Лабораторная работа №2

Задание 1. Циклические алгоритмы

Цель задания: изучить средства отладки программ в среде Visual Studio. Составить и отладить программу циклического алгоритма.

1.1. Операторы организации циклов

Под *циклом* понимается многократное выполнение одних и тех же операторов при различных значениях промежуточных данных. Число повторений может быть задано в явной или неявной форме.

К операторам цикла относятся: *цикл с предусловием* `while`, *цикл с постусловием* `do while`, *цикл с параметром* `for` и *цикл перебора* `foreach`. Рассмотрим некоторые из них.

1.2. Цикл с предусловием

Оператор цикла `while` организует выполнение одного оператора (простого или составного) неизвестное заранее число раз. Формат цикла `while`:

```
while (B) S;
```

где `B` – выражение, истинность которого проверяется (условие завершения цикла); `S` – тело цикла – оператор (простой или составной).

Перед каждым выполнением тела цикла анализируется значение выражения B: если оно истинно, то выполняется тело цикла, и управление передается на повторную проверку условия B; если значение B ложно – цикл завершается и управление передается на оператор, следующий за оператором `S`.

Если результат выражения `B` окажется ложным при первой проверке, то тело цикла не выполнится ни разу. Отметим, что если условие `B` во время работы цикла не будет изменяться, то возможна ситуация зацикливания, то есть невозможность выхода из цикла. Поэтому внутри тела должны находиться операторы, приводящие к изменению значения выражения `B` так, чтобы цикл мог корректно завершиться.

В качестве иллюстрации выполнения цикла `while` рассмотрим программу вывода целых чисел от 1 до n по нажатию кнопки на форме:

```
private void button1_Click(object sender, EventArgs e) {
int n = 10; // Количество повторений цикла
int i = 1; // Начальное значение
while (i <= n) // Пока i меньше или равно n
   {
       MessageBox.Show(i.ToString()); // Показываем i
    i++; // Увеличиваем i на 1 }
    }
 }
```

1.3. Цикл с постусловием

Оператор цикла `do while` также организует выполнение одного оператора (простого или

составного) неизвестное заранее число раз. Однако в отличие от цикла `while` условие завершения цикла проверяется после выполнения тела цикла. Формат цикла `do while`:

```
do S while (B);
```

где `B` – выражение, истинность которого проверяется (условие завершения цикла); `S` – тело цикла – оператор (простой или блок).

Сначала выполняется оператор `S`, а затем анализируется значение выражения `B`: если оно истинно, то управление передается оператору `S`, если ложно – цикл завершается, и управление передается на оператор, следующий за условием `B`. Так как условие `B` проверяется после выполнения тела цикла, то в любом случае тело цикла выполнится хотя бы один раз.

В операторе `do while`, так же как и в операторе `while`, возможна ситуация зацикливания в случае, если условие `B` всегда будет оставаться истинным.

1.4. Цикл с параметром

Цикл с параметром имеет следующую структуру:

```
for (<инициализация>; <выражение>; <модификация>) <оператор>;
```

Инициализация используется для объявления и/или присвоения начальных значений величинам, используемым в цикле в качестве параметров (счетчиков). В этой части можно записать несколько операторов, разделенных запятой. Областью действия переменных, объявленных в части инициализации цикла, является цикл и вложенные блоки. Инициализация выполняется один раз в начале исполнения цикла.

Выражение определяет условие выполнения цикла: если его результат истинен, цикл выполняется. Истинность выражения проверяется перед каждым выполнением тела цикла, таким образом, цикл с параметром реализован как *цикл с предусловием*. В блоке *выражение* через запятую можно записать несколько логических выражений, тогда запятая равносильна операции *логическое И* (&&).

Модификация выполняется после каждой итерации цикла и служит обычно для изменения параметров цикла. В части *модификация* можно записать несколько операторов через запятую.

Оператор (простой или составной) представляет собой тело цикла.

Любая из частей оператора `for` (инициализация, выражение, модификация, оператор) может отсутствовать, но точку с запятой, определяющую позицию пропускаемой части, надо оставить.

Пример формирования строки, состоящей из чисел от 0 до 9, разделенных пробелами:

```
string s = ""; // Инициализация строки
for (int i = 0; i <= 9; i++) // Перечисление всех чисел
s += i.ToString() + " "; // Добавляем число и пробел
MessageBox.Show(s.ToString()); // Показываем результат
```

Данный пример работает следующим образом. Сначала вычисляется начальное значение переменной `i`. Затем, пока значение `i` меньше или равно 9, выполняется тело цикла, и затем

повторно вычисляется значение `i`. Когда значение `i` становится больше 9, условие – ложно и управление передается за пределы цикла.

1.5. Средства отладки программ

Практически в каждой вновь написанной программе после запуска обнаруживаются ошибки.

