

Практическая работа №13

Тема: Разработка приложения с использованием текстовых компонентов

Цель работы: формирование навыков создания приложений Windows Forms, научиться размещать и настраивать внешний вид элементов управления на форме

Задачи:

- изучение приемов создания приложений Windows Forms;
- создание проектов с использованием элементов управления.

Материально-техническое обеспечение:

Место проведения: Компьютерный класс.

Время на выполнение работы: 2 часа.

Оборудование: ПК

Средства обучения: операционная система, текстовый процессор MS Word, программные средства определенного вида

Исходные данные:

1. Конспект занятия.
2. Задание для практической работы.

Перечень справочной литературы:

1) Программирование на языке высокого уровня. Программирование на языке C++: учеб. пособие / Т. И. Немцова, С. Ю. Голова, А. И. Терентьев ; под ред. Л. Г. Гагариной. – М. : ИД «ФОРУМ» : ИНФРА-М, 2018. – 512 с. – (Среднее профессиональное образование).

Краткие теоретические сведения:

Создание Windows-приложений

Система Microsoft Visual Studio содержит удобные средства визуальной разработки Windows-приложений, которые позволяют в интерактивном режиме конструировать программы с графическим интерфейсом, используя готовые компоненты и шаблоны. В процессе разработки выделяют два основных этапа:

Этапы разработки и структура Windows-приложения:

- 1) После запуска MS Visual Studio выбирать **Создание проекта...**
- 2) В окне **Создание проекта** выберите шаблон **Приложение Windows Forms (.NET Framework)** для C#.
(При желании вы можете уточнить условия поиска, чтобы быстро перейти к нужному шаблону. Например, введите *Приложение Windows Forms* в поле поиска. Затем выберите C# в списке языков и Windows в списке платформ.)
- 3) В поле **Имя проекта** окна **Настроить новый проект** введите *НазваниеПроекта* (например, *Form1_Familia*). Затем расположение проекта и решения (папка для работ по МДК) и выберите **Создать**.
- 4) В результате открывается окно с формой в режиме конструктора (Design) (рис. 1). Слева по умолчанию располагается **Панель элементов (Toolbox)**, а справа **Обозреватель решений (Solution Explorer)**. При отсутствии их можно вызвать с помощью меню **Вид (View)**.

Логические переменные и операции над ними

Переменные логического типа описываются посредством служебного слова **bool**. Они могут принимать только два значения – **False** (ложь) и **True** (истина). Результат False (ложь) и True (истина) возникает при использовании операций сравнения > (больше), < (меньше), != (не равно), >= (больше или равно), <= (меньше или равно), == (равно).

Описываются логические переменные следующим образом:

bool b;

В языке C# имеются логические операции, применяемые к переменным логического типа. Это операции *логического отрицания (!)*, *логическое И (&&)* и *логическое ИЛИ (||)*. Операция логического отрицания является унарной операцией. Результат операции **!** есть False, если операнд истинен, и True, если операнд имеет значение «ложь». Так,

!True → False (неправда есть ложь),

!False → True (не ложь есть правда).

Результат операции логическое И (&&) есть истина, только если оба ее операнда истинны, и ложь во всех других случаях. Результат операции логическое ИЛИ (||) есть истина, если какой-либо из ее операндов истинен, и ложен только тогда, когда оба операнда ложны.

Условные операторы

Операторы ветвления позволяют изменить порядок выполнения операторов в программе. К операторам ветвления относятся условный оператор if и оператор выбора switch.

Условный оператор if используется для разветвления процесса обработки данных на два направления. Он может иметь одну из форм: сокращенную или полную.

Форма сокращенного оператора if: **if (B) S;**

где **B** - логическое или арифметическое выражение, истинность которого проверяется; **S** - оператор.

При выполнении сокращенной формы оператора if сначала вычисляется выражение **B**, затем проводится анализ его результата: если **B** истинно, то выполняется оператор **S**; если **B** ложно, то оператор **S** пропускается. Таким образом, с помощью сокращенной формы оператора **f** можно либо выполнить оператор **S**, либо пропустить его.

Форма полного оператора if : **if (B) S1; else S2;**

где **B** - логическое или арифметическое выражение, истинность которого проверяется; **S1, S2**- операторы.

