значение `true` этого выражения говорит о том, что объект удовлетворяет запросу и обладает всеми нужными свойствами.

Запрос на поиск элемента, у которого нет свойств, заданных в образце

Как проверить, что объект person не обладает ни одним из свойств, указанных в образце?

```
(~person.Properties & pattern) == pattern
```

Запрос на поиск элемента, обладающего некоторыми свойствами образца

Как проверить, что объект person обладает некоторыми свойствами, указанными в образце?

```
(person.Properties & pattern) > 0
```

Запрос на поиск элемента, обладающего некоторыми, но не всеми свойствами образца

Как проверить, что объект person обладает некоторыми свойствами, но не всеми свойствами, указанными в образце?

```
((person.Properties & pattern) > 0) &&  
((person.Properties & pattern) < pattern)
```

Запрос на поиск элемента, в точности совпадающего с образцом

Как проверить, что объект person обладает всеми свойствами, указанными в образце, и никакими другими свойствами не обладает?

```
((person.Properties & pattern) == pattern) &&  
((person.Properties & ~pattern) == 0)
```

Пример работы со свойствами объектов класса Person

У нас уже построен класс Person, одно из полей которого является объектом перечисления Dream_Properties. Давайте теперь посмотрим, как клиенты класса Person создают объекты этого класса, задают их свойства и осуществляют поиск персоны, обладающей нужными свойствами. Следующие методы будем создавать в классе Program

Начнем с простого примера, где создается один объект

```
/// <summary>  
/// Один объект и много запросов  
/// </summary>  
static void TestQueries()  
{  
    Person person;  
    Dream_Properties pattern;  
    person = new Person("Петров");  
    person.Properties = Dream_Properties.добрый |  
        Dream_Properties.умный;  
    string str = person.Properties.ToString();  
    Console.WriteLine("Свойства персоны: " + str);  
  
    pattern = Dream_Properties.богатый |  
        Dream_Properties.добрый;  
    str = pattern.ToString();  
    Console.WriteLine("Свойства образца: " + str);  
  
    Dream_Properties temp = person.Properties & pattern;  
    bool query1, query2, query3, query4, query5;  
    // все свойства образца
```

```

    query1 = temp == pattern;

    // ни одно из свойств образца
    query2 = (~person.Properties & pattern) == pattern;

    // некоторые свойства образца
    query3 = temp > 0;

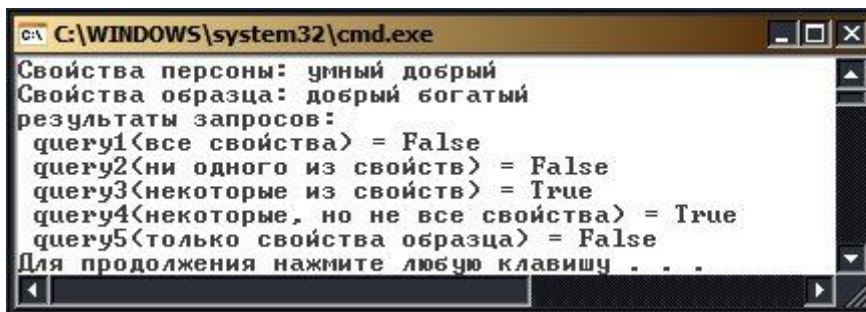
    // некоторые, но не все свойства образца
    query4 = temp > 0 && temp < pattern;

    // только свойства образца
    query5 = (temp == pattern) &&
        ( (person.Properties & ~pattern) == 0 );

    Console.WriteLine("результаты запросов: ");
    Console.WriteLine(" query1(все свойства) = {0}", query1);
    Console.WriteLine(" query2(ни одного из свойств) = {0}",
        query2);
    Console.WriteLine(" query3(некоторые из свойств) = {0}",
        query3);
    Console.WriteLine(" query4(некоторые, но не все свойства) = {0}",
        query4);
    Console.WriteLine(" query5(только свойства образца) = {0}",
        query5);
}

```

В этом методе создается объект класса `Person` и образец с заданным набором свойств. Затем к объекту применяются все пять вышеописанных запросов. Результаты работы метода:



Продолжим наш пример. В предыдущем методе к одному объекту применялись различные запросы. Чаще всего существует обратная ситуация: один запрос применяется к некоторому множеству объектов, среди которых и разыскивается объект с нужными свойствами. Рассмотрим такую ситуацию и начнем с метода, моделирующего создание массива объектов `Person`, каждый из которых обладает своим набором свойств.

```

/// <summary>
/// Создание массива Person из 5 элементов,
/// обладающих набором свойств Dream_Properties
/// </summary>
/// <returns>массив объектов Person</returns>
static Person[] CreatePersonsWithProperties()
{
    Person[] persons = new Person[5];
    persons[0] = new Person("Петров");
}

```



```

    persons[0].Properties = Dream_Properties.умный |
Dream_Properties.добрый;
    persons[1] = new Person("Фролов");
    persons[1].Properties = Dream_Properties.умный |
Dream_Properties.богатый;
    persons[2] = new Person("Климов");
    persons[2].Properties = Dream_Properties.богатый |
Dream_Properties.добрый;
    persons[3] = new Person("Карпов");
    persons[3].Properties = Dream_Properties.умный;
    persons[4] = new Person("Иванов");
    persons[4].Properties = Dream_Properties.добрый;
    return persons;
}

```

Следующий метод показывает, как один запрос применяется к массиву объектов Person. В качестве результата возвращается первый найденный объект в массиве, удовлетворяющий запросу. Если же таких объектов нет, то возвращается значение null.

```

/// <summary>
/// Поиск в массиве persons
/// персоны со свойствами, заданными образцом pattern
/// </summary>
/// <param name="persons"></param>
/// <param name="pattern"></param>
/// <returns></returns>
static Person FindOnePerson(Person[] persons, Dream_Properties
pattern)
{
    foreach (Person person in persons)
        if ((person.Properties & pattern) == pattern)
            return person;
    return null;
}

```

Для полноты картины приведем еще тестовый метод, запускающий на выполнение два приведенных выше метода.

```

static void TestScale()
{
    const string NOT_EXIST =
        " К сожалению, умные, добрые и " +
        "одновременно богатые встречаются крайне редко!";
    Person[] persons = CreatePersonsWithProperties();
    Person person;
    Dream_Properties pattern = Dream_Properties.богатый |
        Dream_Properties.добрый | Dream_Properties.умный;
    string strPattern = pattern.ToString();
    Console.WriteLine("pattern = " + strPattern);
    person = FindOnePerson(persons, pattern);
    if (person == null)
        Console.WriteLine(NOT_EXIST);
    else
        Console.WriteLine(person.Fam + " - " +
            person.Properties.ToString());
    pattern = Dream_Properties.добрый | Dream_Properties.умный;
    strPattern = pattern.ToString();
}

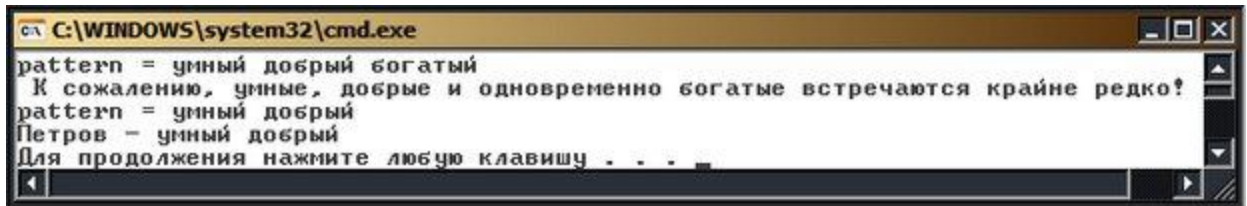
```

```

        Console.WriteLine("pattern = " + strPattern);
        person = FindOnePerson(persons, pattern);
        if (person == null)
            Console.WriteLine(NOT_EXIST);
        else
            Console.WriteLine(person.Fam + " - " +
                               person.Properties.ToString());
    }

```

В этом тестовом методе к одному и тому же массиву персон применяется один и тот же запрос, но с двумя разными образцами поиска. Результаты поиска:



Пример. Класс Job

Пример содержательный, он описывает ситуацию, когда некоторая фирма нанимает на работу программистов, предварительно опубликовав список свойств, рассматриваемых при приеме на работу. Набор этих свойств определяется перечислением, заданным шкалой.

```

/// <summary>
/// Свойства претендентов на должность программиста,
/// описывающие знание технологий и языков программирования
/// </summary>
[Flags]
public enum Prog_Properties
{
    VB = 1,           //00001
    C_sharp = 2,      //00010
    C_plus_plus = 4,  //00100
    Web = 8,          //01000
    Prog_1C = 16       //10000
}

```

Приведу теперь полный текст класса Job, описывающего процесс приема программистов на работу в фирму на основе анализа их свойств и в соответствии с требованиями фирмы. Конечно, пример модельный, и наибольшие сложности связаны с вероятностным моделированием множества кандидатов и их свойств.

```

/// <summary>
/// Прием программистов на работу
/// </summary>
public class Job
{
    //fields
    /// <summary>
    /// Число претендентов