Ошибки первого уровня (ошибки компиляции) связаны с неправильной записью операторов (орфографические, синтаксические). При обнаружении ошибок компилятор формирует список, который отображается по завершению компиляции (рис. 1.1). При этом возможен только запуск программы, которая была успешно скомпилирована для предыдущей версии программы.При выявлении ошибок компиляции в нижней части экрана появляется текстовое

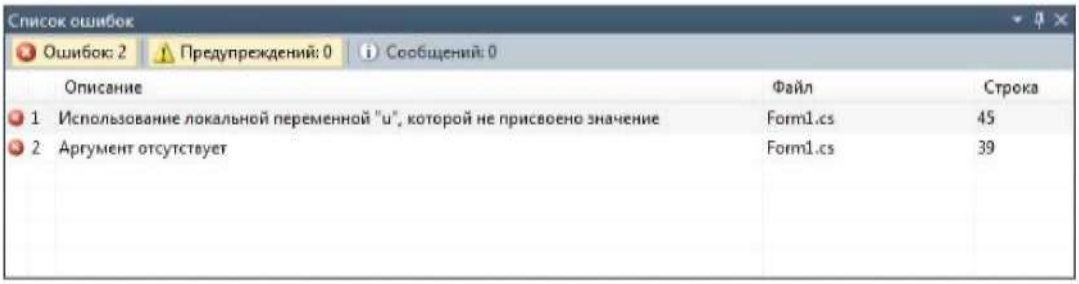

Рис. 1.1. Окно со списком ошибок компиляции

окно (см. рис. 1.1), содержащее сведения обо всех ошибках, найденных в проекте. Каждая строка этого окна содержит имя файла, в котором найдена ошибка, номер строки с ошибкой и характер ошибки. Для быстрого перехода к интересующей ошибке необходимо дважды щелкнуть на строке с ее описанием. Следует обратить внимание на то, что одна ошибка может повлечь за собой другие, которые исчезнут при ее исправлении. Поэтому необходимо исправлять их последовательно, сверху вниз и, после исправления каждой – компилировать программу снова.

Ошибки второго уровня (ошибки выполнения) связаны с ошибками выбранного алгоритма решения или с неправильной программной реализацией алгоритма. Эти ошибки проявляются в том, что результат расчета оказывается неверным либо происходит переполнение, деление на ноль и др. Поэтому перед использованием отлаженной программы ее надо протестировать, т. е. сделать просчеты при таких комбинациях исходных данных, для которых заранее известен результат. Если тестовые расчеты указывают на ошибку, то для ее поиска следует использовать встроенные средства отладки среды программирования.

В простейшем случае для локализации места ошибки рекомендуется поступать следующим образом. В окне редактирования текста установить точку останова перед подозрительным участком, которая позволит остановить выполнение программы и далее более детально следить за ходом работы операторов и изменением значений переменных. Для этого достаточно в окне редактирования кода щелкнуть слева от нужной строки. В результате чего данная строка будет выделена красным (рис. 1.2).

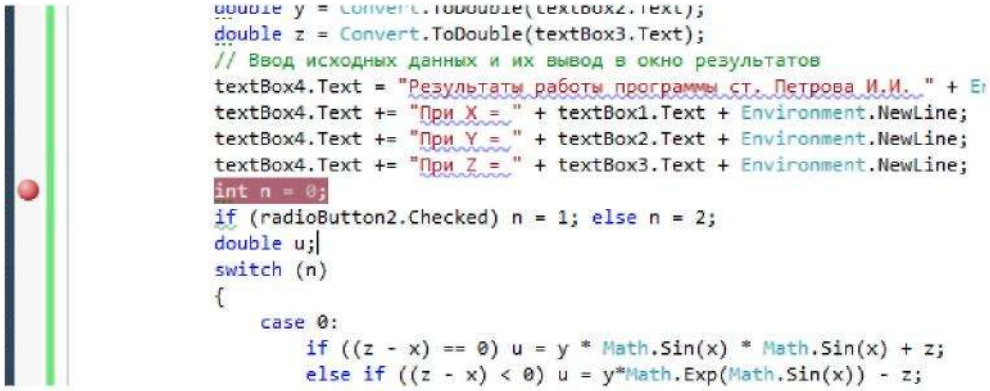

Рис. 1.2. Фрагмент кода с точкой останова

При выполнении программы и достижения установленной точки программа будет остановлена, и далее можно выполнять код по шагам с помощью команд *Отладка -> Шаг с обходом* (без захода в методы) или *Отладка -> Шаг с заходом* (с заходом в методы) (рис. 1.3).

