МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)
Кафедра программной инженерии

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«_9_» _09_ 2024г.

Языки программирования

Методические указания по выполнению практических работ для студентов, обучающихся по направлению 02.03.03 Математическое обеспечение и администрирование информационных систем

Курск 2024

2

УДК 681.3(075)

Составитель Л. А. Лисицин

Рецензент

Кандидат технических наук, доцент *Халин Ю.А.*

Языки-программирования [Текст]: методические указания по выполнению практических работ по направлению 02.03.03 Математическое обеспечение и администрирование информационных систем / Юго-Зап. гос. ун-т; сост.: Л. А. Лисицин. Курск, 2024. 91 с.: ил. 4. табл. 3. Библиогр. с. 91.

Содержат основные теоретические положения и сведения, необходимые для выполнения практических работ по программированию на языке C#.

Методические указания по структуре, содержанию и стилю изложения материала соответствуют методическим и научным требованиям, предъявляемым к учебным и методическим пособиям. Отражен порядок выполнения практических работ и правила оформления отчетов.

Методические указания предназначены для студентов, обучающихся по направлению 02.03.03 Математическое обеспечение и администрирование информационных систем.

Текст печатается в авторской редакции

Подписано в печать 9.09. Формат 60x84 1/16.
Усл.печ. л. 5,5 Уч.-изд. л. 5,1. Тираж 100 экз. Заказ 584 Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

Оглавление

ВВЕДЕНИЕ

Основной целью проведения практических занятий является формирование умений и навыков по программированию с использованием современных технологий, включая объектно-ориентированное программирование.

Проведению практических занятий предшествует самостоятельная работа студентов, направленная на ознакомление с соответствующим теоретическим материалом. При необходимости, студенты по заданиям преподавателей выполняют подготовительную работу, обеспечивающую более эффективный процесс закрепления умений и наыков.

Практические занятия проводятся в специализированном классе, оснащенным персональными ЭВМ не ниже Intel i3, с операционной системой не ниже Windows 7.

При проведении практического занятия рекомендуется сочетание интерактивного и практического обучения.

Контроль умений и навыков, приобретаемых на практических занятиях осуществляется в форме собеседования.

Содержание практических занятий и объем в часах на каждую тему приведены в таблице 1.

Таблица 1- Практические занятия

№ п/п	Наименование практического занятия	Объем, час.
1.	Изучение массивов и разнородные структуры данных	4
2.	Объекто-ориентированное программирование: классы, объекты и методы	4

3.	Объекто-ориентированное программирование: абстрактные классы и интерфейсы	4
4.	Динамические структуры данных	4
5.	GUI, обработка событий	2
Итого		18

Практическое занятие №1

Изучение массивов и разнородных структур данных

1.1 Цель работы: изучить и получить практические навыки проектирования программ с использованием массивов данных, а также способов хранения

1.2. Краткие теоретические сведения

Массив – это структурированный тип данных, состоящий из фиксированного числа упорядоченных по индексу элементов, имеющих один и тот же тип.

Если в качестве типа элемента одномерного массива берется другой массив, то образуется структура, называемая двумерным массивом. Для описания типа элементов может быть использован любой объявленный тип. *Двумерный массив* **(**или матрица) представляет собой прямоугольную таблицу элементов одинакового типа.

Пример:

```
int mas[3][5];
```

В данном примере объявлен двумерный массив (*mas*.). Массив *mas* представляет собой матрицу, состоящую из 3 строк и 5 столбцов.

Доступ к значениям элементов двумерного массива обеспечивается через индексы, каждый из которых заключается в квадратные скобки. Например, **mas[3][2]** – значение элемента, лежащего на пересечении 4-й строки и 3-го столбца, так как индексы начинаются с нуля.

По такому принципу можно описать не только двумерные, но и nмерные массивы. Размерность массивов ограничивает только объем памяти конкретной ЭВМ.

Пример описания трехмерных массивов a и b:

```
typedef int mes[3];
typedef mes ves[5];
typedef ves res[8]; res
a; int b[3][5][8];
```

Для описания многомерных массивов можно использовать предварительно определенные константы:

```
const  N=10, M=15;
int  mas[N][M];
```

Если многомерный массив инициализируется при его объявлении, список значений по каждой размерности заключается в фигурные скобки. Объявление трехмерного массива A3 размерностью 4 на 3 на 2 имеет вид

```
int A3[4][3][2]={{{0, 1}, {2, 3}, {4, 5}},
                 {{6, 7}, {8, 9}, {10, 11}},
                 {{12, 13}, {14, 15}, {16, 17}},
                 {{18, 19}, {20, 21}, {22, 23}} };
```

Для работы с двумерными массивами обычно используются вложенные циклы (рисунок 1). Часто применяется вложенная структура алгоритма с модификаторами.

Рисунок 1 – Схема алгоритма вложенных циклов

Обнуление элементов двумерного массива можно выполнить, используя вложенный оператор for: for (int k=0; k<3; k++) for (int m=0; m <5; m ++) a[k][m]=0;

Ввод и вывод значений элементов двумерного массива с клавиатуры можно осуществить с помощью компоненты TStringGrid страницы Additional, предназначенной для создания таблиц, в ячейках которых располагаются произвольные текстовые строки. С помощью компоненты TEdit, представляющей собой однострочный редактор текста, можно вводить и/или отображать достаточно длинные текстовые строки.

Пример:

```
Edit5->Text=StringGrid1->Cells[i][j];
```

/* позволяет вывести в окно содержимое ячейки, где i – номер столбца; j – номер строки. */

После выполнения присваивания в Edit5->Text хранится содержимое (текст) ячейки таблицы.

StringGrid1->Cells[i][j] определяет содержимое ячейки с табличными координатами (i, j).

Динамическое размещение многомерных массивов

Многомерные массивы можно размещать в динамической памяти с помощью операций **new** и **delete**. Например, операторы const N=3, M=5;

```
int **ary;
   ary = new int * [N];    // массив указателей
   for (int i = 0; i <N; i++) {
      ary[i] = new int [M];     // инициализация указателей
   }
```

объявляют массив с именем ary, содержащий 3 строки и 5 столбцов. Чтобы освободить память, можно использовать операторы

```
for (int i = 0; i <N; i++) {        delete [] ary[i];
   }
   delete [] ary;
```

Пример программирования с использованием двумерного массива

Задание. Подсчитайте сумму всех элементов в двумерном массиве a[4][5].

1. На рисунке 2 – разработка алгоритма:

   – входные данные: a – массив, состоящий из вещественных чисел;

   – выходные данные: sum – вещественная переменная, сумма всехэлементов массива;

   – промежуточные данные: i, j – счетчики циклов.

Рисунок 2 – Схема алгоритма вычисления суммы элементов массива

При вычислении суммы элементов массива в начале алгоритма переменную, хранящую сумму, необходимо обнулить.

При вычислении произведения элементов массива в начале алгоритма переменной, хранящей произведение, необходимо присвоить единицу.

2. Разработка формы – рисунок 24, таблица 13.

Рисунок 24 – Внешний вид формы

Таблица 13

Используемые компоненты

Имя компонента	Страница палитры компонент	Настраиваемые свойства	Значение
1. Form1	–	Caption	Лабораторная работа № 6
2. Label1	Standard	Caption	Введите элементы матрицы
3. Label2	Standard	Caption	Сумма элементов равна
4. Label3	Standard	Caption	
5. Button1	Standard	Caption	Рассчет
6. StringGrid1	Additional	FixedCols	0
		RowCount	4
		FixedRows	0
		Options	[goEditing, goTabs]

Текст программы:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{ const m=4, n=5;
typedef double
mas[m][n];
int i, j;
mas a;
double sum;   for(i=0;i<m;i++)
for(j=0;j<n;j++)
```