```

```
/// </summary>
int n;

/// <summary>
/// массивы, задающие свойства претендентов
/// </summary>
Prog_Properties[] cand;
string[] strCand;
Prog_Properties pattern;

Prog_Properties currentScale;
Random rnd;

//Constructors
public Job()
{
    n = 10;
    cand = new Prog_Properties[n];
    strCand = new string[n];
    rnd = new Random();
}
public Job(int n)
{
    this.n = n;
    cand = new Prog_Properties[n];
    strCand = new string[n];
    rnd = new Random();
}
public Job(Prog_Properties[] pp)
{
    n = pp.Length;
    cand = pp;
    strCand = new string[n];
    rnd = new Random();
}

//Properties
public Prog_Properties Pattern
{
    set { pattern = value; }
}

//Methods
/// <summary>
/// формирование свойств кандидатов
/// Каждое свойство появляется с вероятностью 0.5
/// </summary>
public void FormCands()
{
    int properties = 5;
    int p = 0, q = 0, currentProps = 0;
    string strQ;
    for (int i = 0; i < n; i++)
    {
        currentProps = 0; strQ = "";
        for (int j = 0; j < properties; j++)
```

```
{
    p = rnd.Next(2);
    q = (int)Math.Pow(2, j);
    if (p == 1)
    {
        currentProps += q;
        strQ += (Prog_Properties)q + ", ";
    }
}
cand[i] = (Prog_Properties)currentProps;
if (strQ != "")
    strCand[i] = strQ.Remove(strQ.Length - 2);
else strCand[i] = "";
}
} //FormCands
public string[] GetStrCands()
{
    return strCand;
}
public Prog_Properties[] GetCands()
{
    return cand;
}
/// <summary>
/// Список кандидатов, которые обладают
/// свойствами, заданных образцом.
/// </summary>
public ArrayList CandsHavePat()
{
    ArrayList temp = new ArrayList();
    for (int i = 0; i < n; i++)
        if ((cand[i] & pattern) == pattern)
            temp.Add("cand[" + i + "]");
    return temp;
}

/// <summary>
/// Список кандидатов, которые не обладают
/// всеми свойствами, заданных образцом.
/// </summary>
public ArrayList CandsHaveNotAllPat()
{
    ArrayList temp = new ArrayList();
    for (int i = 0; i < n; i++)
        if ((~cand[i] & pattern) == pattern)
            temp.Add("cand[" + i + "]");
    return temp;
}

/// <summary>
/// Список кандидатов, которые обладают
/// некоторыми свойствами, заданных образцом.
/// </summary>
public ArrayList CandsHaveSomePat()
{
    ArrayList temp = new ArrayList();
    for (int i = 0; i < n; i++)
    {
```

```

        currentScale = cand[i] & pattern;
        if (currentScale > 0 && currentScale < pattern)
            temp.Add("cand[" + i + "]");
    }
    return temp;
}
/// <summary>
/// Список кандидатов, которые обладают
/// только свойствами, заданных образцом.
/// </summary>
public ArrayList CandsHaveOnlyPat()
{
    ArrayList temp = new ArrayList();
    for (int i = 0; i < n; i++)
        if (((cand[i] & pattern) == pattern) &&
            ((cand[i] & ~pattern) == 0))
            temp.Add("cand[" + i + "]");
    return temp;
}
}

```

Анализируя код этого класса, следует обратить внимание на метод `FormCands`, моделирующий создание множества кандидатов, представленного объектом динамического класса `ArrayList`. Группа методов, реализующих запросы на поиск нужных кандидатов среди множества кандидатов, интересна тем, что результаты поиска также задаются объектом класса `ArrayList`.

В заключение приведу теперь текст метода, тестирующего работу с классом `Job`:

```

/// <summary>
/// Тестирование процесса приема на работу
/// </summary>
static void TestJob()
{
    Prog_Properties pattern = Prog_Properties.C_sharp |
        Prog_Properties.Web;
    Console.WriteLine("Требования, заданные образцом:" +
        " Знание языка C# и Web технологии");

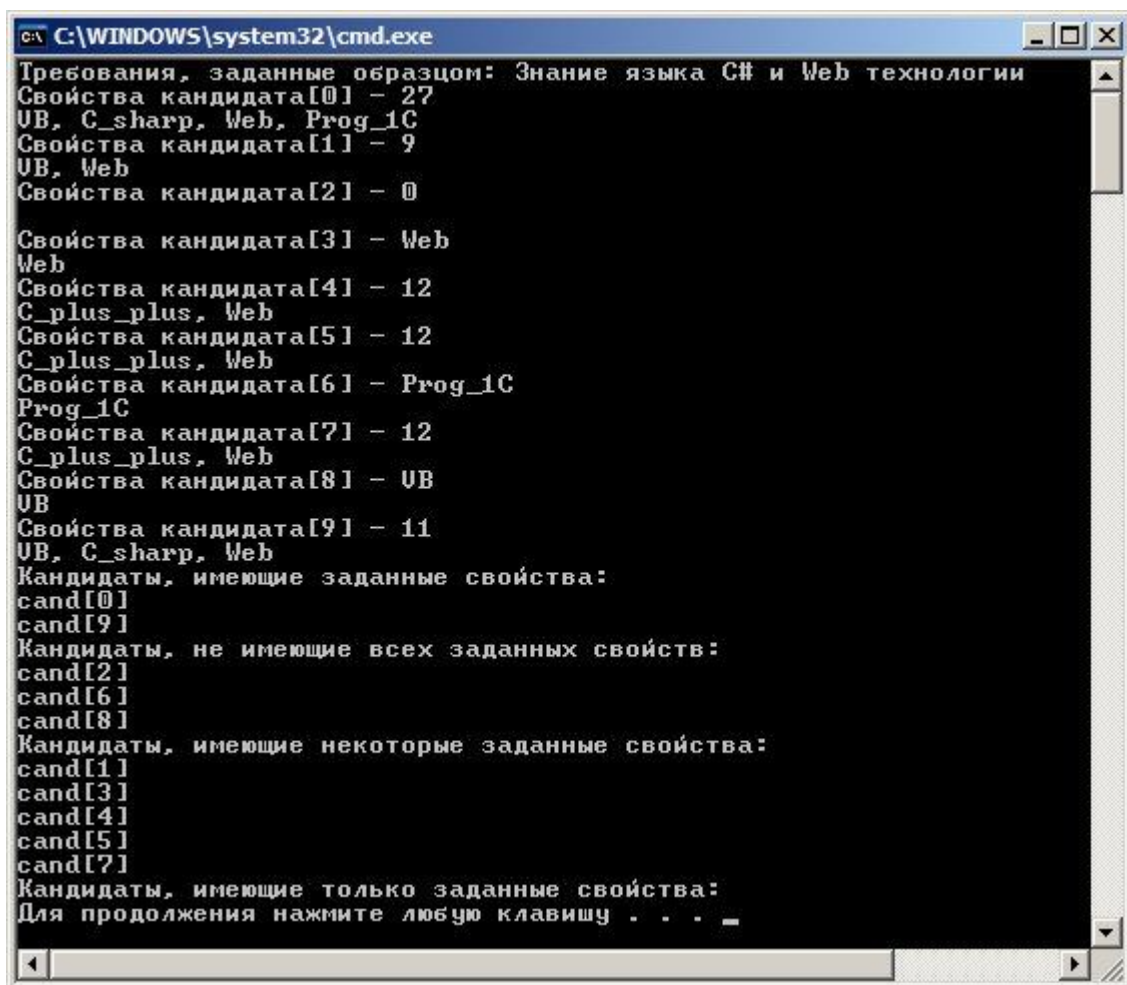
    int n = 10;
    Job mys = new Job(n);
    mys.FormCands();
    Prog_Properties[] cand = mys.GetCands();
    string[] strCand = mys.GetStrCands();
    for (int i = 0; i < n; i++)
    {
        Console.WriteLine("Свойства кандидата[{0}] - {1}",
            i, cand[i]);
        Console.WriteLine(strCand[i]);
    }

    mys.Pattern = pattern;
    ArrayList result;
}

```

```
result = mys.CandsHavePat();  
Console.WriteLine("Кандидаты, имеющие заданные свойства");  
foreach (string pretender in result)  
    Console.WriteLine(pretender);  
  
result = mys.CandsHaveNotAllPat();  
Console.WriteLine("Кандидаты, не имеющие всех свойств");  
foreach (string pretender in result)  
    Console.WriteLine(pretender);  
  
result = mys.CandsHaveSomePat();  
Console.WriteLine("Кандидаты, имеющие некоторые свойства");  
foreach (string pretender in result)  
    Console.WriteLine(pretender);  
  
result = mys.CandsHaveOnlyPat();  
Console.WriteLine("Кандидаты: только заданные свойства");  
foreach (string pretender in result)  
    Console.WriteLine(pretender);  
}
```

Результаты работа теста:



```
C:\WINDOWS\system32\cmd.exe  
Требования, заданные образом: Знание языка C# и Web технологии  
Свойства кандидата[0] - 27  
VB, C_sharp, Web, Prog_1C  
Свойства кандидата[1] - 9  
VB, Web  
Свойства кандидата[2] - 0  
  
Свойства кандидата[3] - Web  
Web  
Свойства кандидата[4] - 12  
C_plus_plus, Web  
Свойства кандидата[5] - 12  
C_plus_plus, Web  
Свойства кандидата[6] - Prog_1C  
Prog_1C  
Свойства кандидата[7] - 12  
C_plus_plus, Web  
Свойства кандидата[8] - VB  
VB  
Свойства кандидата[9] - 11  
VB, C_sharp, Web  
Кандидаты, имеющие заданные свойства:  
cand[0]  
cand[9]  
Кандидаты, не имеющие всех заданных свойств:  
cand[2]  
cand[6]  
cand[8]  
Кандидаты, имеющие некоторые заданные свойства:  
cand[1]  
cand[3]  
cand[4]  
cand[5]  
cand[7]  
Кандидаты, имеющие только заданные свойства:  
Для продолжения нажмите любую клавишу . . .
```

7.2. Windows-приложения

7.2.1. Постановка задачи

Системный класс Math является примером статического класса, играющего единственную роль – роль модуля. У этого класса нет собственных данных, если не считать двух математических констант - e и π , а его методы являются сервисами, которые он предоставляет другим классам.

Построим аналог класса Math и поместим этот класс в динамическую библиотеку DLL, что позволит повторно использовать его, присоединяя при необходимости к различным проектам. В нашем примере не будем моделировать все сервисы класса Math. Ограничимся рассмотрением вычисления функции $\sin(x)$. Эту функцию, как и другие математические функции, можно вычислить, используя разложение в ряд Тэйлора:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \dots = \sum_{i=0}^{\infty} (-1)^i \frac{x^{2i+1}}{(2i+1)!} \quad (1.1)$$

Детали вычислений, использующих формулу 1.1, отложим на момент реализации. А пока продолжим уточнять цель нашего примера. Итак, мы хотим построить DLL, содержащей класс, являющийся аналогом класса Math из библиотеки FCL. Затем мы построим Windows-проект, интерфейс которого позволит провести некоторые исследования. Оба проекта будут находиться в одном Решении.