При выполнении полной формы оператора if сначала вычисляется выражение **B**, затем анализируется его результат: если **B** истинно, то выполняется оператор **S1**, а оператор **S2** пропускается; если **B** ложно, то выполняется оператор **S2**, а **S1** - пропускается. Таким образом, с помощью полной формы оператора if можно выбрать одно из двух альтернативных действий процесса обработки данных.

Для примера вычислим значение функции:

$$y(x) = \begin{cases} \sin(x), & \text{если } x \leq a \\ \cos(x), & \text{если } a < x < b \\ \operatorname{tg}(x), & \text{если } x \geq b \end{cases}$$

Указанное выражение может быть запрограммировано в виде

Важное примечание! В С-подобных языках программирования, к которым относится и С#, операция сравнения представляется двумя знаками равенства, например: **if (a == b)**

```
if (x <= a)
    y = Math.Sin(x);
if ((x > a) && (x < b))
    y = Math.Cos(x);
if (x >= b)
    y = Math.Sin(x) / Math.Cos(x);
```

или с помощью оператора else:

```
if (x <= a)
    y = Math.Sin(x);
else
    if (x < b)
        y = Math.Cos(x);
    else
        y = Math.Sin(x) / Math.Cos(x);
```

Оператор выбора **switch** предназначен для разветвления процесса вычислений по нескольким направлениям.

Формат оператора:

switch (<выражение>)

```
{
case <константное_выражение_1>:
    [<оператор 1>];
    <оператор перехода>;
case <константное_выражение_2>:
    [<оператор 2>];
    <оператор перехода>;
...
case <константное_выражение_n>:
    [<оператор n>];
    <оператор перехода>;
[default:
    <оператор>;]
}
```

Замечание. Выражение, записанное в квадратных скобках, является необязательным элементом в операторе switch. Если оно отсутствует, то может отсутствовать и оператор перехода.

Выражение, стоящее за ключевым словом switch, должно иметь арифметический, символьный или строковый тип. Все константные выражения должны иметь разные значения, но их тип должен совпадать с типом выражения, стоящим после switch или приводиться к нему. Ключевое слово case и расположенное после него константное выражение называют также меткой case.

Выполнение оператора начинается с вычисления выражения, расположенного за ключевым словом switch. Полученный результат сравнивается с меткой case. Если результат выражения соответствует метке case, то выполняется оператор, стоящий после этой метки, за которым обязательно должен следовать оператор перехода: break, goto, return и т. д. При использовании оператора break происходит выход из switch и управление передается оператору, следующему за switch. Если же используется оператор goto, то управление передается оператору, помеченному меткой, стоящей после goto. Наконец, оператор return завершает выполнение текущего метода.

Если ни одно выражение case не совпадает со значением оператора switch, управление передается операторам, следующим за необязательной подписью default. Если подписи default нет, то управление передается за пределы оператора switch.

Пример использования оператора **switch**:

```
int dayOfWeek = 5;
switch (dayOfWeek)
{
    case 1:
        MessageBox.Show("Понедельник");
        break;
    case 2:
        MessageBox.Show("Вторник");
        break;
    case 3:
        MessageBox.Show("Среда");
        break;
    case 4:
        MessageBox.Show("Четверг");
        break;
    case 5:
        MessageBox.Show("Пятница");
        break;
    case 6:
        MessageBox.Show("Суббота");
        break;
    case 7:
        MessageBox.Show("Воскресенье");
        break;
    default:
        MessageBox.Show("Неверное значение!");
        break;
}
```

Поскольку на момент выполнения оператора switch в этом примере переменная dayOfWeek равнялась 5, то будут выполнены операторы из блока case 5.

В ряде случаев оператор switch можно заменить несколькими операторами if, однако не всегда такая замена будет легче для чтения. Например, приведенный выше пример можно переписать так:

```
int dayOfWeek = 5;
if (dayOfWeek == 1)
    MessageBox.Show("Понедельник");
else
    if (dayOfWeek == 2)
        MessageBox.Show("Вторник");
    else
        if (dayOfWeek == 3)
            MessageBox.Show("Среда");
        else
            if (dayOfWeek == 4)
                MessageBox.Show("Четверг");
            else
                if (dayOfWeek == 5)
                    MessageBox.Show("Пятница");
                else
                    if (dayOfWeek == 6)
                        MessageBox.Show("Суббота");
                    else
                        if (dayOfWeek == 7)
                            MessageBox.Show("Воскресенье");
                        else
                            MessageBox.Show("Неверное значение!");
```

Кнопки-переключатели

При создании программ в Visual Studio для организации разветвлений часто используются элементы управления в виде кнопок-переключателей (**RadioButton**). Состояние такой кнопки (включено–выключено) визуально отражается на форме, а в программе можно узнать его с помощью свойства **Checked**: если кнопка включена, это свойство будет содержать **True**, в противном случае **False**. Если пользователь выбирает один из вариантов переключателя в группе, все остальные автоматически отключаются.