```
a[i][j]=StrToFloat(StringGrid1-
>Cells[j][i]);  sum=0;  for(i=0;i<m;i++)
for(j=0;j<n;j++)  sum+=a[i][j];
 Label3->Caption=FloatToStr(sum);
}
        //---------------------------------------------------------------------
```

Индивидуальные задания

При выполнении заданий первого уровня сложности рекомендуется использовать статический массив, а для заданий второго уровня сложности – динамический массив.

Задачи первого уровня сложности

1. Дана вещественная матрица размером 4 строки, 5 столбцов. Переставляя ее строки и столбцы, добейтесь того, чтобы наибольший элемент (один из них) оказался в верхнем левом углу.
2. Определите, является ли заданная целочисленная квадратная матрица порядка 5 симметричной относительно главной диагонали.
3. Дана вещественная матрица размером 4 строки, 5 столбцов. Поменяйте местами максимальный и минимальный элементы матрицы.
4. Дана целочисленная квадратная матрица порядка 5. Найдите максимальный элемент среди элементов, лежащих ниже главной диагонали, и максимальный элемент среди элементов, лежащих выше главной диагонали, поменяйте их местами.
5. Дана целочисленная квадратная матрица порядка 5. Найдите максимальный элемент среди элементов, лежащих левее вспомогательной диагонали, и максимальный элемент среди элементов, лежащих правее вспомогательной диагонали, поменяйте их местами.
6. Среди строк целочисленной квадратной матрицы порядка 5 найдите строку с минимальной суммой элементов.

7. Дана целочисленная квадратная матрица порядка 5. Найдите минимальный элемент среди элементов, лежащих ниже главной диагонали, и максимальный элемент среди элементов, лежащих выше главной диагонали, вычислите их среднее арифметическое.

8. Дана целочисленная квадратная матрица порядка 5. Найдите минимальный элемент среди элементов, лежащих левее вспомогательной диагонали, и максимальный элемент среди элементов, лежащих правее вспомогательной диагонали, и вычислить их среднее геометрическое.

9.Среди столбцов целочисленной квадратной матрицы порядка 5 найдите столбец с максимальной суммой элементов.

10.Среди тех столбцов целочисленной матрицы размером 3 строки, 5 столбцов, которые содержат только такие элементы, значения которых по модулю не превышают 10, найдите столбец с минимальным произведением элементов.

11.Даны целые числа $a_1,...,a_{10}$, целочисленная квадратная матрица порядка 5. Замените нулями в матрице те элементы, для которых имеются равные числа среди $a_1,...,a_{10}$.

12.В двумерном целочисленном массиве размером 4 строки, 5 столбцов поменяйте местами строки, симметричные относительно середины массива (горизонтальной линии).

13.В двумерном целочисленном массиве размером 4 строки, 5 столбцов поменяйте местами столбцы, симметричные относительно середины массива (вертикальной линии).

14.Даны две вещественные квадратные матрицы порядка 4. Получите новую матрицу прибавлением к элементам каждого столбца первой матрицы минимального элемента соответствующего столбца второй матрицы.

15.В целочисленной квадратной матрице порядка 4 найдите наибольший по модулю элемент. Получите квадратную матрицу порядка 3 путем выбрасывания из исходной матрицы строки и столбца, на пересечении которых расположен элемент с найденным значением.

16.В данной вещественной квадратной матрице порядка 5 найдите наименьший элемент. Получите квадратную матрицу порядка

4 путем выбрасывания из исходной матрицы строки и столбца, на пересечении которых расположен элемент с найденным значением.

17.Дана вещественная матрица размером 4 строки, 5 столбцов. Получите новую матрицу путем вычитания из всех элементов данной матрицы наименьшего по модулю элемента.

18. Дана целочисленная квадратная матрица порядка 4. Найдите в каждой строке наибольший элемент и поменяйте его местами с элементом, расположенным на главной диагонали.

19. Дана целочисленная квадратная матрица порядка 5. Найдите в каждой строке наименьший элемент и поменяйте его местами с элементом, расположенным на вспомогательной диагонали.

20. Дана вещественная матрица размером 4 строки, 5 столбцов. Определите числа b_1,..,b_4, равные значениям средних арифметических элементов строк.

21. Дана вещественная матрица размером 4 строки, 5 столбцов. Определите числа b_1,..,b_5, равные значениям средних арифметических элементов столбцов.

22. Дана вещественная матрица размером 4 строки, 5 столбцов. Определите числа b_1,..,b_5, равные среднему арифметическому значению максимального и минимального элементов каждого столбца.

23. В данной вещественной квадратной матрице порядка 5 найдите сумму элементов строки, в которой расположен элемент с наименьшим значением. Предполагается, что такой элемент единственный.

24. В данной вещественной квадратной матрице порядка 5 найдите среднее геометрическое положительных элементов столбца, в котором расположен элемент с наименьшим значением. Предполагается, что такой элемент единственный.

25. Дана целочисленная квадратная матрица порядка 4. Для каждой строки найдите минимальный элемент или максимальный в зависимости от условия: если на главной диагонали находится элемент больше элемента, расположенного на вспомогательной диагонали, то найдите минимальный элемент, иначе – максимальный.

26. Дана целочисленная квадратная матрица порядка 4. Ниже главной диагонали найдите минимальный элемент среди четных элементов, а выше главной диагонали среди нечетных элементов.

27. Дана целочисленная квадратная матрица порядка 4. Ниже главной диагонали найдите минимальный элемент среди положительных элементов, а выше главной диагонали – максимальный элемент среди отрицательных элементов.

28. Дана целочисленная квадратная матрица порядка 5. Левее вспомогательной диагонали найдите минимальный элемент среди четных элементов, а правее вспомогательной диагонали среди нечетных элементов.

29. Дана целочисленная квадратная матрица порядка 5. Левее вспомогательной диагонали найдите минимальный элемент среди положительных элементов, а правее вспомогательной диагонали – максимальный элемент среди отрицательных элементов.

30. Дана целочисленная квадратная матрица порядка 5. Для каждого столбца найдите минимальный элемент или максимальный в зависимости от условия: если на главной диагонали находится элемент меньше элемента, расположенного на вспомогательной диагонали, то найдите минимальный элемент, иначе – максимальный.

Задачи второго уровня сложности

1. Дана вещественная матрица размером n строк, m столбцов. Для каждой строки найдите сумму элементов и упорядочьте строки матрицы по возрастанию сумм.

2. Дана вещественная матрица размером n строк, m столбцов. Для каждого столбца найдите сумму элементов и упорядочьте столбцы матрицы по возрастанию сумм.

3. Дана целочисленная матрица размером n строк, m столбцов. Для каждого столбца найдите сумму нечетных элементов и упорядочьте столбцы матрицы по возрастанию сумм.

4. Дана целочисленная матрица размером n строк, m столбцов. Для каждого столбца найдите сумму положительных элементов и упорядочте столбцы матрицы по возрастанию сумм.

5. Дана целочисленная матрица размером n строк, m столбцов. Для каждого столбца найдите произведение элементов, значения которых лежат в диапазоне [1, 10], упорядочьте столбцы матрицы по убыванию произведений.

6. Дана целочисленная матрица размером n строк, m столбцов. Для каждого столбца найдите наибольшие элементы и упорядочьте столбцы матрицы по убыванию наибольших элементов.

7. Дана вещественная матрица размером n строк, m столбцов. Упорядочьте ее столбцы по возрастанию их первых элементов.

8. Дана целочисленная матрица размером n строк, m столбцов. Найдите и выведите строку, в которой длина максимальной серии одинаковых элементов максимальна.

9. Дана целочисленная матрица размером n строк, m столбцов. Найдите и выведите столбец, в котором длина максимальной серии возрастающих по значению элементов максимальна.

10. Дана вещественная матрица размером n строк, m столбцов. Для каждого столбца найдите среднее арифметическое элементов и упорядочьте столбцы матрицы по возрастанию среднего арифметического элементов.

11. Дана вещественная матрица размером n строк, m столбцов. Для каждой строки найдите среднее арифметическое элементов и упорядочьте строки матрицы по возрастанию среднего арифметического элементов.

12. Дана вещественная матрица размером n строк, m столбцов. Для каждого столбца найдите среднее геометрическое положительных элементов и упорядочьте столбцы матрицы по возрастанию среднего геометрического элементов.

13. Дана вещественная матрица размером n строк, m столбцов. Для каждой строки найдите среднее геометрическое положительных элементов и упорядочьте строки матрицы по убыванию среднего геометрического элементов.

14. Дана вещественная матрица размером n строк, m столбцов. Найдите строку, содержащую максимальный элемент, и упорядочьте эту строки матрицы по возрастанию значений элементов.

15. Дана вещественная матрица размером n строк, m столбцов. Найдите строку, содержащую минимальный элемент, и упорядочьте эту строки матрицы по убыванию значений элементов.

16. Дана вещественная матрица размером n строк, m столбцов. Найдите столбец, содержащий минимальный элемент, и упорядочьте этот столбец матрицы по убыванию значений элементов.

17. Дана вещественная матрица размером n строк, m столбцов. Найдите столбец, содержащий максимальный элемент, и упорядочьте этот столбец матрицы по возрастанию значений элементов.

18. Дана целочисленная матрица размером n строк, m столбцов. Найдите столбец, сумма элементов которого минимальна, и упорядочьте этот столбец матрицы по убыванию значений элементов.

19. Дана целочисленная матрица размером n строк, m столбцов. Найдите строку, сумма элементов которой максимальна, и упорядочьте эту строку матрицы по убыванию значений элементов.

20. Дана целочисленная матрица размером n строк, m столбцов. Найдите строку, среднее арифметическое элементов которой максимально, и упорядочьте эту строку матрицы по возрастанию значений элементов.

21. Дана целочисленная матрица размером n строк, m столбцов. Найдите столбец, среднее арифметическое элементов которого максимально, и упорядочьте этот столбец матрицы по возрастанию значений элементов.

22. Дана целочисленная матрица размером n строк, m столбцов. Найдите строку, среднее геометрическое элементов которой максимально, и упорядочьте эту строку матрицы по убыванию значений элементов.

23. Дана целочисленная матрица размером n строк, m столбцов. Найдите столбец, среднее геометрическое элементов которого минимально, и упорядочьте этот столбец матрицы по убыванию значений элементов.

24. Дана целочисленная матрица размером n строк, m столбцов. Найдите строку, произведение элементов которой максимально, и упорядочьте эту строку матрицы по убыванию значений элементов.

25. Дана целочисленная матрица размером n строк, m столбцов. Найдите столбец, произведение элементов которого минимально, и упорядочьте этот столбец матрицы по убыванию значений элементов.

26. Дана целочисленная матрица размером n строк, m столбцов. Найдите строку, количество различных элементов в которой максимально, и упорядочьте эту строку матрицы по возрастанию значений элементов.

Контрольные вопросы к защите лабораторной работы

1. Напишите фрагмент программы ввода двумерного целочисленного массива, в котором 5 строк и 10 столбцов.
2. Напишите фрагмент программы вывода двумерного вещественного массива, в котором 5 строк и 6 столбцов.
3. Дайте описание трехмерного целочисленного массива.
4. Дайте описание четырехмерного целочисленного массива в качестве переменной и в качестве типа.

5.Опишите динамические массивы:

float a[5][8], b[7][5];

6. Допустимы ли в ***C++*** операции над этими массивами как над единым целым?

7. Введите квадратную вещественную матрицу 4-го порядка,элементы которой заданы построчно, и распечатайте ее по столбцам.

8. Присвойте нулевые значения всем элементам массива:

float a[10][5];

9. Для чего предназначается компонент TStringGrid?
10. Какими свойствами обладает компонент TStringGrid?

Содержание отчета

Отчет по лабораторной работе включает:

– титульный лист;
– условие задания;
– алгоритм решения задачи;
– текст программы;
– результаты тестирования программы.

Практическая работа №2

Объектно-ориентированное программирование: классы, объекты и методы

1.1. Цель работы: изучить основы объектно-ориентированного программирования, классов, объектов, методов, приватных, защищенных и публичных членов классов

1.2. Краткие теоретические сведения

C# является полноценным объектно-ориентированным языком. Это значит, что программу на C# можно представить в виде взаимосвязанных взаимодействующих между собой объектов.

Описанием объекта является класс, а объект представляет экземпляр этого класса. Можно еще провести следующую аналогию. У нас у всех есть некоторое представление о человеке - наличие двух рук, двух ног, головы, пищеварительной, нервной системы, головного мозга и т.д. Есть некоторый шаблон - этот шаблон можно назвать классом. Реально же существующий человек (фактически экземпляр данного класса) является объектом этого класса. Класс определяется с помощью ключевого слова class:

```
class Book
{

}
```

Вся функциональность класса представлена его членами - полями (полями называются переменные класса), свойствами, методами, событиями. В прошлой главе мы создали структуру Book. Теперь переделаем ее в класс Book:

```
class Book
{
        public string name;     public string author;     public int year;
     public void Info()
    {
       Console.WriteLine("Книга '{0}' (автор {1}) была издана в {2}
году", name, author, year);
    }
}
```

Кроме обычных методов в классах используются также и специальные методы, которые называются конструкторами. Конструкторы вызываются при создании нового объекта данного класса. Отличительной чертой конструктора является то, что его название должно совпадать с названием класса:

```
class Book
{
        public string name;     public string author;
    public int year;

        public Book()
    { }      public Book(string name, string author, int year)
    {        this.name = name;        this.author = author;        this.year = year;
    }

    public void Info()
    {
       Console.WriteLine("Книга '{0}' (автор {1}) была издана в {2}
году", name, author, year);
    }
}
        }
```

Одно из назначений конструктора - начальная инициализация членов класса. В данном случае мы использовали два конструктора. Один пустой. Второй конструктор наполняет поля класса начальными значениями, которые передаются через его параметры.

Поскольку имена параметров и имена полей (name, author, year) в данном случае совпадают, то мы используем ключевое слово this. Это ключевое слово представляет ссылку на текущий экземпляр класса. Поэтому в выражении this.name = name; первая часть this.name означает, что name - это поле текущего класса, а не название параметра name. Если бы у нас параметры и поля назывались по-разному, то использовать слово this было бы необязательно.

Теперь используем класс в программе. Создадим новый проект. Затем нажмем правой кнопкой мыши на название проекта в окне Solution Explorer (Обозреватель решений) и в появившемся меню выберем пункт Class.

В появившемся диалоговом окне дадим новому классу имя Book и нажмем кнопку Add (Добавить). В проект будет добавлен новый файл Book.cs, содержащий класс Book.

Изменим в этом файле код класса Book на следующий:

```
class Book
{
    public string name;     public string author;
   public int year;

    public Book()
   {
            name = "неизвестно";
      author = "неизвестно";        year = 0;
    }
    public Book(string name, string author, int year)
```

```
        {       this.name = name;       this.author = author;       this.year =
year;
        }

        public void GetInformation()
        {
            Console.WriteLine("Книга '{0}' (автор {1}) была издана в {2}
году", name, author, year);
        }
    }
```

Теперь перейдем к коду файла *Program.cs* и изменим метод Main класса Program следующим образом:

```
    class Program
    {
        static void Main(string[] args)
        {
            Book b1 = new Book("Война и мир", "Л. Н. Толстой",
    1869);      b1.GetInformation();

            Book b2 = new Book();       b2.GetInformation();

            Console.ReadLine();
        } }
```

Если мы запустим код на выполнение, то консоль выведет нам информацию о книгах b1 и b2. Обратите внимание, что чтобы создать новый объект с использованием конструктора, нам надо использовать ключевое слово new. Оператор new создает объект класса и выделяет для него область в памяти.

Инициализаторы объектов

Для нашего класса Book мы могли бы установить последовательно значения для всех трех полей класса:

```
Book b1 = new Book();
b1.name = "Война и мир";
b1.author = "Л. Н. Толстой";
```

```
b1.year = 1869;
 b1.GetInformation();
```

Но можно также использовать **инициализатор объектов**:

```
Book b2 = new Book { name = "Отцы и дети", author = "И. С. Тургенев", year = 1862 }; b2.GetInformation();
```

С помощью инициализатора объектов можно присваивать значения всем доступным полям и свойствам объекта в момент создания без явного вызова конструктора.

ЗАДАНИЕ

Разработать класс, инкапсулирующий двумерный массив. Класс должен содержать поля и методы, необходимые для реализации приведенного ниже задания к ЛР. Номер варианта задания соответствует порядковому номеру студента с списке группы.

Варианты заданий

Вариант 1

Дана вещественная матрица размером 4 строки, 5 столбцов. Переставляя ее строки и столбцы, добейтесь того, чтобы наибольший элемент (один из них) оказался в верхнем левом углу.

Вариант 2

Определите, является ли заданная целочисленная квадратная матрица порядка 5 симметричной относительно главной диагонали.

Вариант 3

Дана вещественная матрица размером 4 строки, 5 столбцов. Поменяйте местами максимальный и минимальный элементы матрицы.

Вариант 4

Дана целочисленная квадратная матрица порядка 5. Найдите максимальный элемент среди элементов, лежащих ниже главной диагонали, и максимальный элемент среди элементов, лежащих выше главной диагонали, поменяйте их местами.

Вариант 5

Дана целочисленная квадратная матрица порядка 5. Найдите максимальный элемент среди элементов, лежащих левее вспомогательной диагонали, и максимальный элемент среди элементов, лежащих правее вспомогательной диагонали, поменяйте их местами.

Вариант 6

Среди строк целочисленной квадратной матрицы порядка 5 найдите строку с минимальной суммой элементов.

Вариант 7

Дана целочисленная квадратная матрица порядка 5. Найдите минимальный элемент среди элементов, лежащих ниже главной диагонали, и максимальный элемент среди элементов, лежащих выше главной диагонали, вычислите их среднее арифметическое.

Вариант 8

Дана целочисленная квадратная матрица порядка 5. Найдите минимальный элемент среди элементов, лежащих левее вспомогательной диагонали, и максимальный элемент среди элементов, лежащих правее вспомогательной диагонали, и вычислить их среднее геометрическое.

Вариант 9

Среди столбцов целочисленной квадратной матрицы порядка 5 найдите столбец с максимальной суммой элементов.

Вариант 10

Среди тех столбцов целочисленной матрицы размером 3 строки, 5 столбцов, которые содержат только такие элементы, значения которых по модулю не превышают 10, найдите столбец с минимальным произведением элементов.

Вариант 11

Даны целые числа $a_1,...,a_{10}$, целочисленная квадратная матрица порядка 5. Замените нулями в матрице те элементы, для которых имеются равные числа среди $a_1,...,a_{10}$.

Вариант 12

В двумерном целочисленном массиве размером 4 строки, 5 столбцов поменяйте местами строки, симметричные относительно середины массива (горизонтальной линии).

Вариант 13

В двумерном целочисленном массиве размером 4 строки, 5 столбцов поменяйте местами столбцы, симметричные относительно середины массива (вертикальной линии).

Вариант 14

Даны две вещественные квадратные матрицы порядка 4. Получите новую матрицу прибавлением к элементам каждого столбца первой матрицы минимального элемента соответствующего столбца второй матрицы.

Вариант 15

В целочисленной квадратной матрице порядка 4 найдите наибольший по модулю элемент. Получите квадратную матрицу порядка 3 путем выбрасывания из исходной матрицы строки и столбца, на пересечении которых расположен элемент с найденным значением.

Вариант 16

В данной вещественной квадратной матрице порядка 5 найдите наименьший элемент. Получите квадратную матрицу порядка 4 путем выбрасывания из исходной матрицы строки и столбца, на пересечении которых расположен элемент с найденным значением.

2.4 Содержание отчета

1. Титульный лист
2. Задание на лабораторную работу
3. Листинг программы
4. Скриншот работы приложения

Практическая работа № 3

Объектно-ориентированное программирование: абстрактные классы и интерфейсы

3.1. Цель работы: Расширенные возможности объектно-ориентированного программирования в C#: абстрактные классы, полиморфизм, интерфейсы.

3.2. Краткие теоретические сведения:

Кроме обычных классов в C# есть абстрактные классы. Абстрактный класс похож на обычный класс. Он также может иметь переменные, методы, конструкторы, свойства. Единственное, что при определении абстрактных классов используется ключевое слово abstract:

```
abstract class Human
{
   public int Length { get; set; }
   public double Weight { get; set; }
}
```

Но главное отличие состоит в том, что мы не можем использовать конструктор абстрактного класса для создания его объекта. Например, следующим образом:

```
Human h = new
Human();
```

Зачем нужны абстрактные классы? Допустим, в нашей программе для банковского сектора мы можем определить две основных сущности: клиента банка и сотрудника банка. Каждая из этих сущностей будет отличаться, например, для сотрудника надо определить его должность, а для клиента - сумму на счете. Соответственно клиент и сотрудник будут составлять отдельные классы Client и Employee. В то же время обе этих сущности могут иметь что-то общее, например, имя и фамилию, какую-то другую общую функциональность. И эту общую функциональность лучше вынести в какой-то отдельный класс,

например, Person, который описывает человека. То есть классы Employee (сотрудник) и Client (клиент банка) будут производными от класса Person. И так как все объекты в нашей системе будут представлять либо сотрудника банка, либо клиента, то напрямую мы от класса Person создавать объекты не будем. Поэтому имеет смысл сделать его абстрактным:

```
abstract class Person
{
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public Person(string name, string surname)
        {
        FirstName = name;
        LastName = surname;
}
  public void Display()
   {
      Console.WriteLine(FirstName + " " +
LastName);
   }
}
class Client : Person
{
   public int Sum { get; set; }    // сумма
на счету
    public Client(string name, string surname,
int sum)
       : base(name, surname)
   {
```

```
        Sum = sum;
    }
}
 class Employee : Person
{
    public string Position { get; set; } //
должность
    public Employee(string name, string
surname, string position)
        : base(name, surname)
    {
            Position = position;
    }
}
```

Затем мы можем использовать эти классы:

```
Person client = new Client(“Tom”, “Smith”,500);
Person employee = new Employee ( (“Tomson”, “Bob”,”Операционист”);
```

Но мы НЕ можем создать объект Person, используя конструктор класса Person:

```
Person person = new
Person
(“Bill”,”Gates”);
```

Однако несмотря на то, что напрямую мы не можем вызвать конструктор класса Person для создания объекта, тем не менее конструктор в абстрактных классах то же может играть важную роль. В частности, в классе Person конструктор инициализирует свойства FirstName и LastName. И хотя напрямую он не вызывается, тем не менее производные классы Client и Employee могут обращаться к нему.

Используя механизм наследования, мы можем дополнять и переопределять общий функционал базовых классах в классах-наследниках. Однако напрямую мы можем наследовать только от одного класса, в отличие, например, от языка C++, где имеется множественное наследование.

В языке C# подобную проблему позволяют решить интерфейсы. Они играют важную роль в системе ООП. Интерфейсы позволяют определить некоторый функционал, не имеющий конкретной реализации. Затем этот функционал реализуют классы, применяющие данные интерфейсы.

Для определения интерфейса используется ключевое слово interface. Как правило, названия интерфейсов в C# начинаются с заглавной буквы I, например, IComparable, IEnumerable (так называемая венгерская нотация), однако это не обязательное требование, а больше стиль программирования. Интерфейсы также, как и классы, могут содержать свойства, методы и события, только без конкретной реализации.

Определим следующий интерфейс IAccount, который будет содержать методы и свойства для работы со счетом клиента. Для добавления интерфейса в проект можно нажать правой кнопкой мыши на проект и в появившемся контекстном меню выбрать Add-> New Item и в диалоговом окне добавления нового компонента выбрать (Рисунок 4)

Interface:

Рисунок 4 - Контекстное меню добавления нового компонента

Изменим пустой код интерфейса IAccount на следующий:

```
interface IAccount
{
        // Текущая сумма на счету
        int CurrentSum { get; }
    // Положить деньги на счет
```