7.2.2. Создание DLL - проекта типа "Class Library"

Запустим Visual Studio, со стартовой страницы перейдем к созданию проекта и в качестве типа проекта укажем тип "Class Library". В открывшемся окне создания DLL, показанном на рис. 1, все поля заполнены значениями по умолчанию. Как правило, их следует переопределить, задавая собственную информацию.

В поле Name задается имя строящейся DLL - MathTools в нашем случае.

В поле Location указывается путь к папке, где будет храниться Решение, содержащее проект. Для Решений этого курса создана специальная папка.

В поле Solution выбран элемент "Create New Solution", создающий новое Решение. Альтернативой является элемент списка, указывающий, что проект может быть добавлен к существующему Решению.

В окне Solution Name задано имя Решения. Здесь выбрано имя Ch1, указывающее на то, что все проекты вложены в одно Решение.

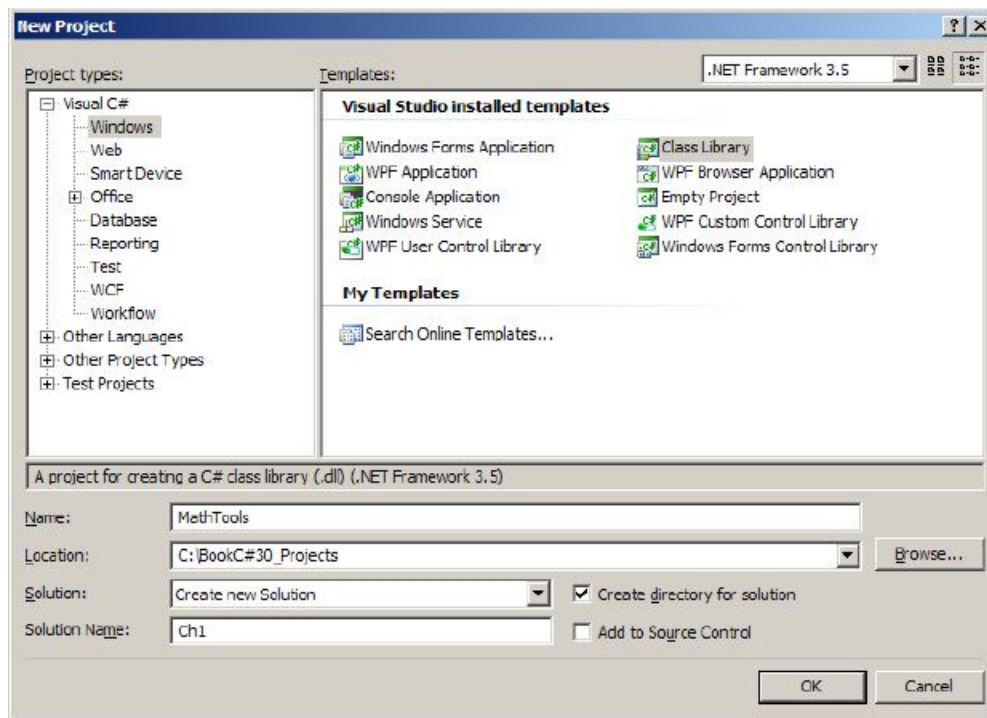


Рис. 1. Создание проекта DLL

Задав требуемые установки и щелкнув по кнопке "OK", получим автоматически построенную заготовку проекта DLL, открытую в среде разработки проектов Visual Studio. На рис. 2 показан внешний вид среды с построенным Решением и проектом.

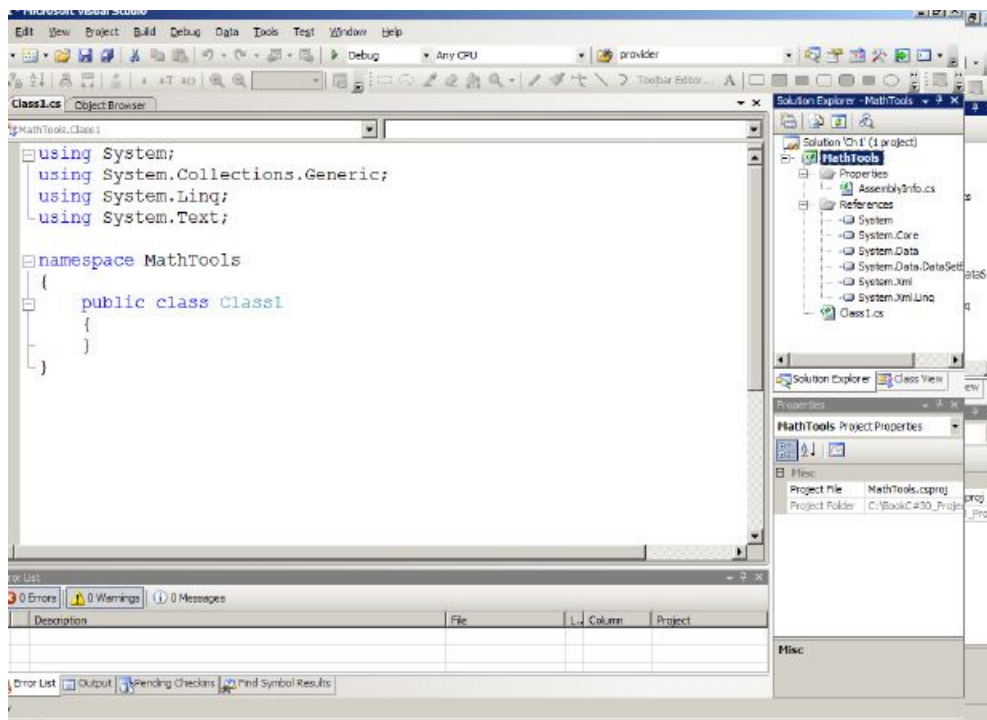


Рис. 2. Среда Visual Studio с начальным проектом DLL

В окне проектов Solution Explorer показано Решение с именем "Ch1", содержащее проект DLL с именем "MathTools". В папке "Properties" проект содержит файл с описанием сборки - ее имя и другие характеристики. В папке "References" лежат ссылки на основные пространства имен библиотеки FCL, которые могут понадобиться в процессе работы DLL.

Поскольку всякая DLL содержит один или несколько классов, то для одного класса, которому по умолчанию дано имя "Class1", заготовка построена. Класс этот, показанный в окне кода, пока что пуст - не содержит никаких элементов.

Построенный автоматически класс вложен в пространство имен, которое по умолчанию получило имя, совпадающее с именем проекта - MathTools. Перед именем пространства заданы четыре предложения using, играющие роль инструкций для компилятора. В этих предложениях указываются имена пространств имен, присоединенных к проекту. Когда в коде создаваемого класса нужно сослаться на класс из пространств, указанных в предложениях using, можно задавать собственное имя этого класса, опуская имя пространства.

Мы рассмотрели подготовительную часть работы, которую Visual Studio выполнила для нас. Изменим имя "Class1" на имя "MyMath". Нельзя вручную изменять имена объектов в проектах. В окне кода проекта выделите имя изменяемого объекта, затем в главном меню выберите пункт Refactor и подпункт Rename. В открывшемся окне укажите новое имя. Тогда будут показаны все места, требующие переименования объекта. В данном случае будет только одна очевидная замена, но в общем случае замен много, так что автоматическая замена всех вхождений крайне полезна.

Следующий шаг также продиктован правилом стиля: имя класса и имя файла, хранящего класс, должны совпадать. Переименование имени файла делается непосредственно в окне проектов Solution Explorer.

Для добавления документируемого комментария достаточно в строке, предшествующей заголовку класса, набрать три подряд идущих слеша (три косых черты). В результате перед заголовком класса появится заголовочный комментарий - тэг "summary", в который и следует добавить краткое, но содержательное описание сути класса. Тэги "summary", которыми следует сопровождать классы, открытые (public) методы и поля класса играют три важные роли. Они облегчают разработку и сопровождение проекта, делая его самодocuментируемым. Специальный инструмент позволяет построить документацию по проекту, включающую информацию из тегов "summary". В нашем случае комментарий к классу MyMath может быть достаточно простым - "Аналог класса Math библиотеки FCL".

Поскольку мы хотим создать аналог класса Math, в нашем классе должны быть аналогичные методы. Начнем, как уже говорилось, с метода, позволяющего вычислить функцию $\sin(x)$. Заголовок метода сделаем такой же, как и в классе аналоге. Зададим заголовочный комментарий к методу. Назовем этот метод SinOld, т.к. мы планируем создать метод и затем провести оптимизацию в новом методе. В результате в тело класса добавим следующий код:

```
/// <summary>  
/// Sin(x)
```

```
/// </summary>
/// <param name="x">угол в радианах - аргумент функции Sin</param>
/// <returns>Возвращает значение функции Sin для заданного
угла</returns>
public static double SinOld(double x)
{
}
```

Осталось написать реализацию вычисления функции, заданную формулой 1.1.

Вычисление конечных и бесконечных сумм

Вычисление конечных сумм и произведений - это наиболее часто встречающийся тип элементарных задач. Какова бы не была сложность выражений, стоящих под знаком конечной суммы с заданным числом слагаемых, задачу всегда можно записать в виде:

$$S = \sum_{k=1}^n a_k \quad (1.2)$$

и применить для ее решения следующий шаблон:

```
S=0;
for(int k=1; k<=n; k++)
{
    //Вычислить текущий член суммы ak
    ...
    S+=ak;
}
```

Заметьте, если перед началом цикла не позаботиться о том, чтобы эта переменная была равна нулю, то после завершения цикла корректность результата не гарантируется.

В этой схеме основные проблемы могут быть связаны с вычислением текущего члена суммы a_k . Нужно понимать, что a_k - это не массив, а скаляр - простая переменная. Значения этой переменной вычисляются заново на каждом шаге цикла, задавая очередной член суммирования. Кроме того, следует заботиться об эффективности, применяя два основных правила, позволяющие уменьшить время вычислений.