Группируются радиокнопки с помощью какого-либо контейнера – часто это бывает элемент **GroupBox**. Радиокнопки, размещенные в разных контейнерах, образуют независимые группы.

```
if (radioButton1.Checked)
    MessageBox.Show("Выбрана функция: синус");
else if (radioButton2.Checked)
    MessageBox.Show("Выбрана функция: косинус");
else if (radioButton3.Checked)
    MessageBox.Show("Выбрана функция: экспонента");
```

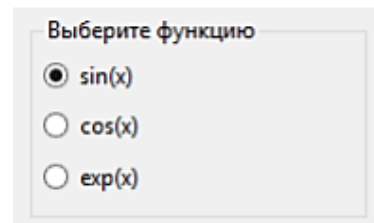


Рис. 1. Группа радиокнопок

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. 2.1). При этом возможен только запуск программы, которая была успешно скомпилирована для предыдущей версии программы.

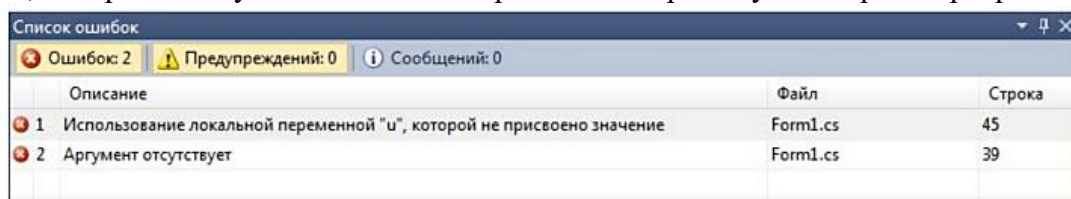



Рис. 2.1. Окно со списком ошибок компиляции


При выявлении ошибок компиляции в нижней части экрана появляется текстовое окно (см. рис. 2.1), 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.

Красный круг  появляется на месте установки точки останова.

Точки останова — это один из самых простых и важных компонентов надежной отладки. Точка останова указывает, где Visual Studio следует приостановить выполнение кода, чтобы вы могли проверить значения переменных или поведение памяти либо выполнение ветви кода.

Нажмите клавишу **F5** или кнопку **Начать отладку** . Запустится приложение, и отладчик перейдет к строке кода, в которой задана точка останова.

В результате чего данная строка будет выделена красным (рис. 2.2).

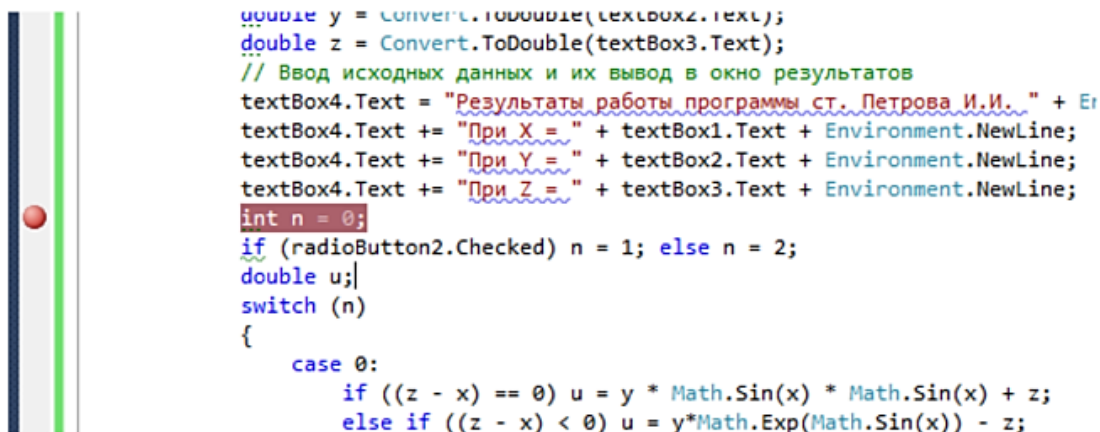
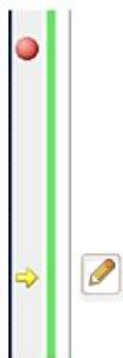