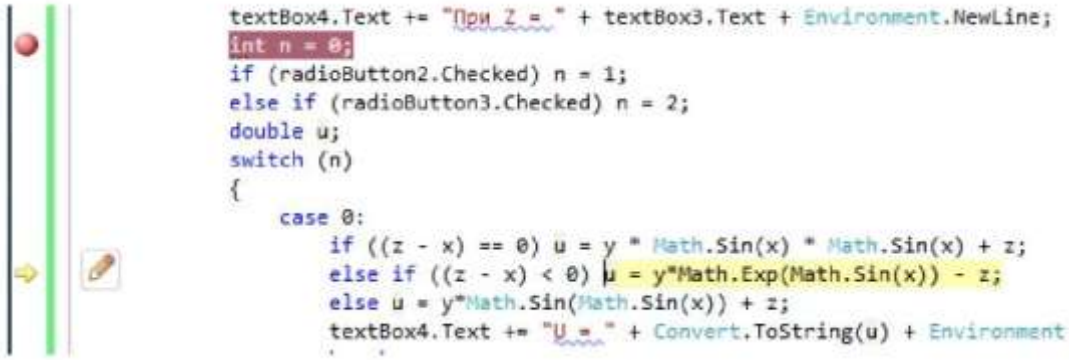

Рис. 1.3. Отладка программы

Желтым цветом выделяется оператор, который будет выполнен. Значение переменных во время выполнения можно увидеть, наведя на них курсор. Для прекращения отладки и остановки программы нужно выполнить команду меню *Отладка -> Остановить отладку.*

Для поиска алгоритмических ошибок можно контролировать значения промежуточных переменных на каждом шаге выполнения подозрительного кода и сравнивать их с результатами, полученными вручную.

1.6. Порядок выполнения задания

Задание: Вычислить и вывести на экран таблицу значений функции $y = a*ln(x)$ при x, изменяющемся от x_0 до x_k с шагом dx, a – константа.

Панель диалога представлена на рис. 1.4. Текст обработчика нажатия кнопки **Вычислить** приведен ниже.

```
private void buttonl_Click(object sender, EventArgs e)
    {
    // Считывание начальных данных
    double x0 = Convert.ToDouble(textBoxl.Text);
    double xk = Convert.ToDouble(textBox2.Text);
```

```
double dx = Convert.ToDouble(textBox3.Text);
double a = Convert.ToDouble(textBox4.Text);
textBox5.Text = "Работу выполнил ст. Иванов М.А." +
    Environment.NewLine;
    // Цикл для табулирования функции
    double x = x0;
    while (x <= (xk + dx / 2))
    {
    double y = a * Math.Log(x);
    textBox5.Text += "x=" + Convert.ToString(x) +
    "; y=" + Convert.ToString(y) +
     Environment.NewLine; x = x + dx;
    }
}
```

После отладки программы следует проверить правильность работы программы с помощью контрольного примера (см. рис. 1.4). Установите точку останова на оператор перед циклом и запустите программу. После попадания на точку остановки, выполните пошагово программу и проследите, как меняются все переменные в процессе выполнения.

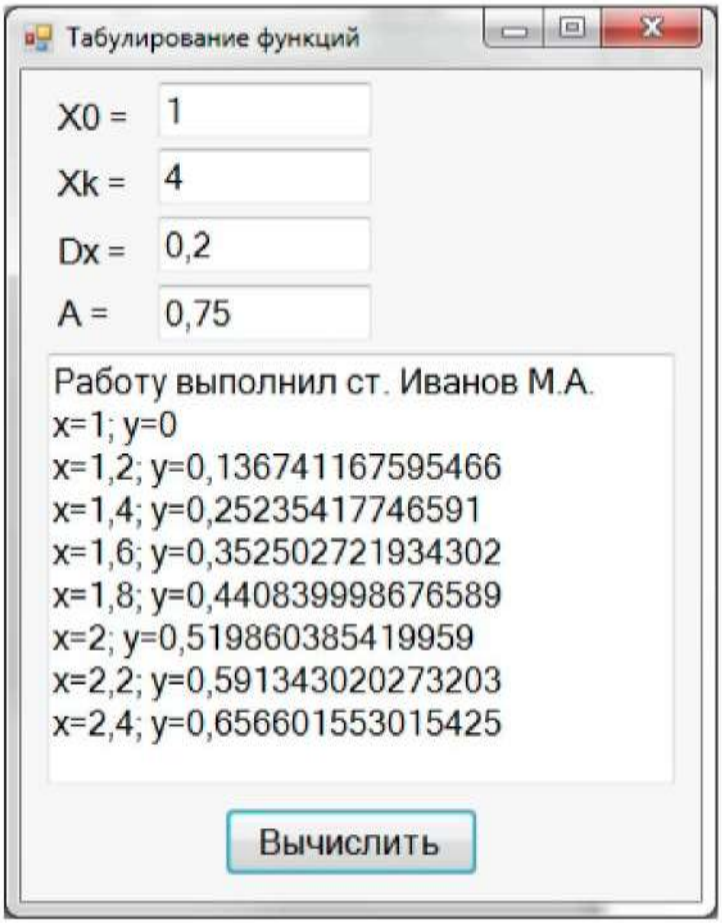

Рис. 1.4. Окно программы для табулирования функции

Индивидуальные задания

Составьте программу табулирования функции $y(x)$, выведите на экран значения x и $y(x)$. Нужный вариант задания выберите из нижеприведенного списка по указанию преподавателя. Откорректируйте элементы управления в форме в соответствии со своим вариантом задания.