Чистка цикла. Все вычисления, не зависящие от k , следует вынести из цикла (в раздел Init).

Рекуррентная формула. Часто можно уменьшить время вычислений a_k , используя предыдущее значение a_k , построив рекуррентную формулу $a_{k+1} = f(a_k)$. Этот прием с успехом применяется как при вычислении функции $\sin(x)$ по формуле 1.1, так и при аналогичных вычислениях большинства других математических функций.

Покажем на примере формулы 1.1, как можно построить необходимые рекуррентные соотношения. Запишем соотношения для a_0, a_k, a_{k+1} :

$$a_0 = \frac{(-1)^0 x}{1!} = x; \quad a_k = \frac{(-1)^k x^{2k+1}}{(2k+1)!}; \quad a_{k+1} = \frac{(-1)^{k+1} x^{2k+3}}{(2k+3)!} \quad (1.3)$$

Вычислив отношение a_{k+1}/a_k , получим требуемое рекуррентное соотношение:

$$a_{k+1} = a_k \frac{-x^2}{(2k+2)(2k+3)} \quad (1.4)$$

Значение a_0 задает базис вычислений, позволяя инициализировать начальное значение переменной a_k , а соотношение 1.4 позволяет каждый раз в теле цикла вычислять новое значение этой переменной. Заметьте: введение рекуррентного соотношения позволило избавиться от вычисления факториалов и возведения в степень на каждом шаге цикла.

Иногда следует ввести несколько дополнительных переменных, хранящих вычисленные значения предыдущих членов суммы. Рекуррентная формула выражает новое значение a_k через предыдущее значение и дополнительные переменные, если они требуются. Начальные значения a_k и дополнительных переменных должны быть корректно установлены перед выполнением цикла.

При вычислении на компьютере значения функции, заданной разложением в бесконечный сходящийся ряд, не ставится задача получения абсолютно точного результата. Достаточно вычислить значение функции с заданной точностью ε . На практике вычисления продолжаются до тех пор, пока текущий член суммы не станет по модулю меньше заданного ε . Чтобы этот прием корректно работал, необходима сходимость ряда.

Вернемся к задаче вычисления функции $\sin(x)$. Вот возможный шаблон решения:

```
while (Abs (ak) > EPS)
{
    S+=ak;
    k++;
    //Вычислить новое значение ak
    ...
}
```

При применении этого шаблона предполагается, что в начале объявляются и должным образом инициализируются нужные переменные - S, ak, k. По завершению цикла переменная S содержит значение функции, вычисленное с заданной точностью.

Код

Приведем полный код проекта DLL, построенный на данный момент:

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

namespace MathTools
```

```

{
    /// <summary>
    /// Аналог класса Math библиотеки FCL
    /// </summary>
    public class MyMath
    {
        //Константы класса
        const double TWOPI = 2 * Math.PI;
        const double EPS = 1E-9;

        //Статические методы класса

        /// <summary>
        /// Sin(x)
        /// </summary>
        /// <param name="x">
        ///     угол в радианах - аргумент функции Sin
        /// </param>
        /// <returns>
        ///     Возвращает значение функции Sin для заданного угла
        /// </returns>
        public static double SinOld(double x)
        {
            //Оптимизация - приведение к интервалу
            x = x % TWOPI;

            //Init
            double a = x;
            double res = 0;
            int k = 0;

            //Основные вычисления
            while (Math.Abs(a) > EPS)
            {
                res += a;
                a *= -x * x / ((2 * k + 2) * (2 * k + 3));
                k++;
            }
            return res;
        }
    }
}

```

Поставленная цель достигнута - построена DLL, содержащая класс, метод которого позволяет вычислять по заданному аргументу x функцию $\sin(x)$. Метод построен в полном соответствии с описанным алгоритмом. При его построении использованы две важные оптимизации. Во-первых, применено рекуррентное соотношение, позволяющее существенно ускорить время и точность вычисления функции. Во-вторых, аргумент x приведен к сравнительно небольшому интервалу, что увеличивает скорость сходимости и гарантирует работоспособность метода для больших значений x . Если не делать этой оптимизации, то для больших по модулю значений метод может давать некорректные результаты, - проверьте это предположение.

Оптимизацию можно продолжить, правда, не столь уже существенную. Сейчас для вычисления значения переменной a требуется выполнить 1 деление, 5 умножений, 2

сложения, взятие результата с обратным знаком. Добавим в класс новую версию метода с улучшенными показателями, сохранив для новой версии имя метода - Sin.

Оставим в классе и старый метод с именем SinOld. Две версии, давая один и тот же результат вычислений, позволят нам в дальнейшем провести некоторые полезные исследования. Вот код нового метода с дополнительной оптимизацией:

```
public static double Sin(double x)
{
    //Оптимизация - приведение к интервалу
    x = x % TWOPI;

    //Init
    double a = x;
    double res = 0;
    int k = 0;
    double x2 = x * x;

    //Основные вычисления
    while (Math.Abs(a) > EPS)
    {
        res += a;
        k+=2;
        a *= -x2 / (k * (k + 1));
    }
    return res;
}
```

Код метода стал элегантнее и короче: вместо пяти умножений теперь делается только два, и вместо двух сложений - одно.

Построим Решение, содержащее проект, для чего в Главном меню среды выберем пункт Build|Build Solution. В результате успешной компиляции будет построен файл с уточнением dll.

Поскольку построенная сборка не содержит выполняемого файла, то непосредственно запустить наш проект на выполнение не удастся!

Построим проект, к которому присоединим нашу DLL, и протестируем, насколько корректно работают созданные нами методы.

7.2.3.Windows- проект

Добавим в Решение новый проект. В качестве типа проекта выберем "Windows Forms Application", дадим проекту имя "WindowsFormsToMathTools". Результат этой работы показан на рис. 3.

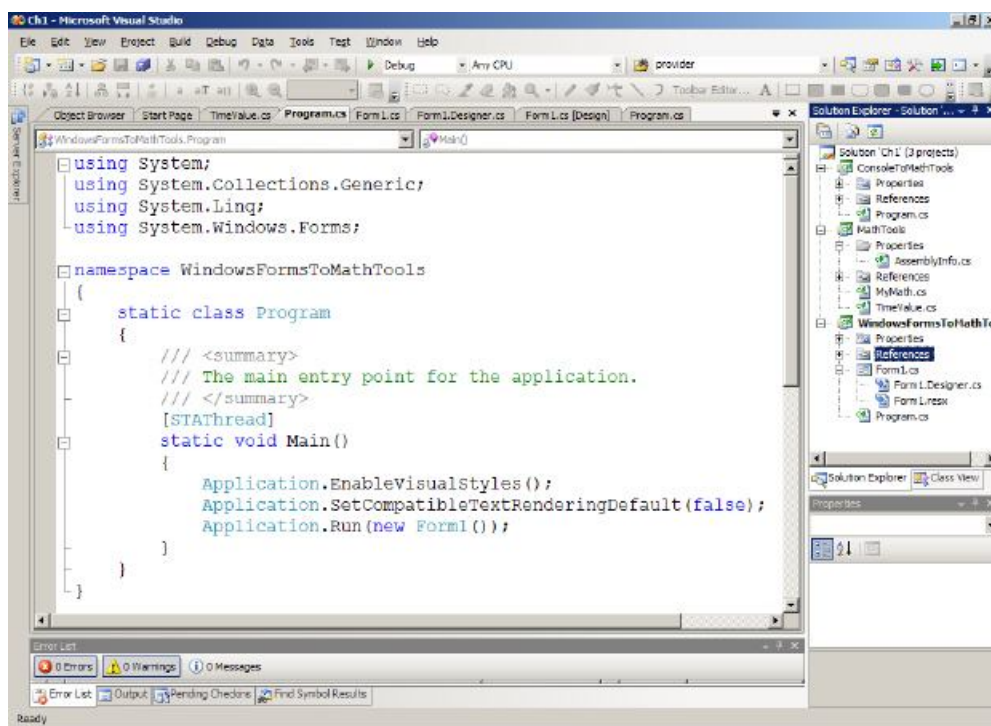


Рис. 3. Решение, содержащее три проекта - Class Library, Console, Windows

7.2.4.Связывание с DLL

Первым делом свяжем два построенных проекта, для чего в новый проект добавим ссылку на проект с DLL MathTools. В окне Solution Explorer подведем указатель мыши к имени проекта и из контекстного меню, появляющегося при щелчке правой кнопки, выберем пункт меню "Add Reference". В открывшемся окне добавления ссылок выберем вкладку "Projects". Поскольку проект MathTools включен в Решение, то он автоматически появится в открывшемся окне. Если ссылку нужно установить на проект, не включенный в Решение, то в окне добавления ссылок нужно задать путь к проекту. Нам проще, путь указывать не нужно, достаточно щелкнуть по появившемуся в окне имени MathTools. Ссылка на DLL появится в папке "References" нового проекта. Теперь проекты связаны и из нового проекта доступны сервисы, предоставляемые DLL.

При создании проекта DLL автоматически создавался в проекте один пустой класс. В Windows-проекте автоматически создаются два класса - класс с именем Form1 и класс с именем Program.

Первый из этих классов является наследником класса Form из библиотеки FCL и наследует все свойства и поведение (методы и события) родительского класса. Класс Form поддерживает организацию интерфейса пользователя в визуальном стиле. Форма является контейнером для размещения визуальных элементов управления - кнопок (Button), текстовых полей (TextBox), списков (ListBox), меток (Label), блоков группировки (GroupBox) и более экзотичных элементов - таблиц (DataGridView), деревьев (TreeView) и многих других элементов.

Классы в C# синтаксически не являются неделимыми и могут состоять из нескольких частей, каждая из которых начинается с ключевого слова "partial" (частичный). Таковым является и построенный автоматически класс Form1. Возможность разбиения описания одного класса на части появилась еще в версии языка C# 2.0, что облегчает работу над большим классом. Каждая часть класса хранится в отдельном файле со своим именем.

Одна часть класса Form1 лежит в файле с именем "Form1.Designer.cs". Эта часть класса заполняется автоматически инструментарием, называемым Дизайнером формы. Другая часть класса Form1, хранящаяся в файле "Form1.cs", предназначена для разработчика - именно в ней располагаются автоматически создаваемые обработчики событий, происходящих с элементами управления, код которых создается самим разработчиком.

Класс Program, автоматически создаваемый в Windows-проекте, содержит точку входа - статический метод Main. В отличие от консольного проекта, где тело процедуры Main изначально было пустым и должно было заполняться разработчиком проекта, в Windows-проектах процедура Main уже готова и, как правило, разработчиком не изменяется. Автоматически созданная процедура Main (текст которой можно видеть на рис. 3) работает с классом Application библиотеки FCL, вызывая поочередно три статических метода этого класса - EnableVisualStyles, SetCompatibleTextRenderingDefault, Run. О назначении первых двух методов можно судить по их содержательным именам. Основную работу выполняет метод Run - в процессе его вызова создается объект класса Form1 и открывается форма - визуальный образ объекта, с которой может работать конечный пользователь проекта. Типичной ситуацией является проведение вычислений по данным, введенным пользователем, и отображение результатов этих вычислений в полях формы, предназначенных для этих целей.

Процедура Main, задающая точку входа в Windows-проект, открывает главную форму, и пользователь попадает в спроектированный для этой формы мир графических объектов. Теперь пользователь становится у руля управления ходом выполнения проекта и является "возмутителем спокойствия" в этом мире объектов. Он начинает вводить тексты в текстовые окна, выбирать нужные ему элементы из открывающихся списков, нажимать командные кнопки и выполнять другие действия над элементами управления. Каждое такое действие пользователя приводит к возникновению соответствующего события у программного объекта. Так, когда пользователь изменяет текст в текстовом окне, у соответствующего объекта класса TextBox возникает событие TextChanged, при нажатии командной кнопки у объекта возникает событие Click, двойной щелчок по командной кнопке приводит к событию DbClick. В ответ на возникновение события объект посылает сообщение операционной системе. Обработывая очередь сообщений, операционная система отыскивает обработчик события, если объект предусмотрел таковой, и передает ему управление.

Подключить к объекту обработчик события можно визуально в процессе проектирования элемента управления. В окне свойств (Properties) элемента управления можно перейти к списку событий этого элемента и в этом списке выбрать (включить) нужное событие. Передача значения в объекте класса TextBox происходит через параметр Text (напр. TextBox1.Text).

7.2.5. Построение интерфейса формы

При создании нового Windows приложения по умолчанию генерируется пустая форма (Form1). Как уже говорилось ранее, основной код приложения (обработки событий) будет

размещен именно в форме (Form1.cs), а не в файле Program.cs. Можно выбрать режим просмотра формы – «дизайнерский» для визуального размещения элементов, или же режим редактирования кода (при нажатии правой кнопкой на форму или на файл Form1.cs в блоке Solution Explorer: View Code или View Designer). Сама форма состоит из трех файлов: основной - Form1.cs, а также – не требующие изменений: Form1.Designer.cs и Form.resx.

Прежде чем заняться построением интерфейса формы, переименуем класс Form1, дав ему имя - FormResearchSinus. Переименование объектов класса хотя и можно делать вручную, но это далеко не лучший способ, к тому же, чреватый ошибками. Для этих целей следует использовать возможности, предоставляемые меню Refactor|Rename. Параллельно с переименованием класса следует переименовать и файл (файлы) с описанием класса.

Таким образом, в нашем случае необходимо перейти в режим кода для Form1 (View Code), в строке:

```
public partial class Form1 : Form
```

выбрать “Form1”, и через пункт меню Refactor|Rename переименовать данный класс в FormResearchSinus. После чего перейти в режим дизайнера (ViewDesigner), выбрать форму, затем в блоке Properties установить в свойстве Text нужное значение. Осталось переименовать и сам файл Form1.cs в блоке Solution Explorer.

Займемся теперь построением интерфейса - размещением в форме элементов управления. В нашем классе реализована пока только одна функция - $\sin(x)$, так что можем построить пока калькулятор одной функции.

На рис. 4 показан интерфейс спроектированной формы.

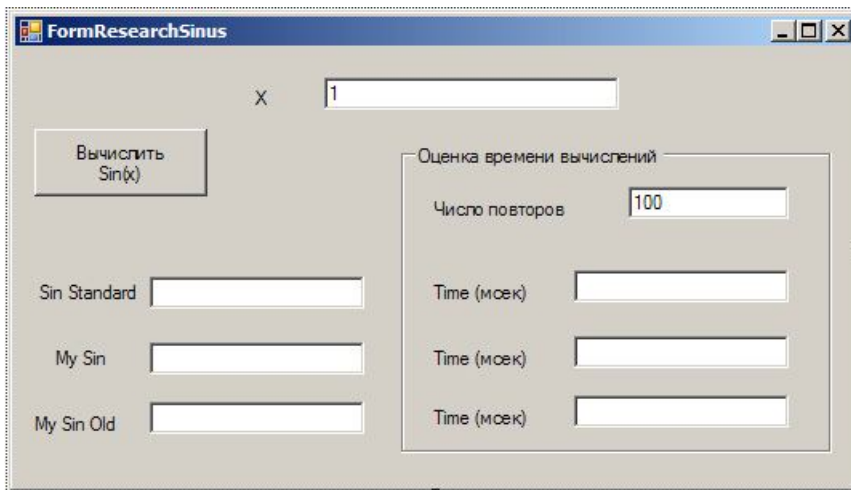



Рис. 4. Интерфейс формы класса FormResearchSinus

В форму включено текстовое поле для ввода значения аргумента x , три текстовых поля предназначены для отображения результата вычислений функции $\sin(x)$ тремя различными методами. В форме есть отдельный контейнер для оценки временных характеристик. В контейнер помещены три текстовых поля, в которых будет отображаться время, затрачиваемое на вычисление функции каждым из анализируемых методов. В

контейнере размещено окно, позволяющее задать число повторов вычисления функции при измерении времени работы.

Размещаются новые элементы через блок меню Toolbox. В нашем проекте используются элементы Button, Label, TextBox, GroupBox. Каждый элемент имеет свое имя, которое используется для доступа к нему в коде и изменению параметров. Для удобства желательно давать каждому элементу свое понятное название. Например, поле ввода переменной x мы можем переименовать из TextBox1 в tboxInputX (в разделе Properties пункт (Name)), а кнопку Button1 в buttonCalculateSin.

Для входных текстовых полей (аргумент функции и число повторов) заданы значения по умолчанию (задаются через свойство Text в разделе Properties). В форме находится командная кнопка, щелчок по которой приводит к возникновению события Click этого объекта, а обработчик этого события запускает вычисление значений функции, получение оценок времени вычисления и вывод результатов в соответствующие текстовые поля.

Другие события для каждого элемента можно посмотреть, нажав кнопку  Events в блоке Properties.

Каков сценарий работы пользователя? Когда при запуске проекта открывается форма, пользователь может в соответствующих полях задать значение аргумента функции и число повторов, после чего нажать кнопку с надписью "Вычислить $\sin(x)$ ". В выходных текстовых полях появятся результаты вычислений. Меняя входные данные, можно наблюдать, как меняются результаты вычислений.

Таким образом, код обработки нажатия кнопки будет выглядеть следующим образом:

```
private void buttonCalculateSin_Click(object sender, EventArgs e)
{
    Double x = Convert.ToDouble(textBox1.Text);
    textBoxSinStandard.Text = Math.Sin(x).ToString();
    textBoxMySin.Text =
        MathTools.MyMath.Sin(x).ToString();
    textBoxMySinOld.Text =
        MathTools.MyMath.SinOld(x).ToString();
}
```

В коде класса формы по умолчанию присутствует функция

```
private void FormEE_Load(object sender, EventArgs e),
```

которая позволяет проводить предварительные вычисления или обработку данных до того как пользователь начнет работу с формой.

Проведем еще одно важное исследование - оценим время, затрачиваемое на вычисление функции, т.е. время, затрачиваемое компьютером на вычисление функции как стандартным методом класса Math, так и методами класса MyMath из библиотеки MathTools.

7.2.6. Как оценить время работы метода

Чем считать операции, зачастую проще непосредственно измерить реальное время вычислений. В библиотеке CLR для этих целей создан класс `DateTime`, позволяющий работать с датами и временами. У этого класса есть статический метод `Now`, вызов которого возвращает в качестве результата объект класса `DateTime`, задающий текущую дату и текущее время (по часам компьютера). Многочисленные свойства этого объекта - `Year`, `Month`, `Hour`, `Second` и многие другие позволяют получить все характеристики даты и текущего времени. Текущее время можно также измерять и в единицах, называемых "тиками", где один тик равен 100 наносекунд или, что то же, 10^{-7} секунды.

Имея в своем арсенале такой класс, не составит большого труда измерить время, требуемое на выполнение некоторого участка кода. Достаточно иметь две переменные с именами, например, `start` и `finish` класса `DateTime`. Переменной `start` присвоим значение, возвращаемое функцией `Now` перед началом измеряемого участка кода, а переменной `finish` - в конце участка кода. Разность времен даст нам требуемую оценку длительности выполнения кода.

Добавим в библиотеку классов DLL `MathTools` новый класс. Вот описание этого класса, пока пустого, но содержащего заголовочный комментарий:

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

namespace MathTools
{
    /// <summary>
    /// Класс спроектирован для получения оценок времени
    /// выполнения различных методов.
    /// Встроенные делегаты определяют сигнатуры методов
    /// </summary>
    public class TimeValue
    {
    }
}
```

Вместо того чтобы в работающей программе окружать вызовы этих методов специальными операторами, напомним специальную процедуру, производящую оценку времени работы, передавая ей в качестве параметра имя исследуемого метода.