Рис. 2.2. Фрагмент кода с точкой останова

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. 2.3).



```
textBox4.Text += "При Z = " + textBox3.Text + Environment.NewLine;
int n = 0;
if (radioButton2.Checked) n = 1;
else if (radioButton3.Checked) n = 2;
double u;
switch (n)
{
    case 0:
        if ((z - x) == 0) u = y * Math.Sin(x) * Math.Sin(x) + z;
        else if ((z - x) < 0) u = y * Math.Exp(Math.Sin(x)) - z;
        else u = y * Math.Sin(Math.Sin(x)) + z;
        textBox4.Text += "U = " + Convert.ToString(u) + Environment.NewLine;
    }
}
```

Рис. 2.3. 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.

Ход работы:

Требования к содержанию отчета:

- Номер и название практической работы.
- Цель работы.
- По каждой заданию (задаче/примеру) экранные формы (при наличии) и листинг программного кода, показывающие порядок выполнения практической работы, и результаты, полученные в ходе её выполнения.
- Ответы на контрольные вопросы в тетради.

Порядок выполнения работы:

Все проекты практической работы размещать в своей сетевой в новой папке **Пр13_ФИО**

В начале каждого файла проекта установить комментарии: пр.р.№_____ (указать номер), свою Фамилию. Формулировку задания

Задание 1. Создание приложения нахождения значения функции с использованием элементов управления и условного оператора:

Пример 1. Условие задачи: ввести три числа - x , y , z .
Вычислить

$$U = \begin{cases} y \cdot \sin(x) + z, & \text{при } z - x = 0 \\ y \cdot e^{\sin(x)} - z, & \text{при } z - x < 0 \\ y \cdot \sin(\sin(x)) + z, & \text{при } z - x > 0 \end{cases}$$

1. Запустите **MS Visual Studio2019**. Создайте новый проект

Приложение Windows Forms (.NET Framework) для **C#**, имя проекта **pr13_Прил1_Фамилия**.

2. Создайте форму, в соответствии с рис.2.

Разместите на форме элементы **Label**, **TextBox** и **Button**. Поле для вывода результатов также является элементом **TextBox** с установленным в **True** свойством **Multiline** и свойством **ScrollBars** установленным в **Both**.

3. Создание обработчиков событий: выполняется аналогично предыдущим работам.

Примечание: Приложение 1 описание арифметических действий и стандартных функций

Текст обработчика события нажатия на кнопку «Пуск»

```

20 private void button1_Click(object sender, EventArgs e)
21 {
22     // Получение исходных данных из TextBox
23     double x = Convert.ToDouble(textBox1.Text);
24     double y = Convert.ToDouble(textBox2.Text);
25     double z = Convert.ToDouble(textBox3.Text);
26     // Ввод исходных данных в окно результатов
27     textBox4.Text = "Результаты работы программы " + Environment.NewLine + "студента Фамилия И.О. " + Environment.NewLine;
28     textBox4.Text += "При X = " + textBox1.Text + Environment.NewLine;
29     textBox4.Text += "При Y = " + textBox2.Text + Environment.NewLine;
30     textBox4.Text += "При Z = " + textBox3.Text + Environment.NewLine;
31     // Вычисление выражения
32     double u;
33     if ((z-x) == 0)
34         u = y * Math.Sin(x) * Math.Sin(x) + z;
35     else
36         if ((z-x) < 0)
37             u = y * Math.Exp(Math.Sin(x)) - z;
38         else
39             u = y * Math.Sin(Math.Sin(x)) + z;
40     // Вывод результата
41     textBox4.Text += "U = " + u.ToString() + Environment.NewLine;
42 }
43

```

Пояснения: Информация, которая отображается построчно в окне, находится в массиве строк **Lines**, каждая строка которого имеет тип **string**. Однако нельзя напрямую обратиться к этому свойству для добавления новых строк, поскольку размер массивов в C# определяется в момент их инициализации. Для добавления нового элемента используется свойство **Text**, к текущему содержимому которого можно добавить новую строку, например:

textBox4.Text += Environment.NewLine + "Привет";

В этом примере к текущему содержимому окна добавляется символ перевода курсора на новую строку (который может отличаться в разных операционных системах и потому представлен свойством класса **Environment**) и сама новая строка. Если добавляется числовое значение, то его предварительно нужно привести в символьный вид методом **ToString()**.