1) $y = 10^{-2}bc / x + \cos\sqrt{a^3 x},$
$x_0 = -1.5; x_k = 3.5; dx = 0.5;$
$a = -1.25; b = -1.5; c = 0.75;$

2) $y = 1.2(a-b)^3 e^{x^2} + x,$
$x_0 = -0.75; x_k = -1.5; dx = -0.05;$
$a = 1.5; b = 1.2;$

3) $y = 10^{-1} a x^3 tg(a - bx)$,
$x_0 = -0.5; x_k = 2.5; dx = 0.05;$
$a = 10.2; b = 1.25;$

4) $y = ax^3 + \cos^2(x^3 - b)$,
$x_0 = 5.3; x_k = 10.3; dx = 0.25;$
$a = 1.35; b = -6.25;$

5) $y = x^4 + \cos(2 + x^3 - d)$,
$x_0 = 4.6; x_k = 5.8; dx = 0.2;$
$d = 1.3;$

6) $y = x^2 + tg(5x + b/x)$,
$x_0 = -1.5; x_k = -2.5; dx = -0.5;$
$b = -0.8;$

7) $y = 9(x + 15\sqrt{x^3 + b^3})$,
$x_0 = -2.4; x_k = 1; dx = 0.2;$
$b = 2.5;$

8) $y = 9x^4 + \sin(57.2 + x)$,
$x_0 = -0.75; x_k = -2.05; dx = -0.2;$

9) $y = 0.0025bx^3 + \sqrt{x + e^{0.82}}$,
$x_0 = -1; x_k = 4; dx = 0.5;$
$b = 2.3;$

10) $y = x \cdot \sin(\sqrt{x + b - 0.0084})$,
$x_0 = -2.05; x_k = -3.05; dx = -0.2;$
$b = 3.4;$

11) $y = x + \sqrt{|x^3 + a - be^x|}$,
$x_0 = -4; x_k = -6.2; dx = -0.2;$
$a = 0.1;$

12) $y = 9(x^3 + b^3)tgx$,
$x_0 = 1; x_k = 2.2; dx = 0.2;$
$b = 3.2;$

13) $y = |x - b|^{1/2} / |b^3 - x^3|^{3/2} + \ln|x - b|$,
$x_0 = -0.73; x_k = -1.73; dx = -0.1;$
$b = -2;$

14) $y = (x^{5/2} - b)\ln(x^2 + 12.7)$,
$x_0 = 0.25; x_k = 5.2; dx = 0.3;$
$b = 0.8;$

15) $y = 10^{-3}|x|^{5/2} + \ln|x + b|$,
$x_0 = 1.75; x_k = -2.5; dx = -0.25;$
$b = 35.4;$

16) $y = 15.28|x|^{-3/2} + \cos(\ln|x| + b)$,
$x_0 = 1.23; x_k = -2.4; dx = -0.3;$
$b = 12.6;$

17) $y = 0.00084(\ln|x|^{5/4} + b)/(x^2 + 3.82)$,
$x_0 = -2.35; x_k = -2; dx = 0.05;$
$b = 74.2;$

18) $y = 0.8 \cdot 10^{-5}(x^3 + b^3)^{7/6}$,
$x_0 = -0.05; x_k = 0.15; dx = 0.01;$
$b = 6.74;$

19) $y = (\ln(\sin(x^3 + 0.0025)))^{3/2} + 0.8 \cdot 10^{-3}$,
$x_0 = 0.12; x_k = 0.64; dx = 0.2;$

20) $y = a + x^{2/3}\cos(x + e^x)$,
$x_0 = 5.62; x_k = 15.62; dx = 0.5;$
$a = 0.41$

Задание 2. Классы и объекты

Цель задания: изучить основные понятия, относящиеся к классам и объектам, освоить динамическое создание объектов в программном коде.

2.1. Классы и объекты

В объектно-ориентированном подходе существуют понятия *класс* и *объект*.

Класс – это программная единица, которая задает общий шаблон для конкретных объектов. Класс содержит все необходимые описания переменных, свойств и методов, которые относятся к объекту. Примером класса в реальной жизни является *понятие «автомобиль»*: как правило, автомобиль содержит некоторое количество колес, дверей, имеет какой-то цвет, но эти конкретные детали в классе не описываются.

Объект – это экземпляр класса. Свойства объекта содержат конкретные данные, характерные для данного экземпляра. В реальной жизни примером объекта будет конкретный экземпляр автомобиля с 4 колесами, 5 дверками и синего цвета.

2.2. Динамическое создание объектов

Чаще всего для размещения на форме кнопки, поля ввода или других управляющих элементов используется дизайнер среды Visual Studio: нужный элемент выделяется в панели элементов и размещается на форме. Однако иногда создавать элементы нужно уже в процессе выполнения программы. Поскольку каждый элемент управления представляет собой отдельный класс, его помещение на форму программным способом включает несколько шагов:

1. Создание экземпляра класса.
2. Привязка его к форме.
3. Настройка местоположения, размеров, текста и т. п.