7.2.7. Предварительные сведения о делегатах - функциональном типе данных

В языке C# для описания функциональных типов используются классы, называемые делегатами, описание которых начинается с ключевого слова - `delegate`. Описание делегата представляет описание сигнатуры функций, принадлежащих одному функциональному типу. Под сигнатурой функции понимается описание числа, порядка и типов аргументов функции и типа возвращаемого значения. В языках программирования заголовок функции определяет ее сигнатуру. Пусть задан делегат

```
public delegate double DToD(double arg1);
```

Этот делегат задает описание класса с именем DToD (Double To Double), которому принадлежат все функции с одним аргументом типа double и возвращающие результат типа double. Функция $\sin(x)$, как и многие другие математические функции, соответствует этой сигнатуре и, следовательно, является объектом этого класса.

Если задан делегат, то появляется возможность объявлять объекты этого класса, в частности, формальный аргумент метода может принадлежать такому классу, а в качестве фактического аргумента в момент вызова можно передавать имя конкретной функции, принадлежащей данному функциональному типу.

7.2.8. Класс TimeValue

Теперь уже можно привести код класса TimeValue со встроенным делегатом DToD, предоставляющий своим клиентам такой сервис, как оценка времени работы любого метода клиента, сигнатура которого согласована с делегатом. При необходимости этот класс всегда можно расширить, добавив соответствующие сервисы и новые делегаты. Вот этот код:

```
public class TimeValue
{
    public delegate double DToD(double arg1);
    /// <summary>
    /// Возвращает время в секундах,
    /// затраченное на вычисление count раз
    /// метода fun с сигнатурой, которая удовлетворяет
    /// делегату DToD (double to double)
    /// </summary>
    /// <param name="count">число повторений</param>
    /// <param name="fun">имя функции</param>
    /// <param name="x">аргумент</param>
    /// <returns>время в миллисекундах или тиках</returns>
    public static double EvalTimeDToD(int count, DToD fun, double
x)
    {
        DateTime start, finish;
        double res = 0;
        start = DateTime.Now;
        for (int i = 1; i < count; i++)
            fun(x);
        finish = DateTime.Now;
        //res = (finish- start).Ticks;
        res = (finish - start).Milliseconds;
        return res;
    }
}
```

Время можно измерять в разных единицах, например, в тиках или миллисекундах. Статический метод EvalTimeDToD, реализующий сервис класса, устроен достаточно просто. Две переменные start и finish класса DateTime вызывают свойство Now, охватывая цикл по числу повторов вызовов метода, функциональный тип которого задан делегатом, а имя передается в качестве фактического параметра при вызове метода EvalTimeDToD.

Таким образом, в обработке кода нажатия кнопки необходимо добавить:

```
//Оценка времени вычислений
int count = 1;
count = Convert.ToInt32(textBoxCount.Text);

textBoxTimeStandard.Text =
    (MathTools.TimeValue.EvalTimeDToD
     (count, Math.Sin, x)).ToString();

textBoxTimeMySinOld.Text =
    (MathTools.TimeValue.EvalTimeDToD
     (count, MathTools.MyMath.SinOld, x)).ToString();

textBoxTimeMySin.Text =
    (MathTools.TimeValue.EvalTimeDToD
     (count, MathTools.MyMath.Sin, x)).ToString();
```

В заключение приведем результаты вычислений и временных оценок, полученных для нашего примера.

The screenshot shows a Windows application window titled "FormResearchSinus". It contains a text box for "x" with the value "-1000". Below it is a button labeled "Вычислить Sin(x)". To the right, there is a section titled "Оценка времени вычислений" (Timing evaluation) which includes a text box for "Число повторов" (Number of repetitions) set to "100000". Below this are three rows of timing data: "Time (мсек)" values of "16", "31", and "47". On the left side of the window, there are three text boxes showing calculation results: "Sin Standard" with value "-0.826879540532003", "My Sin" with value "-0.826879540419217", and "My Sin Old" with value "-0.826879540419217".

Рис. 5. Сравнительные результаты точности и времени вычисления функции $\sin(x)$

7.3. Ввод-вывод массивов

Возможны три основных способа задания значений массива:

- вычисление значений в программе;
- значения вводит пользователь;
- связывание с источником данных.

В задачах этого раздела ограничимся пока рассмотрением первых двух способов. В классе, работающем с массивами, всегда полезно иметь метод `FillArray`, позволяющий заполнять массив случайными числами.

7.3.1. Ввод-вывод массивов в Windows-приложениях

Приложения Windows позволяют построить «дружелюбный» интерфейс пользователя, облегчающий работу по вводу и выводу массивов. И здесь, когда данные задаются пользователем, заполнение массива проходит через те же этапы, что рассматривались для консольных приложений. Но выглядит все это более красиво, наглядно и понятно. Пример подобного интерфейса, обеспечивающего работу по вводу и выводу одномерного массива, показан на рис. 6.

Пользователь вводит в текстовое окно число элементов массива и нажимает командную кнопку "Создать массив", обработчик которой создает массив заданной размерности, если корректно задан размер массива, в противном случае выдает сообщение об ошибке и ждет корректного ввода.

В случае успешного создания массива пользователь может переходить к следующему этапу - вводу элементов массива. Очередной элемент массива вводится в текстовое окно, а обработчик командной кнопки "Ввести элемент" обеспечивает передачу значения в массив. Корректность ввода контролируется и на этом этапе, проверяя значение введенного элемента и выводя в специальное окно сообщение в случае его некорректности, добиваясь, в конечном итоге, получения от пользователя корректного ввода.

Для облегчения работы пользователя выводится подсказка, какой именно элемент должен вводить пользователь. После того, как все элементы массива введены, окно ввода становится недоступным для ввода элементов. Интерфейс формы позволяет многократно создавать новый массив, повторяя весь процесс.

На рис. 6 форма разделена на две части - для ввода и вывода массива. Крайне важно уметь организовать ввод массива, принимая данные от пользователя. Не менее важно уметь отображать существующий массив в форме, удобной для восприятия пользователя. На рисунке показаны три различных элемента управления, пригодные для этих целей, - ListBox, CheckedListBox и ComboBox. Как только вводится очередной элемент, он немедленно отображается во всех трех списках.

В реальности отображать массив в трех списках, конечно, не нужно, это сделано только в целях демонстрации возможностей различных элементов управления. Для целей вывода подходит любой из них, выбор зависит от контекста и предпочтений пользователя. Элемент ComboBox имеет дополнительное текстовое окно, в которое пользователь может вводить значение. Элемент CheckedListBox обладает дополнительными свойствами в сравнении с элементом ListBox, позволяя отмечать некоторые элементы списка (массива). Отмеченные пользователем элементы составляют специальную коллекцию. Эта коллекция доступна, с ней можно работать, что иногда весьма полезно. Чаще всего для вывода массива используется элемент ListBox.

Рис. 6. Форма для ввода-вывода одномерного массива

Посмотрим, как это все организовано программно. Начну с полей формы OneDimArrayForm, показанной на рис. 6:

```
//fields
int n = 0;
double[] mas;
int currentindex = 0;
double ditem = 0;
const string SIZE = "Корректно задайте размер массива!";
const string INVITE = "Введите число в формате m[,n]";
const string EMPTY = "Массив пуст!";
const string ITEMb = "mas[";
const string ITEMe = "] = ";
const string FULL = "Ввод недоступен!";
const string OK = "Корректный ввод!";
const string ERR = "Ошибка ввода числа! Повторите ввод!";
```

Полями этого класса является одномерный массив, его размер, текущий индекс и константы, используемые в процессе диалога с пользователем. Обработчик события Click командной кнопки, отвечающей за создание массива, имеет вид:

```
private void buttonCreateArray_Click(object sender, EventArgs e)
{
    try
    {
        n = Convert.ToInt32(textBoxN.Text);
        mas = new double[n];
        labelInvite.Text = INVITE;
        labelItem.Text = ITEMB + "0" + ITEME;
        labelResult.Text = EMPTY;
        textBoxItem.ReadOnly = false;
        listBox1.Items.Clear();
        comboBox1.Items.Clear();
        checkedListBox1.Items.Clear();
        comboBox1.Items.Clear();
        currentindex = 0;
    }
    catch (Exception)
    {
        labelResult.Text = SIZE;
    }
}
```

Первым делом принимается размер массива, введенный пользователем. Преобразование к типу `int` введенного значения помещено в охраняемый блок, поэтому ошибки некорректного ввода будут перехвачены с выдачей соответствующего сообщения. Если же массив успешно создан, то инициализируются начальными значениями все элементы интерфейса, участвующие в вводе элементов массива. Рассмотрим, как устроен ввод элементов.

```
private void buttonAddItem_Click(object sender, EventArgs e)
{
    //Заполнение массива элементами
    if (GetItem())
    {
        mas[currentindex] = ditem;
        listBox1.Items.Add(mas[currentindex]);
        checkedListBox1.Items.Add(mas[currentindex]);
        comboBox1.Items.Add(mas[currentindex]);
        currentindex++;
        labelItem.Text = ITEMB + currentindex + ITEME;
        textBoxItem.Text = "";
        labelResult.Text = OK;
        if (currentindex == n)
        {
            labelInvite.Text = "";
            labelItem.Text = "";
            labelResult.Text = FULL;
            textBoxItem.Text = "";
            textBoxItem.ReadOnly = true;
        }
    }
}
```

```
}
```

Функция `GetItem` вводит значение очередного элемента. Если пользователь корректно задал его значение, то элемент добавляется в массив, а заодно и в списки, отображающие текущее состояние массива. Создается подсказка для ввода следующего элемента массива, а если массив полностью определен, то форма переходит в состояние окончания ввода.

```
/// <summary>
/// Ввод с контролем текущего элемента массива
/// </summary>
/// <returns>true в случае корректного ввода значения</returns>
bool GetItem()
{
    string item = textBoxItem.Text;
    bool res = false;
    if (item == "")
        labelResult.Text = INVITE;
    else
    {
        try
        {
            ditem = Convert.ToDouble(item);
            res = true;
        }
        catch (Exception)
        {
            labelResult.Text = ERR;
        }
    }
    return res;
}
```

Форму `OneDimArrayForm` можно рассматривать как некоторый шаблон, полезный при организации ввода и вывода одномерных массивов.