4. Сохраните все. Протестируйте программу.

5. Самостоятельно запрограммируйте кнопку **Очистить**.

Задание 2. Создайте новое приложение (**pr13_Прил2_Фамилия**) нахождения значения функции в соответствии с вашим вариантом (соответствует номеру списка в журнале):

Индивидуальные задания:

В качестве $f(x)$ использовать по выбору: гиперболическим синус $sh(x)$, x^2 , e^x .

$$\begin{aligned}
 1. \quad a &= \begin{cases} (f(x)+y)^2 - \sqrt{f(x)y}, & xy > 0 \\ (f(x)+y)^2 + \sqrt{|f(x)y|}, & xy < 0 \\ (f(x)+y)^2 + 1, & xy = 0. \end{cases} & 2. \quad b &= \begin{cases} \ln(f(x)) + (f(x)^2 + y)^3, & x/y > 0 \\ \ln|f(x)/y| + (f(x)+y)^3, & x/y < 0 \\ (f(x)^2 + y)^3, & x = 0 \\ 0, & y = 0. \end{cases} \\
 3. \quad c &= \begin{cases} f(x)^2 + y^2 + \sin(y), & x - y = 0 \\ (f(x) - y)^2 + \cos(y), & x - y > 0 \\ (y - f(x))^2 + \operatorname{tg}(y), & x - y < 0. \end{cases} & 4. \quad d &= \begin{cases} (f(x) - y)^3 + \operatorname{arctg}(f(x)), & x > y \\ (y - f(x))^3 + \operatorname{arctg}(f(x)), & y > x \\ (y + f(x))^3 + 0.5, & y = x. \end{cases} \\
 5. \quad e &= \begin{cases} i\sqrt{f(x)}, & i - \text{нечетное}, x > 0 \\ i/2\sqrt{|f(x)|}, & i - \text{четное}, x < 0 \\ \sqrt{|if(x)|}, & \text{иначе.} \end{cases} & 6. \quad g &= \begin{cases} e^{f(x)-|b|}, & 0.5 < x < 10 \\ \sqrt{|f(x)+b|}, & 0.1 < x < 0.5 \\ 2f(x)^2, & \text{иначе.} \end{cases} \\
 7. \quad s &= \begin{cases} e^{f(x)}, & 1 < x < 10 \\ \sqrt{|f(x)+4*b|}, & 12 < x < 40 \\ bf(x)^2, & \text{иначе.} \end{cases} & 8. \quad j &= \begin{cases} \sin(5f(x)+3m|f(x)|), & -1 < m < x \\ \cos(3f(x)+5m|f(x)|), & x > m \\ (f(x)+m)^2, & x = m. \end{cases}
 \end{aligned}$$

$$9. \quad l = \begin{cases} 2f(x)^3 + 3p^2, & x \geq p \\ |f(x) - p|, & 3 \leq x \leq p \\ (f(x) - p)^2, & x = |p|. \end{cases}$$

$$11. \quad m = \frac{\max(f(x), y, z)}{\min(f(x), y)} + 5.$$

$$13. \quad p = \frac{|\min(f(x), y) - \max(y, z)|}{2}.$$

$$15. \quad c = \begin{cases} f(\sin(x))^2 + \sin(f(y)), & x - y = 0 \\ (f(\cos(x))) + \cos(f(y)), & x - y > 0 \\ (y - f(\operatorname{tg}(x)))^2 + \operatorname{tg}(y), & x - y < 0. \end{cases}$$

$$17. \quad c = \begin{cases} f(x)^3 - y^3 \cdot \cos(x), & x + y = 0 \\ (f(x) \cdot y)^2 - \cos(y), & x + y > 0 \\ (y \cdot f(x))^2 + \pi, & x + y < 0. \end{cases}$$

$$19. \quad c = \begin{cases} \sin(f(x)) + \cos(f(y)), & x - y = 0 \\ \operatorname{tg}(f(x + y)), & x - y > 0 \\ \sin^2(f(x)) + \cos^2(f(y)), & x - y < 0. \end{cases}$$