Например, чтобы создать кнопку, нужно выполнить следующий код (его следует разместить в обработчике сообщения `Load` или в каком-либо другом методе):

```
Button b = new Button();
```

Здесь объявляется переменная `b`, относящаяся к классу `Button`, как и в предыдущих лабораторных работах. Однако дальше идет нечто новое: с помощью оператора `new` создается экземпляр класса `Button`, и ссылка на него присваивается переменной `b`. При этом выполняется целый ряд дополнительных действий: выделяется память под объект, инициализируются все свойства и переменные.

Далее нужно добавить объект на форму. Для этого служит свойство `Parent`, которое определяет родительский элемент, на котором будет размещена кнопка:

```
b.Parent = this;
```

Ключевое слово `this` относится к тому объекту, в котором размещен выполняемый в данный момент метод. Поскольку все методы в лабораторных работах размещаются в классе формы, то и `this` относится к этому конкретному экземпляру формы.

Вместо формы кнопку можно поместить на другой контейнер. Например, если на форме есть элемент управления Panel, то можно поместить кнопку на него следующим образом:

```
b.Parent = panel1;
```

Чтобы задать положение и размеры кнопки, нужно использовать свойства Location и Size:

```
b.Location = new Point(10, 20);
b.Size = new Size(200, 100);
```

Обратите внимание, что `Location` и `Size` – это тоже объекты. Хотя внутри у `Location` содержатся координаты x и y, задающие левый верхний угол объекта, не получится поменять одну из координат, нужно менять целиком весь объект `Location`. То же самое относится и к свойству `Size`.

На самом деле, каждый раз, когда на форму помещается новый элемент управления или вносятся какие-то изменения в свойства элементов управления, Visual Studio генерирует специальный служебный код, который проделывает приведенные выше операции по созданию и настройке элементов управления. Попробуйте поместить на форму кнопку, изменить у нее какие-нибудь свойства, а затем найдите в обозревателе решений ветку формы `Form1`, разверните ее и сделайте двойной щелчок по ветке `Form1.Designer.cs`. Откроется файл с текстом программы на языке C#, которую среда создала автоматически. Менять этот код вручную крайне не рекомендуется! Однако можно его изучить, чтобы понять принципы создания элементов управления в ходе выполнения программы.

2.3. Область видимости

Переменные, объявленные в программе, имеют область видимости. Это значит, что переменная, описанная в одной части программы, не обязательно будет видна в другой. Вот наиболее часто встречающиеся ситуации:

1. Переменные, описанные внутри метода, не будут видны за пределами этого метода. Например:

```
void MethodA()
{
    // Описываем переменную delta
    int delta = 7;
}
void MethodB()
{
    // Ошибка: переменная delta в этом методе неизвестна!
    int gamma = delta + 1;
}
```

2. Переменные, описанные внутри блока или составного оператора, видны только внутри этого блока. Например:

```
void Method()
{
    if (a == 7)
    {
        int b = a + 5;
    }
    // Ошибка: переменная b здесь уже неизвестна!
```

```
        MessageBox.Show(b.ToString());
    }
```

3. Переменные, описанные внутри класса, являются *глобальными* и доступны для всех методов этого класса, например:

```
class Form1 : Form
{
    int a = 5;
    void Method()
    {
      // Переменная a здесь действительна
    MessageBox.Show(a.ToString());
    }
}
```

2.4. Операции is и as

Часто бывает удобно переменные разных классов записать в один список, чтобы было легче его обрабатывать. Чтобы проверить, к какому классу принадлежит какой-либо объект, можно использовать оператор `is`: он возвращает истину, если объект принадлежит указанному классу. Пример:

```
Button b = new Button();
if (b is Button)
   MessageBox.Show("Это кнопка!");
else
   MessageBox.Show("Это что-то другое...");
```

Как правило, в общих списках объекты хранятся в «обезличенном» состоянии, так, чтобы у всех у них был лишь минимальный общий для всех набор методов и свойств. Для того чтобы получить доступ к расширенным свойствам объекта, нужно *привести* его к исходному классу с помощью *операции приведения* `as`:

```
(someObject as Button).Text = "Это кнопка!";
```

Следует помнить, что операция приведения сработает только в том случае, если объект изначально принадлежит тому классу, к которому его пытаются привести (или совместим с ним), в противном случае оператор `as` выбросит исключение и остановит выполнение программы. Поэтому более безопасный подход состоит в комбинированном применении операторов `as` и `is`: сначала проверяем совместимость объекта и класса, и только потом выполняем операцию приведения:

```
if (someObject is Button)
(someObject as Button).Text = "Это кнопка!";
```

В качестве практического примера использования этих операций рассмотрим пример программы, которая перебирает все элементы управления на форме, и у кнопок (но не у других элементов управления!) заменяет текст на пять звездочек «*****»:

```
private void Form1_Load(object sender, EventArgs e)
{
// Перебираем все элементы управления
foreach (Control c in this.Controls)
if (c is Button) // Кнопка?
    (c as Button).Text = "*****"; // Да!
}
```

2.5. Сведения, передаваемые в событие

Когда происходит какое-либо событие (например, событие `Click` при нажатии на кнопку), в обработчик этого события передаются дополнительные сведения об этом событии в параметре e.