7.3.2. Организация ввода-вывода двумерных массивов

Ввод двумерного массива немногим отличается от ввода одномерного массива. Сложнее обстоит дело с выводом двумерного массива, если при выводе пытаться отобразить структуру массива. К сожалению, все три элемента управления, хорошо справляющиеся с отображением одномерного массива, плохо приспособлены для показа структуры двумерного массива. Хотя у того же элемента `ListBox` есть свойство `MultiColumn`, включение которого позволяет показывать массив в виде строк и столбцов, но это не вполне то, что нужно для наших целей - отображения структуры двумерного массива. Хотелось бы, чтобы элемент имел такие свойства, как `Rows` и `Columns`, а их у элемента `ListBox` нет. Нет их и у элементов `ComboBox` и `CheckedListBox`. Приходится обходиться тем, что есть. На рис. 7 показан пример формы, поддерживающей работу по вводу и выводу двумерного массива.

Ввод-вывод двумерного массива

Ввод

n - строк 2 m - столбцов 3

Введите число в формате m[n]

mas[1, 2] =

Создать массив

Ввести элемент

Результат ввода Корректный ввод!

Вывод

12	23	-13
-5	29	

Рис. 7. Форма, поддерживающая ввод и вывод двумерного массива

Интерфейс формы схож с тем, что использовался для организации работы с одномерным массивом. Схожа и программная организация ввода-вывода элементов массива. Поэтому код, поддерживающий работу с формой TwoDimArrayForm, не приводится. Остановимся лишь на одном моменте, позволяющем отображать двумерный массив в элементе управления ListBox так, чтобы сохранялась структура строк и столбцов массива. Этого можно добиться за счет программной настройки размеров элемента управления ListBox:

```
listBox1.Height = n * HEIGHT_LINE;  
listBox1.Width = m * 2 * HEIGHT_LINE;
```

Константа HEIGHT_LINE задает высоту строки в списке. Вначале вносятся элементы первого столбца; когда весь столбец введен, автоматически следующее вводимое значение будет отображаться в первой строке в следующем столбце.

В общей ситуации, когда значения, вводимые пользователем, могут колебаться в широком диапазоне, трудно гарантировать отображение структуры двумерного массива. Однако ситуация не безнадежна. Есть и другие, более мощные и более подходящие для наших целей элементы управления. Рассмотрим подробнее элемент DataGridView.

7.3.3.Элемент управления DataGridView и отображение массивов

Элемент управления DataGridView является последней новинкой в серии табличных элементов DataGrid, позволяющих отображать таблицы. Главное назначение этих элементов - связывание с таблицами внешних источников данных, прежде всего с таблицами баз данных. Мы же сейчас рассмотрим другое его применение - в интерфейсе, позволяющем пользователю вводить и отображать матрицы - двумерные массивы.

Рассмотрим классическую задачу умножения прямоугольных матриц $C=A*B$. Построим интерфейс, позволяющий пользователю задавать размеры перемножаемых матриц, вводить данные для исходных матриц A и B, перемножать матрицы и видеть результаты этой операции. На рис. 8 показан возможный вид формы, поддерживающей работу пользователя. Форма показана в тот момент, когда пользователь уже задал размеры и значения исходных матриц, выполнил умножение матриц и получил результат.

На форме расположены три текстовых окна для задания размеров матриц, три элемента DataGridView для отображения матриц, три командные кнопки для выполнения операций, доступных пользователю. Кроме того, на форме присутствуют 9 меток (элементов управления label), семь из которых видимы на рис. 8. В них отображается информация, связанная с формой и отдельными элементами управления. Текст у невидимых на рисунке меток появляется тогда, когда обнаруживается, что пользователь некорректно задал значение какого-либо элемента исходных матриц.

А теперь перейдем к описанию того, как этот интерфейс реализован. В классе Form2, которому принадлежит наша форма, зададим поля, определяющие размеры матриц, и сами матрицы:

```
//поля класса Form
    int m, n, p;    //размеры матриц
    double[,] A, B, C; //сами матрицы
```

Рассмотрим теперь, как выглядит обработчик события "Click" командной кнопки "Создать DataGridView". Предполагается, что пользователь разумен и, прежде чем нажать эту кнопку, задает размеры матриц в соответствующих текстовых окнах. Напомню, что при перемножении матриц размеры матриц должны быть согласованы - число столбцов первого сомножителя должно совпадать с числом строк второго сомножителя, а размеры результирующей матрицы определяются размерами сомножителей. Поэтому для трех матриц в данном случае достаточно задать не шесть, а три параметра, определяющие размеры.

Умножение матриц

Задать размеры матриц!

m = n = p =

Матрица A (m*n)

	Column0	Column1	Column2
row0	-4	2	9
row1	3	5	11
►*			

Матрица B (n*p)

	Column0	Column1
row0	2	8
row1	4	9
row2	2	-6
►*		

Матрица C (m*p)

	Column0	Column1
row0	18	-68
row1	48	3
►*		

Создать DataGridView Перенести данные в массив Умножить матрицы

Рис. 8. Форма с элементами DataGridView, поддерживающая работу с матрицами

Обработчик события выполняет три задачи - создает сами матрицы, осуществляет чистку элементов управления DataGridView, удаляя предыдущее состояние, затем добавляет столбцы и строки в эти элементы в полном соответствии с заданными размерами матриц. Вот текст обработчика:

```
private void button1_Click(object sender, EventArgs e)
{
    //создание матриц
    m = Convert.ToInt32(textBox1.Text);
    n = Convert.ToInt32(textBox2.Text);
    p = Convert.ToInt32(textBox3.Text);
    A = new double[m, n];
    B = new double[n, p];
    C = new double[m, p];
    //Чистка DGView, если они не пусты
    int k = 0;
    k = dataGridView1.ColumnCount;
    if (k != 0)
```

```

        for (int i = 0; i < k; i++)
            dataGridView1.Columns.RemoveAt(0);
        dataGridView2.Columns.Clear();
        dataGridView3.Columns.Clear();
        //Заполнение DGView столбцами
        AddColumns(n, dataGridView1);
        AddColumns(p, dataGridView2);
        AddColumns(p, dataGridView3);
        //Заполнение DGView строками
        AddRows(m, dataGridView1);
        AddRows(n, dataGridView2);
        AddRows(m, dataGridView3);
    }

```

Прокомментирую этот текст.

- Прием размеров и создание матриц, надеюсь, не требует дополнительных комментариев.
- Чистка предыдущего состояния элементов DataGridView сводится к удалению столбцов. Продемонстрированы два возможных способа выполнения этой операции. Для первого элемента показано, как можно работать с коллекцией столбцов. Организуется цикл по числу столбцов коллекции, и в цикле выполняется метод `RemoveAt`, аргументом которого является индекс удаляемого столбца. Поскольку после удаления столбца происходит перенумерация столбцов, на каждом шаге цикла удаляется первый столбец, индекс которого всегда равен нулю. Удаление столбцов коллекции можно выполнить одним махом - вызывая метод `Clear()` коллекции, что и делается для остальных двух элементов DataGridView.
- После чистки предыдущего состояния, можно задать новую конфигурацию элемента, добавив в него вначале нужное количество столбцов, а затем и строк. Эти задачи выполняют специально написанные процедуры `AddColumns` и `AddRows`. Вот их текст:

```

private void AddColumns(int n, DataGridView dgw)
{
    //добавляет n столбцов в элемент управления dgw
    //Заполнение DGView столбцами
    DataGridViewColumn column;
    for (int i = 0; i < n; i++)
    {
        column = new DataGridViewTextBoxColumn();
        column.DataPropertyName = "Column" + i.ToString();
        column.Name = "Column" + i.ToString();
        dgw.Columns.Add(column);
    }
}
private void AddRows(int m, DataGridView dgw)
{
    //добавляет m строк в элемент управления dgw
    //Заполнение DGView строками
    for (int i = 0; i < m; i++)
    {
        dgw.Rows.Add();
        dgw.Rows[i].HeaderCell.Value

```

```

        = "row" + i.ToString();
    }
}

```

Приведу краткий комментарий.

- Создаются столбцы в коллекции `Columns` по одному. В цикле по числу столбцов матрицы, которую должен отображать элемент управления `DataGridView`, вызывается метод `Add` этой коллекции, создающий очередной столбец. Одновременно в этом же цикле создается и имя столбца (свойство `Name`), отображаемое в форме. Показана возможность формирования еще одного имени (`DataPropertyName`), используемого при связывании со столбцом таблицы внешнего источника данных. В нашем примере это имя не используется.
- Создав столбцы, нужно создать еще и нужное количество строк у каждого из элементов `DataGridView`. Делается это аналогичным образом, вызывая метод `Add` коллекции `Rows`. Чуть по-другому задаются имена строк - для этого используется специальный объект `HeaderCell`, имеющийся у каждой строки и задающий ячейку заголовка.
- После того как сформированы строки и столбцы, элемент `DataGridView` готов к тому, чтобы пользователь или программа вводила значения в ячейки сформированной таблицы.

Рассмотрим теперь, как выглядит обработчик события "Click" следующей командной кнопки "Перенести данные в массив". Предполагается, что пользователь разумен и, прежде чем нажать эту кнопку, задает значения элементов перемножаемых матриц в соответствующих ячейках подготовленных таблиц первых двух элементов `DataGridView`. Обработчик события выполняет следующие задачи - в цикле читает элементы, записанные пользователем в таблицы `DataGridView`, проверяет их корректность и в случае успеха переписывает их в матрицы. Вот текст обработчика:

```

private void button2_Click(object sender, EventArgs e)
{
    string elem = "";
    bool correct = true;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
        {
            try
            {
                elem=dataGridView1.Rows[i].Cells[j].Value.ToString();
                A[i, j] = Convert.ToDouble(elem);
                label8.Text = "";
            }
            catch (Exception any)
            {
                label8.Text = "Значение элемента" +
                    "A[" + i.ToString() + ", " + j.ToString() + " ]"
                    + " не корректно. Повторите ввод!";
                dataGridView1.Rows[i].Cells[j].Selected= true;
                return;
            }
        }
    for (int i = 0; i < n; i++)

```

```

        for (int j = 0; j < p; j++)
        {
            do
            {
                correct = true;
                try
                {
                    elem =
dataGridView2.Rows[i].Cells[j].Value.ToString();
                    B[i, j] = Convert.ToDouble(elem);
                    label9.Text = "";
                }
                catch (Exception any)
                {
                    label9.Text = "Значение элемента" +
                        "B[" + i.ToString() + ", " + j.ToString() + "]"
                        + " не корректно. Повторите ввод!";
                    dataGridView2.Rows[i].Cells[j].Selected=true;
                    Form3 frm = new Form3();
                    frm.label1.Text =
                        "B[" + i.ToString() + ", " + j.ToString() + "]= ";
                    frm.ShowDialog();
                    dataGridView2.Rows[i].Cells[j].Value =
                    frm.textBox1.Text;
                    correct = false;
                }
            } while (!correct);
        }
    }
}

```

Этот программный код нуждается в подробных комментариях.