$$21. \quad b = \begin{cases} (f(x) - y)^2 + \sin(x), & x + y = 0 \\ (f(x) - y)^2 + \cos(xy), & x + y > 0 \\ (f(x) + y)^2 - \operatorname{tg}(x), & x + y < 0 \end{cases}$$

$$22. \quad a = \begin{cases} (f(x) \cdot y)^2 + \sin^2(x), & xy = 0 \\ f(x) - (x)^2 + \operatorname{tg}(y - x), & xy > 0 \\ (x)^2 + f(x) - \cos(x), & xy < 0 \end{cases}$$

$$23. \quad c = \begin{cases} \cos(xy) \cdot f(x) + y^3, & x - y = 0 \\ \sin(x) \cdot (f(x) - y)^2, & x - y > 0 \\ \operatorname{ctg}(x) \cdot (f(x))^2 - y, & x - y < 0 \end{cases}$$

$$10. \quad k = \begin{cases} \ln(|f(x)| + |q|), & |xq| > 10 \\ e^{f(x)+q}, & |xq| < 10 \\ f(x) + q, & |xq| = 10 \end{cases}$$

$$12. \quad n = \frac{\min(f(x) + y, y - z)}{\max(f(x), y)}.$$

$$14. \quad q = \frac{\max(f(x) + y + z, xyz)}{\min(f(x) + y + z, xyz)}.$$

$$16. \quad a = \begin{cases} \frac{(ax^2 + 2)}{(x^2 + 1)} f(x), & 1 < |x| < 3, \\ a^2 + f(x), & |x| \geq 3 \\ ax \frac{f(x)}{(x + 2)}, & |x| \leq 1. \end{cases}$$

$$18. \quad k = \begin{cases} \ln(|f(x^2)| + |k|), & |x \cdot k| > 10 \\ \pi^{f(x)+q}, & |x \cdot k| < 10 \\ f(x) - k, & |x \cdot k| = 10 \end{cases}$$

$$20. \quad r = \max(\min(f(x), y), z).$$

$$24. \quad c = \begin{cases} 5\cos(xy) - f(x) + 2y^3, & xy = 0 \\ 2\sin\left(\frac{x}{y}\right) \cdot f(x) - y, & xy > 0 \\ \operatorname{ctg}^3(xy) + (f(3x))^2 - 2y, & xy < 0 \end{cases}$$

$$25. \quad b = \begin{cases} (2f(x) + y)^3 + \operatorname{ctg}(x), & x + y = 0 \\ 5(f(x) - y)^2 - \sin(2y), & x + y > 0 \\ (f(x))^2 + \frac{2x}{y} - \cos(3x), & x + y < 0 \end{cases}$$

Задание 3. Откорректировать полученную форму: с помощью радиокнопок (**RadioButton**) дать пользователю возможность во время работы программы выбрать одну из трех приведенных функций в задании 2 и в соответствии с этим программный код реализации вычисления значения выражения выбранной функции.

Задание 4. Создайте новое приложение (**pr13_Прил3_Фамилия**): Вычислить и вывести на экран таблицу значений функции $y = a \cdot \ln(x)$ при x , изменяющемся от x_0 до x_k с шагом dx , a - константа.

Панель диалога представлена на рис. 3. Текст обработчика нажатия кнопки **Вычислить** приведен ниже.

Примечание: Приложение 2 описание операторов цикла

```

20 private void YButton_Click(object sender, EventArgs e)
21 {
22     // считывание начальных данных
23     double x0 = Convert.ToDouble(textBox1.Text);
24     double xk = Convert.ToDouble(textBox2.Text);
25     double dx = Convert.ToDouble(textBox3.Text);
26     double a = Convert.ToDouble(textBox4.Text);
27     textBox5.Text = "Работу выполнил студент Фамилия И.О." + Environment.NewLine;
28     // цикл для табулирования функции
29     double x = x0;
30     while (x <= (xk + dx / 2))
31     {
32         double y = a * Math.Log(x);
33         textBox5.Text += "x= " + Convert.ToString(x) + "; y= " + Convert.ToString(y) + Environment.NewLine;
34         x = x + dx;
35     }
36 }

```

После отладки программы следует проверить правильность работы программы с помощью контрольного примера (см. рис. 3). Установите точку останова на оператор перед циклом и запустите программу. После попадания на точку останова, выполните пошагово программу и проследите, как меняются все переменные в процессе выполнения.