Например, при щелчке кнопки мыши на объекте возникает событие `MouseClick`. Для этого события параметр e содержит целый ряд переменных, которые позволяют узнать информацию о нажатии:

- `Button` - какая кнопка была нажата;
- `Clicks` - сколько раз была нажата и отпущена кнопка мыши;
- `Location` - координаты точки, на которую указывал курсор в момент нажатия, в виде объекта класса `Point;`
- `X` и `Y` - те же координаты в виде отдельных переменных.

Индивидуальные задания

Если в индивидуальном задании используется элемент `Panel,` измените его цвет, чтобы он визуально выделялся на форме. Если используется элемент `Label,` не забудьте присвоить ему какой-либо текст, иначе он не будет виден на форме.

1. Разработать программу, динамически порождающую на окне кнопки. Левый верхний угол кнопки определяется местоположением курсора при щелчке. Вывести надпись на кнопке с координатами ее левого верхнего угла.

2. Разработать программу, динамически порождающую на окне кнопки и поля ввода. Левый верхний угол элемента управления определяется местоположением курсора при щелчке. Кнопка порождается, если курсор находится в левой половине окна, в ином случае порождается поле ввода.

3. На форме размещен элемент управления `Panel.` Написать программу, которая при щелчке мыши на элементе управления Panel добавляет в него кнопки `Button,` а при щелчке на форме в нее добавляются поля ввода `TextBox.`

4. На форме размещены 3 панели (элемент управления Panel). Написать программу, которая при щелчке мыши на первой панели добавляет во вторую панель кнопки Button, при щелчке на второй панели добавляет в третью панель поля ввода TextBox, а при щелчке на третьей панели добавляет на первую панель метки Label. Написать программу, добавляющую на форму кнопки. Кнопки добавляются в узлы прямоугольной сетки. Расстояния между кнопками и

расстояния между крайней кнопкой и границей окна должны быть равны как по горизонтали, так и по вертикали.

5. Разработать программу, при щелчке мыши динамически порождающую на окне кнопки или поля ввода. Каждый четный элемент управления является кнопкой, нечетный – полем ввода. Левый верхний угол кнопки определяется местоположением курсора при щелчке. Для поля ввода положение курсора определяет координаты *правого нижнего* угла.

6. Создать программу с кнопкой, меткой и полем ввода. При щелчке на соответствующий элемент на форме динамически должен создаваться подобный ему элемент. Предусмотреть возможность вывода количества кнопок, меток и полей ввода.

7. Создать программу, добавляющую различные элементы управления на форму и на панель `Panel`. Тип элементов управления выбирается случайным образом. Предусмотреть возможность вывода информации о количестве элементов по типам и информацию о расположении элементов.

8. Разработать программу, добавляющую на форму последовательность элементов управления случайной длины. Тип элементов управления задается случайным образом. Предусмотреть возможность вывода информации о количестве элементов по типам.

9. Написать программу, динамически порождающую на окне кнопки или метки. Левый верхний угол элемента управления определяется местоположением курсора при щелчке. При нажатии правой кнопки мыши на форме с нее удаляются все кнопки.

10. Написать программу, динамически порождающую на окне поочередно кнопки или поля ввода. Левый верхний угол элемента управления определяется местоположением курсора при щелчке. При нажатии правой кнопки мыши на форме с нее удаляются все порожденные элементы.

11. Разработать программу с двумя кнопками на форме. При нажатии на первую на форму добавляется одна панель `Panel`. При нажатии на вторую кнопку в каждую панель добавляется поле ввода.

12. Разработать программу с двумя кнопками на форме. При нажатии на первую на форму добавляется одна кнопка или поле ввода. При нажатии на вторую кнопку каждое поле увеличивается по вертикали в два раза.

13. Написать программу с кнопкой и тремя полями ввода. При нажатии на кнопку программа анализирует содержимое первого поля и динамически порождает элемент управления. Если в первом поле ввода содержится буква «К», то на форму добавляется кнопка, если «П» – поле ввода, если «М» – метка. Во втором и третьем поле ввода содержатся координаты левого верхнего угла будущего элемента управления.

15. Разработать программу, добавляющую на форму метки с текстом. Местоположение и размеры меток определяются в программе динамически через поля ввода. В заголовок окна, анализируя размер всех меток, вывести количество маленьких и больших меток. Маленькой меткой считается метка размером менее 50 пикселей по горизонтали и вертикали.

16. Создать программу с двумя кнопками на форме, динамически порождающую на окне метки или поля ввода. При нажатии на первую кнопку каждая метка увеличивается по горизонтали в два раза. При нажатии на вторую кнопку каждое поле уменьшается по вертикали в два раза.

17. Разработать программу, динамически порождающую на окне кнопки и поля ввода. Координаты элемента управления определяются случайным образом. Элементы управления не должны накладываться друг на друга. Если нет возможности добавить элемент управления (нет места для размещения элемента), то предусмотреть вывод информации об этом.