```
    void Put(int sum);
        // Взять со счета
        void Withdraw(int sum);
}
```

У интерфейса методы и свойства не имеют реализации, в этом они сближаются с абстрактными методами абстрактных классов. Сущность данного интерфейса проста: он определяет свойство для текущей суммы денег на счете и два метода для добавления денег на счет и изъятия денег.

Еще один момент в объявлении интерфейса: все его члены - методы и свойства не имеют модификаторов доступа, но фактически по умолчанию доступ public, так как цель интерфейса - определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации.

3.3. Задание для практической работы.

Напишите программу для создания абстрактного класса с именем Equation, который содержит пустой метод с именем solution(). На основе этого класса создайте подкласс LinearEquation, реализующий решение линейных уравнений

3.4 Содержание отчета

1. Титульный лист
2. Задание на лабораторную работу
3. Листинг программы
4. Скриншот работы приложения

Практическая работа №4

«Объектно-ориентированное программирование в C#: наследование и полиморфизм»

4.1. Цель работы: Наследование и полиморфизм в языке C#.

4.2. Краткие теоретические сведения

Наследование (inheritance) является одним из ключевых моментов ООП. Благодаря наследованию один класс может унаследовать функциональность другого класса.

Пусть у нас есть следующий класс Person, описывающий отдельного человека:

```
class  Person:
{
        private string firstName:
        private string lastName:
        public string lastName:
{
                get { return firstName; }
                set  { firstName = value; }
}
public string LastName
{
                get { return lastName; }
                set  { lastName = value; }
}
public void Display()
}
                Console.Writeline($"{FirstName }
```

```
{LastName}”);
}
```

Но вдруг нам потребовался класс, описывающий сотрудника предприятия - класс Employee. Поскольку этот класс будет реализовывать тот же функционал, что и класс Person, так как сотрудник - это также и человек, то было бы рационально сделать класс Employee производным (или наследником, или подклассом) от класса Person, который, в свою очередь, называется базовым классом или родителем (или суперклассом):

```
class Employee :
Person
{
}
```

После двоеточия мы указываем базовый класс для данного класса. Для класса Employee базовым является Person, и поэтому класс Employee наследует все те же свойства, методы, поля, которые есть в классе Person. Единственное, что не передается при наследовании, это конструкторы базового класса.

Таким образом, наследование реализует отношение isa (является), объект класса Employee также является объектом класса Person: static void Main(string[]

```
{
   Person p = new Person { FirstName = "Bill",
LastName = "Gates" };
        p.Display();
        p = new Employee { FirstName = "Denis",
LastName = "Ritchi" };
        p.Display();
```

```
        Console.Read();
}
```

И поскольку объект Employee является также и объектом Person, то мы можем так определить переменную: Person p = new Employee().

Все классы по умолчанию могут наследоваться. Однако здесь есть ряд ограничений:

•Не поддерживается множественное наследование, класс может наследоваться только от одного класса. Хотя проблема множественного наследования реализуется с помощью концепции интерфейсов, о которых мы поговорим позже.

•При создании производного класса надо учитывать тип доступа к базовому классу - тип доступа к производному классу должен быть таким же, как и у базового класса, или более строгим. То есть, если базовый класс у нас имеет тип доступа internal, то производный класс может иметь тип доступа internal или private, но не public.

•Если класс объявлен с модификатором sealed, то от этого класса нельзя наследовать и создавать производные классы. Например, следующий класс не допускает создание наследников:

```
sealed class
Admin
{
}
```

Доступ к членам базового класса из класса-наследника

Вернемся к нашим классам Person и Employee. Хотя Employee наследует весь функционал от класса Person, посмотрим, что будет в следующем случае:

```
Employee : Person
{
```

```
        public void Display()
        {
      Console.WriteLine(_f irstName);
  }
}
```

Этот код не сработает и выдаст ошибку, так как переменная _firstName объявлена с модификатором privateи поэтому к ней доступ имеет только класс Person. Но зато в классе Person определено общедоступное свойство FirstName, которое мы можем использовать, поэтому следующий код у нас будет работать нормально:

```
class Employee : Person
{
        public void Display()
        {
      Console.WriteLine(FirstName);
   }
}
```

Таким образом, производный класс может иметь доступ только к тем членам базового класса, которые определены с модификаторами public, internal, protected и protected internal.

Ключевое слово base

Теперь добавим в наши классы конструкторы:

```
class Person
{
        public string FirstName { get;  set; }
        public string LastName {get; set; }

   public Person(string firstName, string lastName)
```

```
        {
        FirstName = firstName;
        LastName = lastName;
    }
        public void Display()
   {
      Console.WriteLine($"{FirstName} {LastName}");
   }
}

class Employee : Person
{
        public string Company { get; set; }
        public Employee(string firstName, string
lastName, string comp)
        : base(firstName, lastName)
   {
      Company = comp;
   }
}
```

Класс Person имеет стандартный конструктор, который устанавливает два свойства. Поскольку класс Employee наследует и устанавливает те же свойства, что и класс Person, то логично было бы не писать по сто раз код установки, а как-то вызвать соответствующий код класса Person. К тому же свойств, которые надо установить, и параметров может быть гораздо больше.

С помощью ключевого слова base мы можем обратиться к базовому классу. В нашем случае в конструкторе класса Employee нам надо установить имя, фамилию и компанию. Но имя и фамилию мы передаем на установку в

конструктор базового класса, то есть в конструктор класса Person, с помощью выражения base(fName, lName).

```
Main(string[] args)
{
        Person p = new Person("Bill", "Gates");
        p.Display();
    Employee emp = new Employee ("Tom", "Simpson", "Microsoft");
    emp.Display();
    Console.Read();
}
```

ЗАДАНИЕ

Разработать объектно-ориентированную программу-редактор, позволяющую выводить заданный текст в графической рамке. Рамка отрисовывается символьными примитивами. Предусмотреть минимум 3 типа рамок, реализовать родительский класс для абстрактной рамки, конкретные рамки унаследовать от родительского класса.

1. Разработать объектно-ориентированную программу для конвертации чисел в двоичной системе счисления в десятичную и обратно. Преобразование систем счисления реализовать самостоятельно. Реализовать механизм наследования классов.
2. Разработать объектно-ориентированную программу для конвертации чисел в шестнадцатиричной системе счисления в десятичную и обратно. Преобразование систем счисления реализовать самостоятельно. Реализовать механизм наследования классов.

3. Разработать объектно-ориентированную программу для конвертации чисел в двоичной системе счисления в шестнадцатиричную и обратно. Преобразование систем счисления реализовать самостоятельно. Реализовать механизм наследования классов.

4. Разработать объектно-ориентированную программу для базовой расстановки кораблей на поле для игры «Морской бой». Предусмотреть 4 типа кораблей: однопалубный, двухпалубный, трехпалубный и четырехпалубный. Корабли не должны соприкасаться между собой даже углами. Для каждого корабля реализовать метод, определяющий, касается ли он другого (заданного) корабля на поле. Реализовать следующие методы проверки: потоплен корабли или нет; подбит или нет. Реализовать механизм наследования классов.

5. Реализовать объектно-ориентированную программу для формирования расстановки мин на поле для игры «Сапер». Реализовать механизм наследования классов.

6. Реализовать объектно-ориентированную программу — редактор информации о сотрудниках предприятия. Сотрудники делятся на рядовых сотрудников и руководителей. Для каждого руководителя определен список подчиненных. Программа должна работать в двух режимах: отображение данных о сотруднике без возможности редактирования и редактирование данных. Для каждого режима работы создать отдельные классы. Реализовать механизм наследования классов.

7. Реализовать объектно-ориентированную программу — редактор информации о студентах вуза. Студенты организованы в группы. У каждой группы имеется список студентов и староста. Программа должна работать в двух режимах: отображение данных о студенте /группе без возможности редактирования и редактирование данных.

Для каждого режима работы создать отдельные классы.

Реализовать механизм наследования классов.

8. Реализовать объектно-ориентированную программу для отображения на двумерном графике различных классов точек.

Каждый класс точек отображать разными графическими примитивами разного цвета (например, красные круги, желтые треугольники и т. д.). Реализовать механизм наследования классов.

9. Создать класс Point — точка. На его основе создать классы ColoredPoint и Line. На основе класса Line создать класс ColoredLine и класс PolyLine — многоугольник. Все классы должны иметь методы для установки и получения значений всех координат, а также изменения цвета и получения текущего цвета. Написать демонстрационную программу, в которой будет использоваться список объектов этих классов в динамической памяти.

10. Создать класс Vehicle. На его основе реализовать классы Plane, Car и Ship. Классы должны иметь возможность задавать и получать координаты, параметры средств передвижения (цена, скорость, год выпуска). Для самолета должна быть определена высота, для самолета и корабля — количество пассажиров. Для корабля — порт приписки. Написать программу, создающую список объектов этих классов в динамической памяти и демонстрирующую функционал разработанных классов.

11. Реализовать объектно-ориентированный справочник планет Солнечной системы. Справочник должен включать само Солнце, планеты и их спутники. Для каждой планеты указать наименование, фото, массу и радиус, удаленность от Солнца (от родительской планеты — в случае спутников). Для спутников указать родительскую планету. Реализовать механизм наследования классов.

12. Разработать объектно-ориентированную программу для представления дробей: десятичных, обыкновенных и периодических.

Предусмотреть конвертацию десятичных и периодических в обыкновенные дроби; обыкновенных в десятичные или периодические. Реализовать механизм наследования классов.

13. Разработать объектно-ориентированную программу для представления и вывода на экран комплексных и экспоненциальных чисел. Реализовать механизм наследования классов.

14. Разработать объектно-ориентированную программу для представления и вывода на экран верхних и нижних индексов символа. Реализовать механизм наследования классов.

15. Разработать объектно-ориентированную программу для базовой расстановки фигур на шахматной доске. Для каждой фигуры запрограммировать правила выполнения ходов. Реализовать механизм наследования классов.

4.4. Содержание отчета

1. Титульный лист
2. Задание на лабораторную работу
3. Листинг программы с динамическим массивом
4. Скриншот экрана

Практическая работа №5

GUI. Элементы оформления программного интерфеса WindowForm.

5.1. Цель работы: Введение в разработку графического интерфейса пользователя

5.2 Краткие теоретические сведения

Графический пользовательский интерфейс (GUI) представляет собой удобный для пользователя механизм взаимодействия с приложением. Графический интерфейс (произносится как «GOO-ее») дает приложению отличительный «внешний вид». Графические интерфейсы создаются из компонентов GUI. Иногда они называются элементами управления или виджетами - для оконных гаджетов. Компонент GUI - это объект, с которым пользователь взаимодействует с помощью мыши, клавиатуры или другой формы ввода. Многие IDE (программы, которые позволяют графический интерфейс проектирования) предоставляют инструменты проектирования графического интерфейса, с помощью которых вы можете указать точный размер и местоположение компонента визуально. IDE генерирует для вас код GUI. Хотя это значительно упрощает создание графических интерфейсов, вы не можете понять все свойства и события компонентов. По этой причине мы написали GUI-код руками.

Элементы оформления предназначены для создания программного интерфейса пользователя, обеспечивают управление программным и интерфейсом и удобство использования.

Создание меню MenuStrip

Для создания меню в Windows Forms применяется элемент **MenuStrip**.

Основные свойства элемента меню:

Свойство **Dock –** прикрепляет меню к одной из сторон формы;

Свойство **LayoutStyle –** задает ориентацию панели меню на форме, может принимать следующие значения:

- HorizontalStackWithOverflow: расположение по горизонтали с переполнением - если длина меню превышает длину контейнера, то новые элементы, выходящие за границы контейнера, не отображаются, то есть панель переполняется элементами
- StackWithOverflow: элементы располагаются автоматически с переполнением
- VerticalStackWithOverflow: элементы располагаются вертикально с переполнением
- Flow: элементы размещаются автоматически, но без переполнения - если длина панели меню меньше длины контейнера, то выходящие за границы элементы переносятся
- Table: элементы позиционируются в виде таблицы.

Свойство **ShowItemToolTips –** указывает, будут ли отображаться всплывающие подсказки для отдельных элементов меню;

Свойство **Stretch –** позволяет растянуть панель по всей длине контейнера;

Свойство **TextDirection –** задает направление текста в пунктах меню; Свойство **MenuStrip –** является контейнером для отдельных пунктов меню, которые представлены объектом **ToolStripMenuItem**.

Добавить новые элементы в меню можно в режиме дизайнера:

Для добавления доступно три вида элементов: MenuItem (объект ToolStripMenuItem), ComboBox и TextBox.

Таким образом, в меню можно использовать выпадающие списки и текстовые поля, однако, как правило, эти элементы применяются в основном на панели инструментов. Меню же обычно содержит набор объектов ToolStripMenuItem.

Рисунок 1- MenuStrip.

Также мы можем добавить пункты меню программно в коде C#:

```
public partial class Form1 : Form
{
   public Form1()
   {
      InitializeComponent();

     ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");
  //  ToolStripMenuItem в конструкторе принимает текстовую метку, которая будет использоваться в качестве текста меню "Файл"

      fileItem.DropDownItems.Add("Создать");
fileItem.DropDownItems.Add(new ToolStripMenuItem("Сохранить"));  // Каждый объект ToolStripMenuItem имеет коллекцию DropDownItems, которая хранит дочерние объекты ToolStripMenuItem.

      menuStrip1.Items.Add(fileItem);
 // Добавление к меню созданного фрагмента

  ToolStripMenuItem aboutItem = new ToolStripMenuItem("О программе");  aboutItem.Click += aboutItem_Click;
      menuStrip1.Items.Add(aboutItem);
   }
   //Назначив обработчики для события Click, можно обработать нажатия на пункты меню: aboutItem.Click += aboutItem_Click
```

```
    void aboutItem_Click(object sender, EventArgs e)
    {
       MessageBox.Show("О программе");
    }
}
```

Если передать при добавление строку текста, то для нее неявным образом будет создан объект

ToolStripMenuItem: fileItem.DropDownItems.Add("Создать")

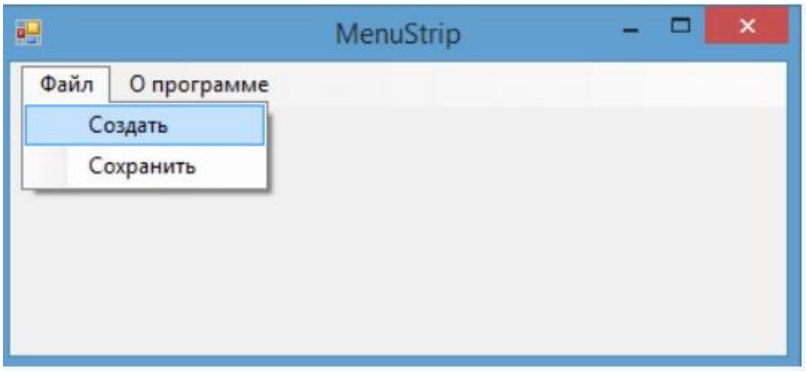

Рисунок 2- fileItem.DropDownItems.Add("Создать")

Отметки пунктов меню

Свойство CheckOnClick при значении true позволяет на клику отметить пункт меню. А с помощью свойства Checked можно установить, будет ли пункт меню отмечен при запуске программы.

Еще одно свойство CheckState возвращает состояние пункта меню - отмечен он или нет. Оно может принимать три значения: Checked (отмечен), Unchecked (неотмечен) и Indeterminate (в неопределенном состоянии)

Например, создадим ряд отмеченных пунктов меню и обработаем событие установки / снятия отметки:

```
public partial class Form1 : Form
{
   public Form1()
   {
      InitializeComponent();
```

```
      ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");

      ToolStripMenuItem newItem = new ToolStripMenuItem("Создать")
{ Checked = true, CheckOnClick = true };
      fileItem.DropDownItems.Add(newItem);

       ToolStripMenuItem saveItem = new
ToolStripMenuItem("Сохранить")
 { Checked = true, CheckOnClick = true };
      saveItem.CheckedChanged += menuItem_CheckedChanged;

      fileItem.DropDownItems.Add(saveItem);
menuStrip1.Items.Add(fileItem);
   }
    void menuItem_CheckedChanged(object sender, EventArgs e)
   {
      ToolStripMenuItem menuItem = sender as
ToolStripMenuItem;          if (menuItem.CheckState ==
CheckState.Checked)          MessageBox.Show("Отмечен");
      else if (menuItem.CheckState == CheckState.Unchecked)
MessageBox.Show("Отметка снята");
   }}
```

Клавиши быстрого доступа.

Если нам надо быстро обратиться к какому-то пункту меню, то мы можем использовать клавиши быстрого доступа. Для задания клавиш быстрого доступа используется свойство **ShortcutKeys**:

```
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();

         ToolStripMenuItem fileItem = new ToolStripMenuItem("Файл");

         ToolStripMenuItem saveItem = new
   ToolStripMenuItem("Сохранить")
```

```
 { Checked = true, CheckOnClick = true };
saveItem.Click+=saveItem_Click;
      saveItem.ShortcutKeys = Keys.Control | Keys.P;

      fileItem.DropDownItems.Add(saveItem);
menuStrip1.Items.Add(fileItem);
   }
    void saveItem_Click(object sender, EventArgs e)
   {
      MessageBox.Show("Сохранение");
   }
}
```

Клавиши задаются с помощью перечисления Keys. В данном случае по нажатию на комбинацию клавиш Ctrl + P, будет срабатывать нажатие на пункт меню "Сохранить".

С помощью изображений мы можем разнообразить внешний вид пунктов меню. Для этого мы можем использовать следующие свойства:

DisplayStyle: определяет, будет ли отображаться на элементе текст, или изображение, или и то и другое.

Image: указывает на само изображение

ImageAlign: устанавливает выравнивание изображения относительно элемента

ImageScaling: указывает, будет ли изображение растягиваться, чтобы заполнить все пространство элемента

ImageTransparentColor: указывает, будет ли цвет изображения прозрачным

Если изображение для пункта меню устанавливает в режиме дизайнера, то нам надо выбрать в окне свойство пункт Image, после чего откроется окно для импорта ресурса изображения в проект.

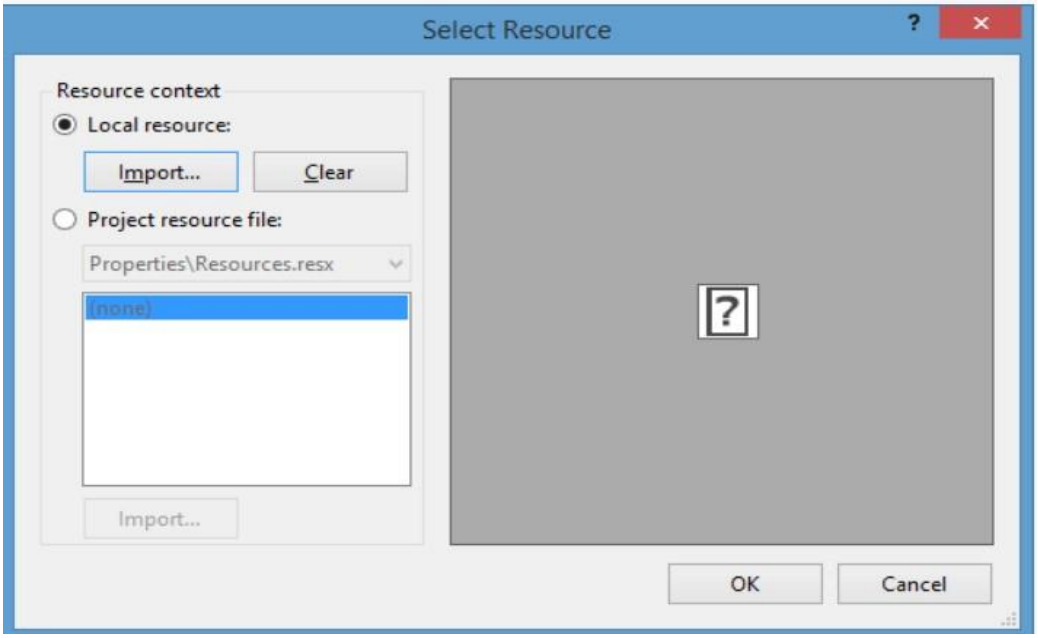

Рисунок 3- окно свойства пункта Image.

Чтобы указать, как разместить изображение, у свойства DisplayStyle надо установить значение Image. Если мы хотим, чтобы кнопка отображала только текст, то надо указать значение Text, либо можно комбинировать два значения с помощью другого значения ImageAndText. По умолчанию изображение размещается слева от текста:

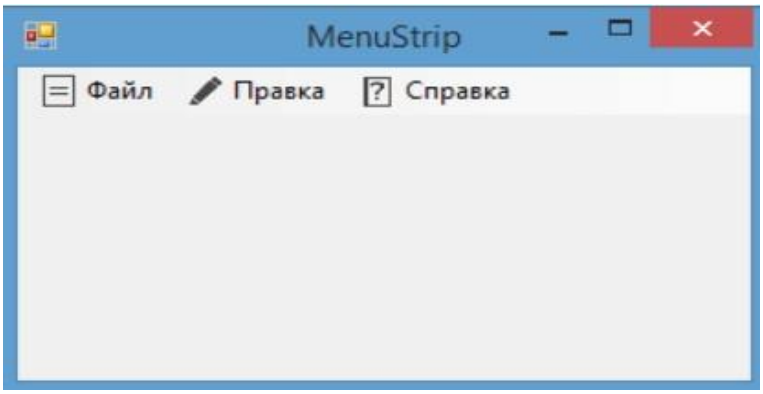

Рисунок 4- Menu ImageAndText..

Также можно установить изображение динамически в коде:
fileToolStripMenuItem.Image = Image.FromFile

(@"D:\Icons\0023\block32.png");

Панель состояния ContextMenuStrip

Элемент **ContextMenuStrip** позволяет создать контекстное меню. Данный компонент во многом аналогичен элементы MenuStrip за тем исключением, что контекстное меню не может использоваться само по себе, оно обязательно применяется к какому-нибудь другому элементу, например, текстовому полю.

Новые элементы в контекстное меню можно добавить в режиме дизайнера:

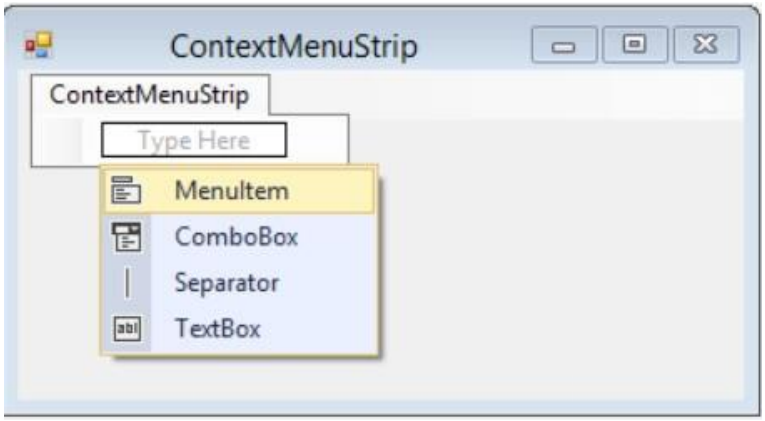

Рисунок 5- контекстное меню.

При этом мы можем добавить все те же элементы, что и в MenuStrip. Но, как правило, использует ToolStripMenuItem, либо элемент
ToolStripSeparator, представляющий горизонтальную полоску разделитель между другими пунктами меню.

Либо на панели свойств можно обратиться к свойству Items компонента ContextMenuStrip и в открывшемся окне добавить и настроить все элементы меню:

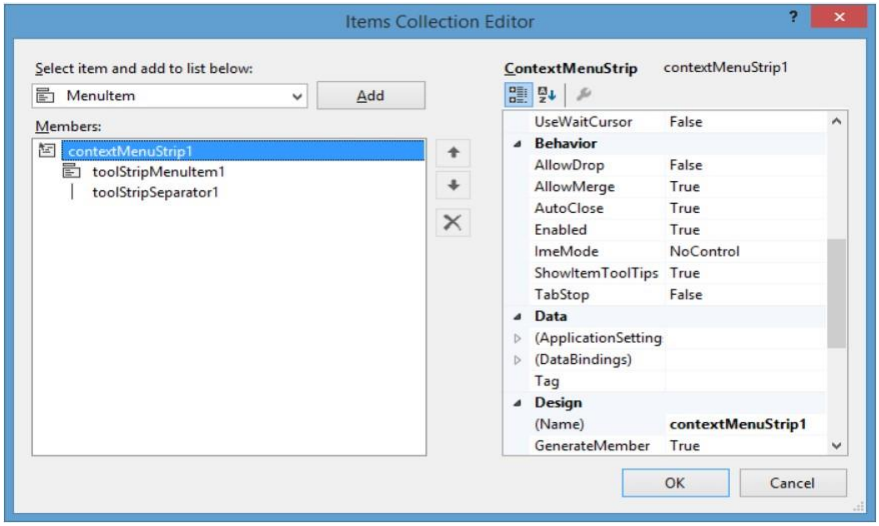

Рисунок 6- свойство Items компонента

Теперь создадим небольшую программу. Добавим на форму элементы ContextMenuStrip и TextBox, которые будут иметь названия contextMenuStrip1 и textBox1 соответственно. Затем изменим код формы следующим образом:

```
public partial class Form1 : Form
{
string
buffer;
   public Form1()
   {
      InitializeComponent();

      textBox1.Multiline = true;
      textBox1.Dock = DockStyle.Fill;

      // создаем элементы меню
      ToolStripMenuItem copyMenuItem = new
ToolStripMenuItem("Копировать");
      ToolStripMenuItem pasteMenuItem = new
ToolStripMenuItem("Вставить");
      // добавляем элементы в меню
      contextMenuStrip1.Items.AddRange(new[] { copyMenuItem,
pasteMenuItem })
      // ассоциируем контекстное меню с текстовым
полем textBox1.ContextMenuStrip =
contextMenuStrip1; // устанавливаем
обработчики событий для меню
copyMenuItem.Click += copyMenuItem_Click;
pasteMenuItem.Click += pasteMenuItem_Click;
   }
   // вставка текста
   void pasteMenuItem_Click(object sender, EventArgs e)
   {
      textBox1.Paste(buffer);
   }
   // копирование текста
   void copyMenuItem_Click(object sender, EventArgs e)
   {
```

```
        // если выделен текст в текстовом поле, то копируем его в
буфер        buffer = textBox1.SelectedText;
    }
}
```

В данном случае выполнена простейшая реализация функциональности copy-paste. В меню добавляется два элемента. А у текстового поля устанавливается многострочность, и оно растягивается по ширине контейнера.

У многих компонентов есть свойство ContextMenuStrip, которое позволяет ассоциировать контекстное меню с данным элементом. В случае с TextBox ассоциация происходит следующим образом: textBox1.ContextMenuStrip = contextMenuStrip1.

И по нажатию на текстовое поле правой кнопкой мыши мы сможем вызвать ассоциированное контекстное меню.

С помощью обработчиков нажатия пунктов меню устанавливаются действия по копированию и вставке строк.

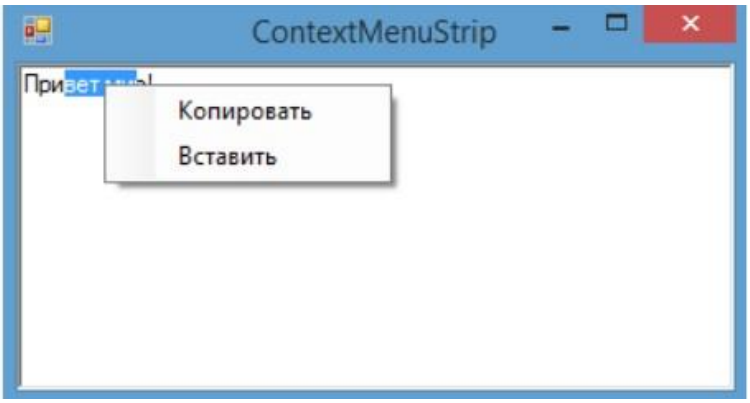

Рисунок 7- вставка строк.

Панель состояния StatusStrip

Элемент **StatusStrip** представляет строку состояния, которая предназначена для отображения текущей информации о состоянии работы приложения.

При добавлении на форму StatusStrip автоматически размещается в нижней части окна приложения (как и в большинстве приложений).

Однако при необходимости мы сможем его иначе позиционировать, управляя свойством Dock, которое может принимать следующие значения:

Bottom: размещение внизу (значение по умолчанию)

Top: прикрепляет статусную строку к верхней части формы

Fill: растягивает на всю форму

Left: размещение в левой части формы

Right: размещение в правой части формы

None: произвольное положение

StatusStrip может содержать различные элементы. В режиме дизайнера мы можем добавить следующие типы элементов:

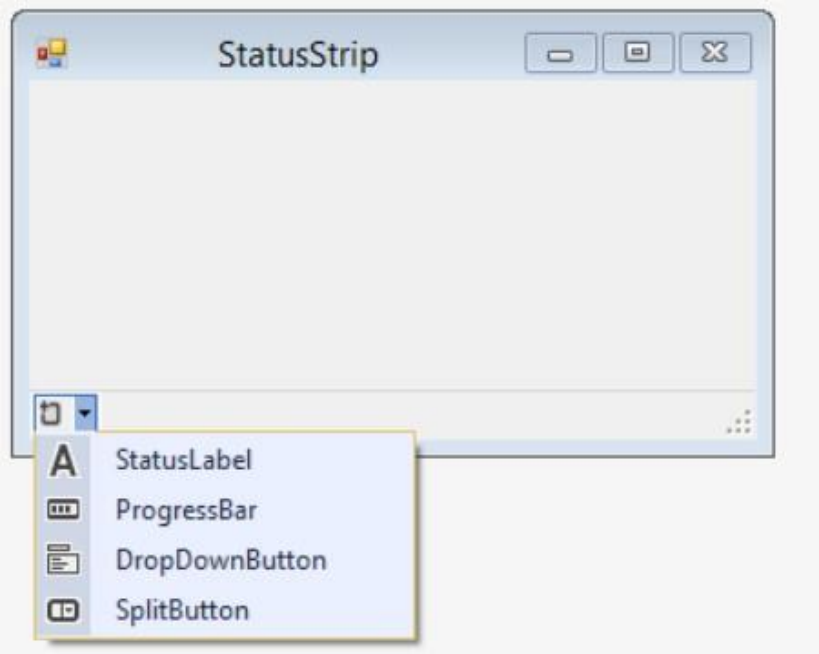

Рисунок 8- Menu StatusStrip

StatusLabel: метка для вывода текстовой информации. Представляет объект ToolStripLabel ProgressBar: индикатор прогресса. Представляет объект

ToolStripProgressBar

DropDownButton: кнопка с выпадающим списком по клику. Представляет объект ToolStripDropDownButton

SplitButton: еще одна кнопка, во многом аналогичная DropDownButton.

Представляет объект ToolStripSplitButton

Либо можно обратиться на панели свойств к свойству Items компонента StatusStrip и открывшемся окне добавить и настроить все элементы:

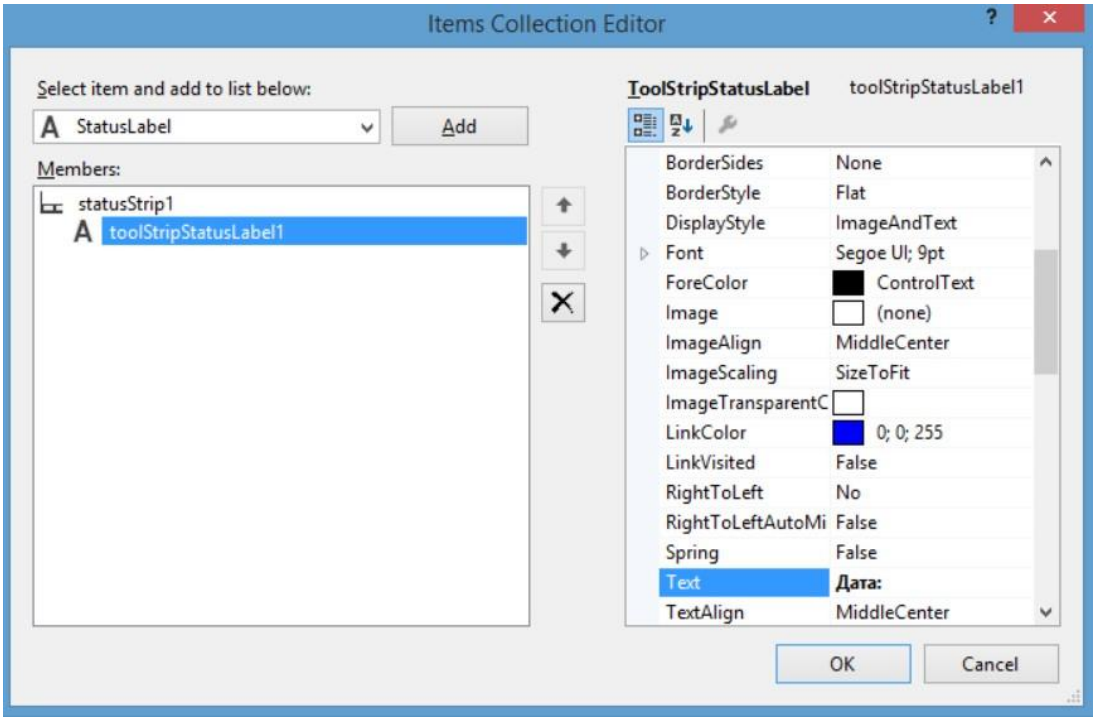

Рисунок 9- панель свойств к свойству Items

Также мы можем добавить элементы программно. Создадим небольшую программу. Определим следующий код формы:

```
public partial class Form1 : Form
{
   ToolStripLabel dateLabel;
   ToolStripLabel timeLabel;
   ToolStripLabel infoLabel;
   Timer timer;
   public Form1()
   {
      InitializeComponent();

      infoLabel = new ToolStripLabel();
infoLabel.Text = "Текущие дата и время:";
dateLabel = new ToolStripLabel();
      timeLabel = new ToolStripLabel();

      statusStrip1.Items.Add(infoLabel);
statusStrip1.Items.Add(dateLabel);
```

```
        statusStrip1.Items.Add(timeLabel);

        timer = new Timer() { Interval = 1000 };
    timer.Tick += timer_Tick;
        timer.Start();
    }
     void timer_Tick(object sender, EventArgs e)
    {
        dateLabel.Text = DateTime.Now.ToLongDateString();
    timeLabel.Text = DateTime.Now.ToLongTimeString();
    } } Здесь создаются три метки на строке
    состояния и таймер.
```

После создания формы таймер запускается, и срабатывает его событие Tick, в обработчике которого устанавливаем текст меток.

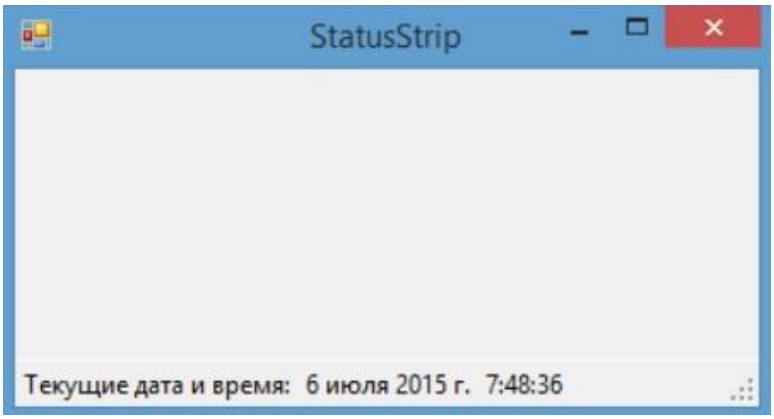

Рисунок 10- строка состояния и таймера

Панель инструментов ToolStrip

Элемент ToolStrip представляет панель инструментов. Каждый отдельный элемент на этой панели является объектом **ToolStripItem**.

Ключевые свойства компонента ToolStrip связаны с его позиционированием на форме:

- Dock: прикрепляет панель инструментов к одной из сторон формы
- LayoutStyle: задает ориентацию панели на форме (горизонтальная, вертикальная, табличная)

- ShowItemToolTips: указывает, будут ли отображаться всплывающие подсказки для отдельных элементов панели инструментов
- Stretch: позволяет растянуть панель по всей длине контейнера
- В зависимости от значения свойства LayoutStyle панель инструментов может располагаться по горизонтали, или в табличном виде:
- HorizontalStackWithOverflow: расположение по горизонтали с переполнением - если длина панели превышает длину контейнера, то новые элементы, выходящие за границы контейнера, не отображаются, то есть панель переполняется элементами
- StackWithOverflow: элементы располагаются автоматически с переполнением
- VerticalStackWithOverflow: элементы располагаются вертикально с переполнением
- Flow: элементы располагаются автоматически, но без переполнения - если длина панели меньше длины контейнера, то выходящие за границы элементы переносятся, а панель инструментов растягивается, чтобы вместить все элементы
- Table: элементы позиционируются в виде таблицы.

Если LayoutStyle имеет значения HorizontalStackWithOverflow / VerticalStackWithOverflow, то с помощью свойства **CanOverflow** мы можем задать поведение при переполнении. Так, если это свойство равно true (значение по умолчанию), то для элементов, не попадающих в границы ToolStrip, создается выпадающий список:

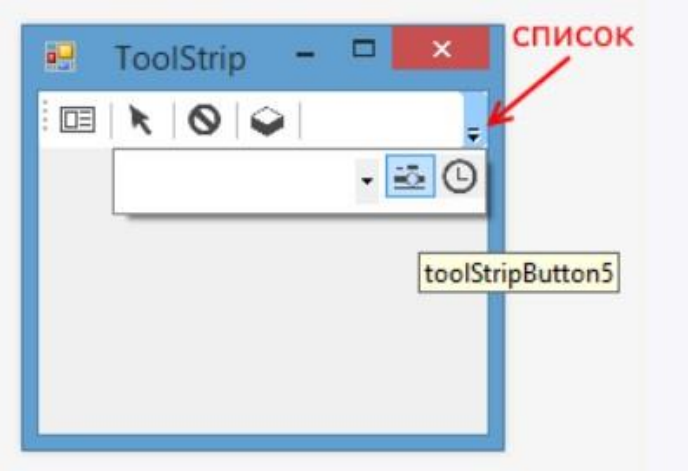

Рисунок 11- Menu свойства **CanOverflow**

При значении false подобный выпадающий список не создается.

Типы элементов панели и их добавление

Панель ToolStrip может содержать объекты следующих классов

- **ToolStripLabel**: текстовая метка на панели инструментов, представляет функциональность элементов Label и LinkLabel
- **ToolStripButton**: аналогичен элементу Button.Также имеет событие

  Click, с помощью которого можно обработать нажатие пользователя на кнопку
- **ToolStripSeparator**: визуальный разделитель между другими элементами на панели инструментов
- **ToolStripToolStripComboBox**: подобен стандартному элементу ComboBox
- **ToolStripTextBox**: аналогичен текстовому полю TextBox **ToolStripProgressBar**: индикатор прогресса, как и элемент ProgressBar
- **ToolStripDropDownButton**: представляет кнопку, по нажатию на которую открывается выпадающее меню

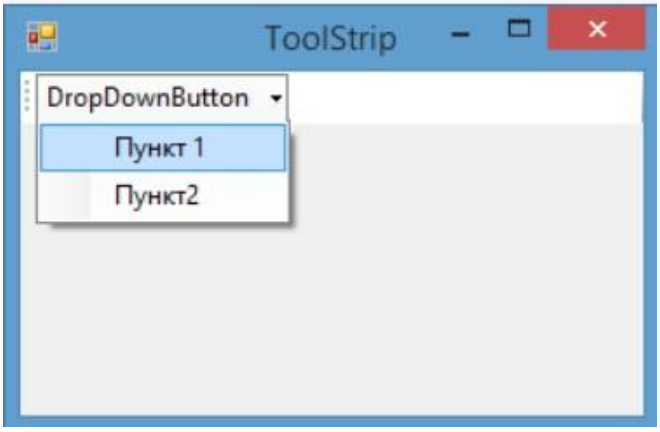

Рисунок 12- Menu ToolStripDropDownButton

К каждому элементу выпадающего меню дополнительно можно прикрепить обработчик нажатия и обработать клик по этим пунктам меню

- **ToolStripSplitButton**: объединяет функциональность ToolStripDropDownButton и ToolStripButton

Добавить новые элементы можно в режиме дизайнера:

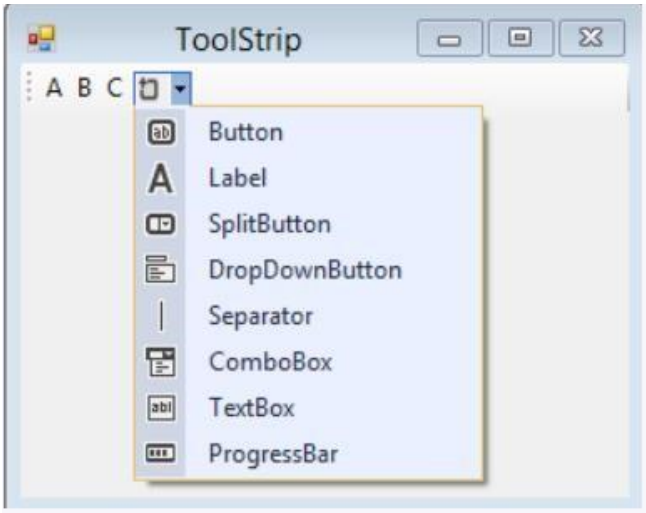

Рисунок 13- элементы в режиме дизайнера

Также можно добавлять новые элементы программно в коде. Их расположение на панели инструментов будет соответствовать порядку добавления. Все элементы хранятся в ToolStrip в свойстве Items. Мы можем добавить в него любой объект класса ToolStripItem (то есть любой из выше перечисленных классов, так как они наследуются от ToolStripItem):

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

```
    public Form1()
    {
        InitializeComponent();