Задание 5. Создайте новое приложение (**pr13_Прил4_Фамилия**) нахождения значения функции в соответствии с вашим вариантом (соответствует номеру списка в журнале): Составьте программу табулирования функции $y(x)$, выведите на экран значения x и $y(x)$.

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

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) $y = x^2 + \operatorname{tg}(5x + b/x)$,
 $x_0 = -1.5; x_k = -2.5; dx = -0.5;$
 $b = -0.8;$
- 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) $y = 9(x^3 + b^3)\operatorname{tg}x$,
 $x_0 = 1; x_k = 2.2; dx = 0.2;$
 $b = 3.2;$
- 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) $y = a + x^{2/3} \cos(x + e^x)$,
 $x_0 = 5.62; x_k = 15.62; dx = 0.5;$
 $a = 0.41$
- 21) $y = |5x^3|^{1/2} + \cos(78.5x + 3b)$
 $x_0 = -0.25; x_k = -2.25; dx = -0.2$
 $b = 0.46$
- 24) $y = |a - 2x^3| + 4\sin(\frac{2x}{3} + a)$
 $x_0 = -5.02; x_k = -2.78; dx = -0.8$
 $a = 4.2$
- 22) $y = 0.05x^5 + e^{2x} + \sqrt{x^2 + 5a}$
 $x_0 = -2.5; x_k = 1.25; dx = 0.1$
 $a = 3.6$
- 25) $y = |8x^2 - 2a| + 4\ln|\frac{4x^3}{3} + a|$
 $x_0 = -0.82; x_k = -1.82; dx = -0.1$
 $a = -3$
- 23) $y = |b - x^3| + \operatorname{ctg}(8.25x + b)$
 $x_0 = 0.12; x_k = 2.78; dx = 0.26$
 $b = 2.34$

Контрольные вопросы:

- 1) Как создать поле формы с отображением в несколько строк?
- 2) Как осуществляется добавление нового элемента к текущему содержимому?
- 3) Формат оператора switch.
- 1) Приведите примеры стандартных математических функций.

Арифметические действия и стандартные функции

При вычислении выражения, стоящего в правой части оператора присвоения, могут использоваться арифметические операции:

- умножение (\times);
- сложение (+);
- вычитание ($-$);
- деление ($/$);
- остаток от деления (%).

Для задания приоритетов операций могут использоваться круглые скобки (). Также могут использоваться стандартные математические функции, представленные методами класса **Math**:

- **Math.Sin(a)**– синус;
- **Math.Sinh(a)**– гиперболический синус;
- **Math.Cos(a)**– косинус (аргумент задается в радианах);
- **Math.Atan(a)**– арктангенс (аргумент задается в радианах);
- **Math.Log(a)**– натуральный логарифм;
- **Math.Exp(a)**– экспонента;
- **Math.Pow(x, y)**– возводит переменную x в степень y;
- **Math.Sqrt(a)**– квадратный корень;
- **Math.Abs(a)**– модуль числа;
- **Math.Truncate(a)**– целая часть числа;
- **Math.Round(a,i)**– округление числа до i знаков после запятой

В тригонометрических функциях все аргументы задаются в радианах.

Описание данных

Для обеспечения контроля типов все переменные, выражения и значения должны принадлежать к определенному типу. Такого понятия, как **бестиповая переменная**, допустимая в ряде скриптовых языков, в С# вообще не существует. Более того, тип значения определяет те операции, которые разрешается выполнять над ним. Операция, разрешенная для одного типа данных, может оказаться недопустимой для другого.