18. Разработать программу, динамически порождающую на окне кнопки и поля ввода. Координаты элемента управления определяются случайным образом. При наведении курсора на элемент управления он должен быть удален с формы.

19. Разработать программу, динамически порождающую при щелчке на окне различные элементы (поля ввода, кнопки, метки). Тип элементов определяется с помощью радиокнопок. Все элементы располагаются горизонтально в ряд. При достижении правой границы окна начинается новый ряд элементов.

20. Разработать программу, динамически порождающую или поле ввода (при нажатии на окне левой кнопкой мыши), или кнопку (при нажатии на окне правой кнопкой мыши). Все элементы располагаются наискосок, начиная с левого верхнего угла окна. Реализовать обработчик события изменения размера окна, в котором удалить все порожденные элементы.

Задание 3. Методы

Цель задания: научиться работать с методами, написать программу с использованием методов.

3.1. Общие понятия

Метод - это элемент класса, который содержит программный код. Метод имеет следующую структуру:

```
[атрибуты] [спецификторы] тип имя ([параметры])
{
    Тело метода;
}
```

Атрибуты - это особые указания компилятору на свойства метода. Атрибуты используются редко.

Спецификаторы - это ключевые слова, предназначенные для разных целей, например:

- определяющие доступность метода для других классов:
  - `private` - метод будет доступен только внутри этого класса;
  - `protected` - метод будет доступен также дочерним классам;
  - `public` - метод будет доступен любому другому классу, который может получить доступ к данному классу;

- указывающие доступность метода без создания класса;
- задающие тип.

Тип определяет результат, который возвращает метод: это может быть любой тип, доступный в C#, а также ключевое слово `void`, если результат не требуется.

Имя метода - это идентификатор, который будет использоваться для вызова метода. К идентификатору применяются те же требования, что и к именам переменных: он может состоять

из букв, цифр и знака подчеркивания, но не может начинаться с цифры.

Параметры - это список переменных, которые можно передавать в метод при вызове. Каждый параметр состоит из типа и названия переменной. Параметры разделяются запятой.

Тело метода - это обычный программный код, за исключением того, что он не может содержать определения других методов, классов, пространств имен и т. д. Если метод должен возвращать какой-то результат, то обязательно в конце должно присутствовать ключевое слово return с возвращаемым значением. Если возвращение результатов не нужно, то использование ключевого слова return не обязательно, хотя и допускается.

Пример метода, вычисляющего выражение:

```
public double Calc(double a, double b, double c)
{
     if (a > b)
           return Math.Sin(a) * Math.Cos(b);
          else
          {
          double k = Math.Tan(a * b);
          return k * Math.Exp(c / k);
          }
}
```

3.2. Перегрузка методов

Язык C# позволяет создавать несколько методов с одинаковыми именами, но разными параметрами. Компилятор автоматически подберет наиболее подходящий метод при построении программы. Например, можно написать два отдельных метода возведения числа в степень: для целых чисел будет применяться один алгоритм, а для вещественных - другой:

```
/// <summary>
/// Вычисление X в степени Y для целых чисел
/// </summary>

private int Pow(int X, int Y)
{
     int b = 1;
     while (Y != 0)
          if (Y % 2 == 0)
          {
           Y /= 2;
           X *= X;
          }
          else
          {
           Y--;
```

```
            b *= X;
            }
        return b;
}

/// <summary>
/// Вычисление X в степени Y для вещественных чисел
/// </summary>
private double Pow(double X, double Y)
{
    if (X != 0)
        return Math.Exp(Y * Math.Log(Math.Abs(X)));
     else if (Y == 0)
       return 1;
     else
       return 0;
}
```

Вызывается такой код одинаково, разница лишь в параметрах – в первом случае компилятор вызовет метод `Pow` с целочисленными параметрами, а во втором – с вещественными:

```
Pow(3, 17);
Pow(3.0, 17.0);
```

3.3. Параметры по умолчанию

Язык C# начиная с версии 4.0 (Visual Studio 2010), позволяет задавать некоторым параметрам *значения по умолчанию* – так, чтобы при вызове метода можно было опускать часть параметров. Для этого при реализации метода нужным параметрам следует присвоить значение прямо в списке параметров:

```
private void GetData(int Number, int Optional = 5)
{
    MessageBox.Show("Number: {0}", Number);
    MessageBox.Show("Optional: {0}", Optional);
}
```

В этом случае вызывать метод можно следующим образом:

```
GetData(10, 20);
GetData(10);
```

В первом случае параметр `Optional` будет равен 20, так как он явно задан, а во втором будет равен 5, т. к. явно он не задан и компилятор берет значение по умолчанию.