- Основная задача переноса данных из таблицы элемента DataGridView в соответствующий массив не вызывает проблем. Конструкция `Rows[i].Cells[j]` позволяет добраться до нужного элемента таблицы, после чего остается присвоить его значение элементу массива.
- Как всегда при вводе основной проблемой является обеспечение корректности вводимых данных. Схема, рассматриваемая нами ранее, нуждается в корректировке. Дело в том, что ранее проверка корректности осуществлялась сразу же после ввода пользователем значения элемента. Теперь проверка корректности выполняется после того, как пользователь полностью заполнил таблицы, при этом некоторые элементы он мог задать некорректно. Просматривая таблицу, необходимо обнаружить некорректно заданные значения и предоставить возможность их исправления. В программе предлагаются два различных подхода к решению этой проблемы.
- Первый подход демонстрируется на примере ввода элементов матрицы A. Как обычно, преобразование данных, введенных пользователем, в значение, допустимое для элементов матрицы A, помещается в охраняемый блок. Если данные некорректны и возникает исключительная ситуация, то она перехватывается универсальным обработчиком `catch(Exception)`. Заметьте, в данном варианте нет цикла, работающего до тех пор, пока не будет введено корректное значение. Обработчик исключения просто прерывает работу по переносу данных, вызывая оператор `return`. Но предварительно он формирует

информационное сообщение об ошибке и выводит его в форму. (Помните, специально для этих целей у формы были заготовлены две метки). В сообщении пользователю предлагается исправить некорректно заданный элемент и повторить ввод - повторно нажать командную кнопку "перенести данные в массив". Этот подход понятен и легко реализуем. Недостатком является его неэффективность, поскольку повторно будут переноситься в массив все элементы, в том числе и те, что были введены вполне корректно. У программиста такая ситуация может вызывать чувство неудовлетворенности своей работой.

- На примере ввода элементов матрицы в продемонстрируем другой подход, когда исправляется только некорректно заданное значение. Прежде чем читать дальше, попробуйте найти собственное решение этой задачи. Это не так просто, как может показаться с первого взгляда. Для организации диалога с пользователем пришлось организовать специальное диалоговое окно, представляющее обычную форму с двумя элементами управления - меткой для выдачи информационного сообщения и текстовым окном для ввода пользователем корректного значения. При обнаружении ошибки ввода открывается диалоговое окно, в которое пользователь вводит корректное значение элемента и закрывает окно диалога. Введенное пользователем значение переносится в нужную ячейку таблицы `DataGridView`, а оттуда в матрицу.
- При проектировании диалогового окна значение свойства формы `FormBorderStyle`, установленное по умолчанию как "sizeable", следует заменить значением "FixedDialog", что влияет на внешний вид и поведение формы. Важно отметить, что форма, представляющая диалоговое окно, должна вызываться не методом `Show`, а методом `ShowDialog`. Иначе произойдет зацикливание, начнут порождаться десятки диалоговых окон, прежде чем вы успеете нажать спасительную в таких случаях комбинацию **Ctrl+ Alt + Del**.

Обработчик события "Click" командной кнопки "Умножить матрицы" выполняет ответственные задачи - реализует умножение матриц и отображает полученный результат в таблице соответствующего элемента `DataGridView`. Но оба эти действия выполняются естественным образом, не требуя, кроме циклов, никаких специальных средств и программистских ухищрений. Я приведу программный код без дополнительных комментариев:

```
private void button3_Click(object sender, EventArgs e)
{
    MultMatr(A, B, C);
    FillDG();
}

void MultMatr(double[,] A, double[,] B, double[,] C)
{
    int m = A.GetLength(0);
    int n = A.GetLength(1);
    int p = B.GetLength(1);
    double S = 0;
    for (int i = 0; i < m; i++)
        for (int j = 0; j < p; j++)
        {
            S = 0;
            for (int k = 0; k < n; k++)
                S += A[i, k] * B[k, j];
            C[i, j] = S;
        }
}
```

```
    }  
void FillDG()  
{  
    for (int i = 0; i < m; i++)  
        for (int j = 0; j < p; j++)  
            dataGridView3.Rows[i].Cells[j].Value  
                = C[i, j].ToString();  
}
```

Справка по объекту DataGridView.

Объект для отображения табличной информации DataGridView

Объект **DataGridView** предназначен для отображения всей информации из таблиц, запросов или фильтров на форме в виде таблицы. Этот объект может быть создан как вручную (с последующим его подключением), так и перетаскиванием всего источника данных из окна **"Data Sources"**. Однако наиболее часто его создают перетаскиванием всей таблицы, запроса или фильтра из окна **"Data Sources"** на форму.

При перетаскивании этого объекта на форму, как и в случае с другими объектами появляется панель навигации. Она выполняет функции: перемещение по записям, добавление, удаление и сохранение записей. После создания объекта **DataGridView** можно настраивать как свойства всего объекта, так и свойства отдельных столбцов. Начнём с настройки свойств всего объекта. Настройка данных свойств осуществляется в основном через меню действий. Возможны следующие настройки:

- **Chose Data Source** - источник данных, отображаемый в таблице;
- **Enable Adding** - добавлять записи;
- **Enable Deleting** - разрешается пользователям удалять записи;
- **Enable Editing** - разрешается пользователям изменять значения полей таблицы;
- **Enable Column Reordering** - разрешается пользователям изменять порядок столбцов, просто перетаскивая их мышью.

Также в меню действий возможны следующие действия с таблицей:

- **Dock in parent container** - вписать объект в форму;
- **Preview Data** - появляется окно с предварительным просмотром таблицы;
- **Add Query** - добавляет SQL - запрос, который выполняется на стороне клиента;
- **Add Column** - добавление нового столбца в таблицу;
- **Edit Columns** - настройка свойств отдельных столбцов таблицы.

Теперь перейдём к настройке отдельных столбцов таблицы.

Настройка свойств столбцов в DataGridView

Если в меню действий выбрать пункт **"Edit Columns"**, то появляется окно, где можно добавлять, удалять и редактировать столбцы. Для этого в списке столбцов левой части окна выбираем столбец, а в правой - настраиваем его свойства. Наиболее часто настраиваются следующие свойства:

1. **Name** - имя столбца;
2. **AutoSizeMode** - подгонка ширины столбца по его содержимому;
3. **ColumnType** - определяет внешний вид ячеек столбца (какой объект для отображения информации находится в ячейках столбца);
4. **DataPropertyName** - имя, отображающего в столбце поля;
5. **Frozen** - фиксация столбца (столбец не передвигается при прокручивании таблицы);
6. **HeaderText** - текст заголовка столбца;
7. **Width** - ширина поля;
8. **MaxInputLength** - максимально вводимая длина текста;
9. **MinimumWidth** - минимальная ширина столбца;
10. **ReadOnly** - блокировка столбца для редактирования данных;
11. **Resizable** - разрешает менять ширину столбца;
12. **SortMode** - сортировка данных в таблице по этому столбцу;
13. **ToolTipText** - всплывающая подсказка для столбца;
14. **Visible** - делает столбец невидимым.

Замечание: Для добавления новых столбцов в окне **"Edit Columns"** необходимо нажать кнопку **Add**, а для удаления кнопку **Remove**.

Замечание: Если необходимо настроить внешний вид всех ячеек таблицы, то для этого необходимо выделить объект **DataGridView** и на панели свойств зайти в свойство **DefaultCellStyle**. Появится окно со свойствами всех ячеек таблицы.

Замечание: В объекте **DataGridView** имеется возможность сортировки данных. Для этого используется метод **Sort**, имеющий следующий синтаксис:

```
DataGridView.Sort(<Имя столбца>, <Порядок сортировки>)
```

где **DataGridView** - это имя объекта, **<Имя столбца>** - это имя столбца (свойство **Name**) по которому происходит сортировка записей в таблице, параметр **<Порядок сортировки>** определяет порядок сортировки и может принимать два значения:

- **System.ComponentModel.ListSortDirection.Ascending** - сортировка по возрастанию;
- **System.ComponentModel.ListSortDirection.Descending** - сортировка по убыванию.

Замечание: Доступ к отдельным ячейкам таблицы можно получить через подобъект **Item**. Обращение к нему осуществляется следующим образом:

```
DataGridView.Item(i, j).<Свойство>
```

Здесь **DataGridView** - это имя объекта, **i** - горизонтальная координата ячейки, а **j** - вертикальная, **<Свойство>** - это настраиваемое свойство ячейки.

Пример: В верхнюю левую ячейку таблицы записать слово "Привет" и сделать цвет текста в ячейке красным.

```
DataGridView.Item(0, 0).Value = "Привет"  
DataGridView.Item(0, 0).Style.ForeColor = Color.Red
```

Здесь DataGridView - это имя объекта, свойство Value определяет содержимое ячейки таблицы, свойство Style.ForeColor определяет цвет текста в ячейке. Нумерация столбцов и строк в таблице начинается с нуля.

Источники:

1. Биллиг В. Основы программирования на С#. Режим доступа: <http://www.intuit.ru/studies/courses/2247/18/info>
2. Биллиг В. Основы программирования на С# 3.0. Ядро языка. Режим доступа: <http://www.intuit.ru/studies/courses/1094/428/info>