Целочисленные типы

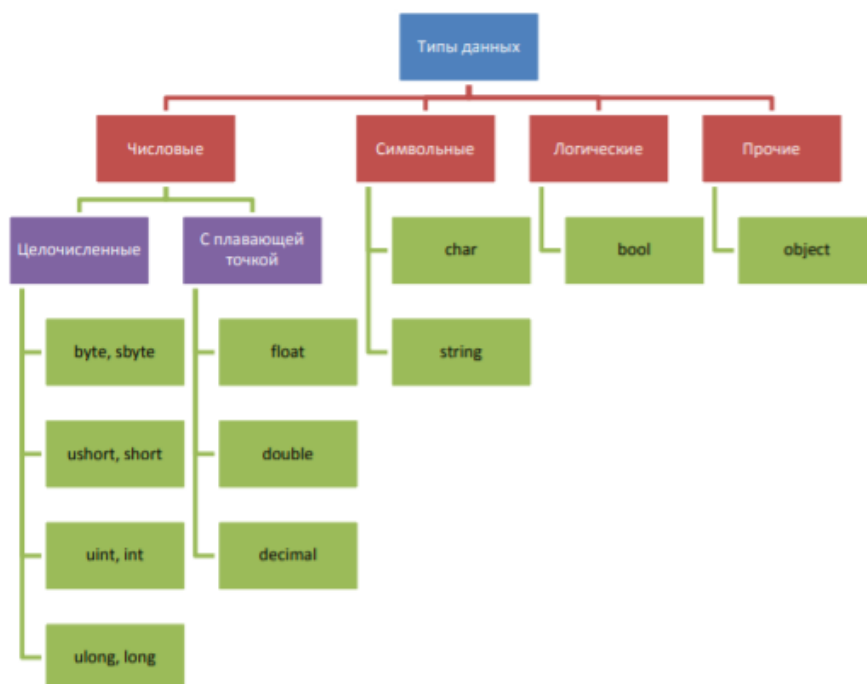
В С# определены девять целочисленных типов: *char*, *byte*, *sbyte*, *short*, *ushort*, *int*, *uint*, *long* и *ulong*. Тип *char* может хранить числа, но чаще используется для представления символов. Остальные восемь целочисленных типов предназначены для числовых расчетов.

Некоторые целочисленные типы могут хранить как положительные, так и отрицательные значения (*sbyte*, *short*, *int* и *long*), другие же – только положительные (*char*, *byte*, *ushort*, *uint* и *ulong*).

Некоторые целочисленные типы могут хранить как положительные, так и отрицательные значения (*sbyte*, *short*, *int* и *long*), другие же – только положительные (*char*, *byte*, *ushort*, *uint* и *ulong*).

Типы с плавающей точкой (вещественные)

Такие типы позволяют представлять числа с дробной частью. В С# имеются три разновидности типов данных с плавающей точкой: *float*, *double* и *decimal*. Первые два типа представляют числовые значения с одинарной и двойной точностью, вычисления над ними выполняются



аппаратно и поэтому быстро. Тип `decimal` служит для представления чисел с плавающей точкой высокой точности без округления, характерного для типов `float` и `double`. Вычисления с использованием этого типа выполняются программно и поэтому более медленны.

Числа, входящие в выражения, C# по умолчанию считает целочисленными. Поэтому следующее выражение будет иметь значение 0, ведь если 1 нацело разделить на 2, то получится как раз 0:

```
double x = 1 / 2;
```

Чтобы этого не происходило, в подобных случаях нужно явно указывать тип чисел с помощью символов-модификаторов: `f` для `float` и `d` для `double`. Приведенный выше пример правильно будет выглядеть так:

```
double x = 1d / 2d;
```

Иногда в программе возникает необходимость записать числа в экспоненциальной форме. Для этого после мантиссы числа записывается символ «e» и сразу после него – порядок. Например, число $2,5 \cdot 10^{-2}$ будет записано в программе следующим образом:

2.5e-2

Символьные типы

В C# символы представлены не 8-разрядным кодом, как во многих других языках программирования, а 16-разрядным кодом, который называется юникодом (Unicode). В юникоде набор символов представлен настолько широко, что он охватывает символы практически из всех естественных языков на свете.

Основным типом при работе со строками является тип `string`, задающий строки переменной длины. Тип `string` представляет последовательность из нуля или более символов в кодировке Юникод. По сути, текст хранится в виде последовательной доступной только для чтения коллекции объектов `char`.

Логический тип данных

Тип `bool` представляет два логических значения: «истина» и «ложь». Эти логические значения обозначаются в C# зарезервированными словами `true` и `false` соответственно. Следовательно, переменная или выражение типа `bool` будет принимать одно из этих логических значений.

Операторы организации циклов в C#

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 закливания, 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
    }
}
```

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 `в`: если оно истинно, то управление передается оператору `s`, если ложно – цикл завершается, и управление передается на оператор, следующий за условием `в`. Так как условие `в` 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 закливания в случае, если условие `в` всегда будет оставаться истинным.

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 пара метром 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.