Параметры по умолчанию можно ставить только в правой части списка параметров,

например, такая сигнатура метода компилятором принята не будет:

```
private void GetData(int Optional = 5, int Number)
```

3.4. Передача параметров по значению и по ссылке

Когда параметры передаются в метод обычным образом (без дополнительных ключевых слов `ref` и `out`), любые изменения параметров внутри метода не влияют на его значение в основной программе. Предположим, у нас есть следующий метод:

```
private void Calc(int Number)
  {
    Number = 10;
  }
```

Видно, что внутри метода происходит изменение переменной Number, которая была передана как параметр. Попробуем вызвать метод:

```
int n = 1;
Calc(n);
MessageBox.Show(n.ToString());
```

На экране появится число 1, то есть, несмотря на изменение переменной в методе `Calc`, значение переменной в главной программе не изменилось. Это связано с тем, что при вызове метода создается *копия* переданной переменной, именно ее изменяет метод. При завершении метода значение копий теряется. Такой способ передачи параметра называется *передачей по значению*.

Чтобы метод мог изменять переданную ему переменную, ее следует передавать с ключевым словом `ref` – оно должно быть как в сигнатуре метода, так и при вызове:

```
private void Calc(ref int Number)
{
   Number = 10;
}
int n = 1;
Calc(ref n);
MessageBox.Show(n.ToString());
```

В этом случае на экране появится число 10: изменение значения в методе сказалось и на главной программе. Такая передача метода называется *передачей по ссылке*, т. е. передается уже не копия, а ссылка на реальную переменную в памяти.

Если метод использует переменные по ссылке только для возврата значений и не имеет значения, что в них было изначально, то можно не инициализировать такие переменные, а передавать их с ключевым словом `out`. Компилятор понимает, что начальное значение переменной не важно, и не ругается на отсутствие инициализации:

```
private void Calc(out int Number)
    {
     Number = 10;
    }
int n; // Ничего не присваиваем!
Calc(out n);
```

Индивидуальное задание

1. Написать метод min(x, y), находящий минимальное значение из двух чисел. С его помощью найти минимальное значение из четырех чисел a, b, c, d.

2. Написать метод max(x, y), находящий максимальное значение из двух чисел. С его помощью найти максимальное значение из четырех чисел a, b, c, d.

3. Написать метод, вычисляющий значение n/x^n. С его помощью вычислить выражение:

$$\sum_{i=1}^{10} \frac{i}{x^i}.$$

4. Написать метод, вычисляющий значение n/x^n. С его помощью вычислить выражение:

$$\prod_{i=1}^{10} \frac{i}{x^i}.$$

5. Написать метод, вычисляющий значение $x^n/(n+x)$. С его помощью вычислить выражение:

$$\sum_{i=1}^{10} \frac{x^i}{x+i}.$$

6. Написать метод, вычисляющий значение sin(x) + cos(2 * x). С его помощью определить, в какой из точек a, b или c значение будет минимальным.

7. Написать метод, вычисляющий значение $x^2 + y^2$. С его помощью определить, с какой парой чисел (a, b) или (c, d) значение будет максимальным.

8. Написать метод, вычисляющий значение $x^2 * y^3 * \sqrt{z}$. С его помощью определить, с какой тройкой чисел (a, b, c) или (d, e, f) значение будет максимальным.

9. Написать метод, который у четных чисел меняет знак, а нечетные числа оставляет без изменения. С его помощью обработать ряд чисел от 1 до 10.

10. Написать метод, который положительные числа возводит в квадрат, а отрицательные – в куб. С его помощью обработать ряд чисел от –10 до 10.

11. Написать метод, который вычисляет значения $x = \sin^2(a)$ и $y = \cos^2(a)$. Напечатать таблицу значений от $-\pi$ до π с шагом $\pi/4$.

12. Написать метод, который вычисляет значения $x=a^2$ и $y = \sqrt{a}$. Напечатать таблицу значений от –10 до 10 с шагом 1.

13. Написать метод, который в переданной строке заменяет все точки на многоточие. С его помощью обработать пять разных строк и отобразить их на экране.

14. Написать метод, который в переданной строке заменяет все строчные буквы на заглавные, и наоборот. С его помощью обработать пять разных строк и отобразить их на экране.

15. Написать метод, который разделяет переданную строку на две отдельных строки: первая содержит исходную строку до первой точки, а вторая – исходную строку после первой точки. С его помощью обработать пять разных строк и отобразить результаты на экране.

16. Написать метод, который подсчитывает количество знаков препинания в переданной строке. С его помощью обработать пять разных строк и отобразить результаты на экране.

17. Написать метод, который находит сумму чисел в переданной строке. Числом считается непрерывная последовательность цифр, отделенная от остального текста пробелами или расположенная в начале либо конце строки. Допустимо использовать метод `Split` класса `String`. С помощью этого метода обработать пять разных строк и отобразить результаты на экране.

18. Написать метод, определяющий, является ли переданная строка палиндромом, то есть текстом, который слева направо и справа налево читается одинаково без учета пробелов и регистра символов. С помощью этого метода обработать пять разных строк и отобразить результаты на экране.

19. Написать метод, находящий сумму матриц одинакового размера и возвращающий новую матрицу. С помощью этого метода обработать пары матриц и отобразить результаты на экране.

20. Написать метод, находящий сумму элементов, находящихся не на главной диагонали переданной матрицы. С помощью этого метода обработать пары матриц и отобразить результаты на экране.