        ToolStripButton clearBtn = new ToolStripButton();
    clearBtn.Text = "Clear";
        // устанавливаем обработчик нажатия
        clearBtn.Click += btn_Click;
        toolStrip1.Items.Add(clearBtn);
    }

    void btn_Click(object sender, EventArgs e)
    {
        MessageBox.Show("Производится удаление");
    }
}
```

Кроме того, здесь задается обработчик, позволяющий обрабатывать нажатия по кнопки на панели инструментов.

Элементы ToolStripButton, ToolStripDropDownButton и ToolStripSplitButton могут отображать как текст, так и изобажения, либо сразу и то, и другое. Для управления размещением изображений в этих элементах имеются следующие свойства:

- DisplayStyle: определяет, будет ли отображаться на элементе текст, или изображение, или и то и другое.
- Image: указывает на само изображение
- ImageAlign: устанавливает выравнивание изображения относительно элемента
- ImageScaling: указывает, будет ли изображение растягиваться, чтобы заполнить все пространство элемента
- ImageTransparentColor: указывает, будет ли цвет изображения прозрачным

Чтобы указать разместить изображение на кнопке, у свойства DisplayStyle надо установить значение Image. Если мы хотим, чтобы кнопка отображала только текст, то надо указать значение Text, либо можно комбинировать два значения с помощью другого значения ImageAndText:

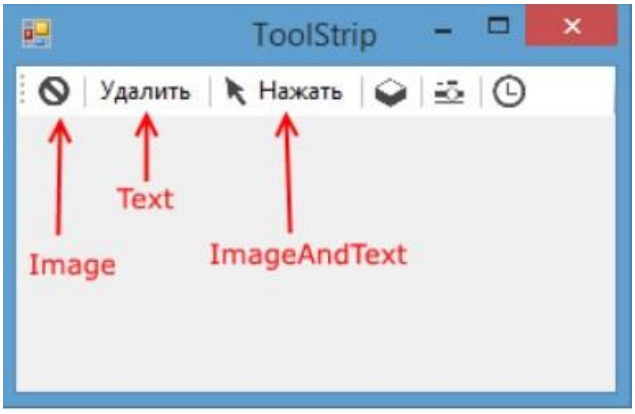

Рисунок 14- Menu ToolStripItemDisplayStyle

Все эти значения хранятся в перечислении **ToolStripItemDisplayStyle**. Также можно установить свойства в коде c#:

```
ToolStripButton clearBtn = new ToolStripButton();
clearBtn.Text = "Поиск"; clearBtn.DisplayStyle =
ToolStripItemDisplayStyle.ImageAndText; clearBtn.Image =
Image.FromFile(@"D:\Icons\0023\search32.png");
// добавляем на панель инструментов
toolStrip1.Items.Add(clearBtn);
```

Элементы ImageList

Элемент **ImageList** не является визуальным элементом управления, однако он представляет собой контейнер, который используется элементами управления.

В контейнер могут быть размещены изображения, которые могут использовать такие элементы, как ListView или TreeView.

Чтобы его добавить в проект, его также можно перенести на форму с Панели Инструментов:

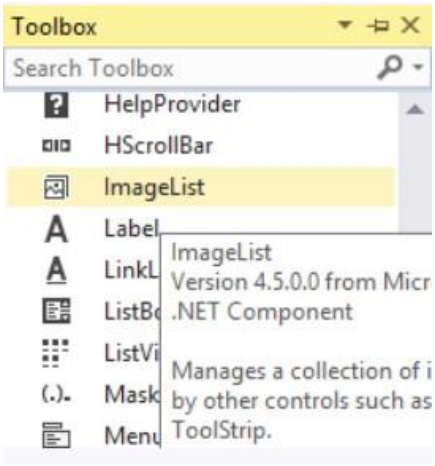

Рисунок 15- Menu Панели Инструментов

Так как компонент не является визуальным элементом, то мы увидим его под формой.

Свойство **Images –** задает коллекцию изображений;

При выборе данного свойства окроется окно редактора изображений, в котором необходимо добавить новое изображение или удалить имеющееся. Свойство **ImageSize –** задает размер изображений для данного ImageList. По умолчанию ширина и высота имеют значение 16 пикселей, но можно изменить в пределах до 256 пикселей.

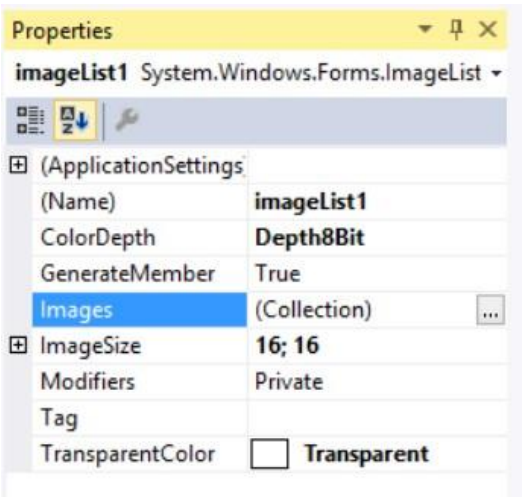

Рисунок 16- свойства Панели Инструментов..

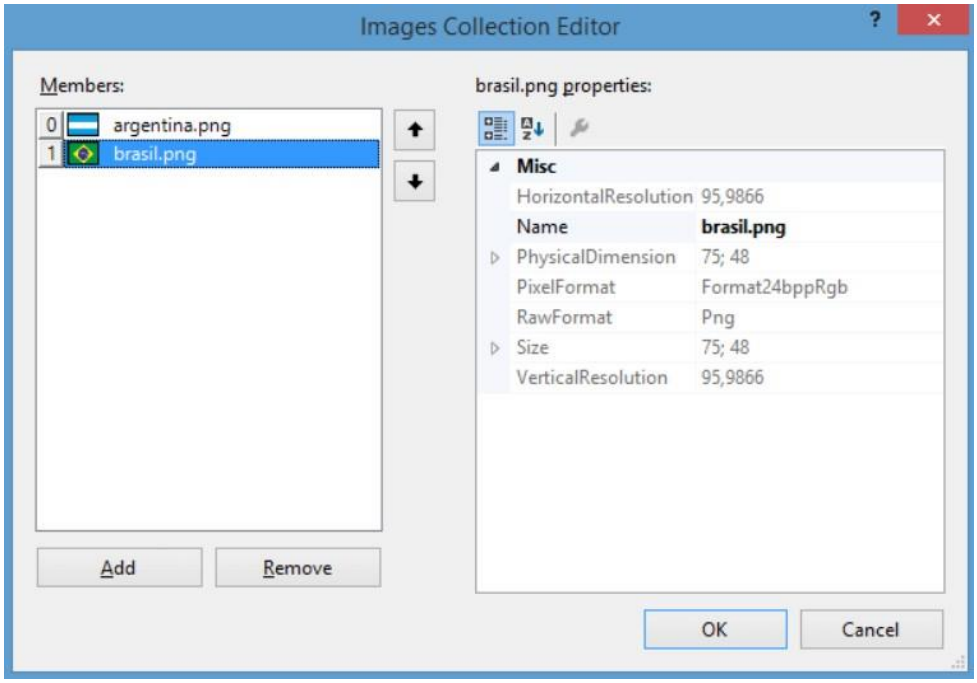

Рисунок 17- Menu коллекции изображений

Чтобы добавить коллекцию изображений из списка программно: ImageList1.Images.Add(Image.FromFile(@"C:\Users\Eugene\Pictures \uruguay.png") ImageList1.Images.RemoveAt(0); // удаляем первое изображение

Например:

Расположим в форме элемент ImageList. Добавим в него три изображения.

Расположим в форме 3 элемента CheckBox.

У каждого CheckBox очистим свойство Text и установим свойство ImageList. В свойстве ImageIndex зададим индексы изображений из ImageList:

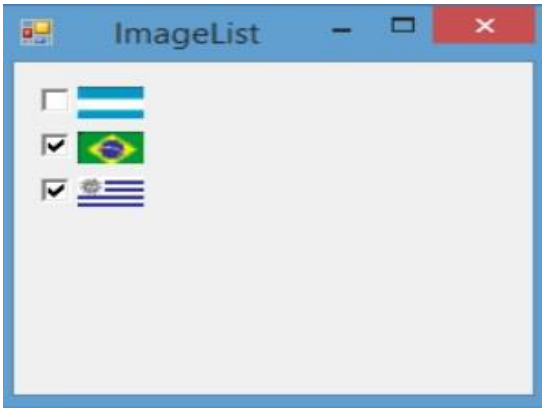

Рисунок 18- Menu ImageList.

И получим форму со списком изображений.

Задания.

1. Разработать класс, инкапсулирующий дату (день, месяц, год). Перегрузить операторы + (количество дней), - (количество дней), == (объект даты), != (объект даты). Реализовать методы вывода даты в разных форматах.

2. Разработать класс, инкапсулирующий время (часы, минуты, секунды). Перегрузить операторы + (количество секунд), + (объект время), - (количество дней), - (объект время), == (объект время), != (объект время). Реализовать методы вывода времени в разных форматах.

3. Разработать класс, инкапсулирующий комплексные числа.

Перегрузить операторы +, -, *, /, ==, !=.

4. Разработать класс, инкапсулирующий простые дроби. Перегрузить операторы +, -, *, /, ==, !=. Реализовать метод упрощения дроби.

5. Разработать классы Point (точка на плоскости) и Points (множество точек на плоскости). Перегрузить операции + (точка) — добавляет точку в множество, - (точка) — убирает точку из множества, если она в нем присутствует, + (множество точек) — объединение множеств, - (множество точек) — вычитание множеств, == - для точки и для множества, != - для точки и для множества.

6. Разработать классы Point (точка в пространстве) и Points (множество точек в пространстве). Перегрузить операции + (точка) — добавляет точку в множество, - (точка) — убирает точку из множества, если она в нем присутствует, + (множество точек) — объединение множеств, - (множество точек) — вычитание множеств, == - для точки и для множества, != - для точки и для множества.

7. Разработать классы Point (точка на плоскости) и Points (множество точек на плоскости). Перегрузить операции + (точка), - (точка) – смещение множества на заданный вектор, * (скаляр) — для точки и для множества, == - для точки и для множества, != - для точки и для множества.

8. Разработать классы Point (точка в пространстве) и Points (множество точек в пространстве). Перегрузить операции + (точка), - (точка) – смещение множества на заданный вектор, * (скаляр) — для точки и для множества, == - для точки и для множества, != - для точки и для множества.

9. Разработать класс, инкапсулирующий матрицу произвольной размерности. Перегрузить операторы +, * (скаляр), * (матрица), ==,

!=.

10. Разработать класс строк на основе массива символов. Перегрузить операторы +, ==, !=.

11. Разработать класс, инкапсулирующий двумерный вектор.

Перегрузить операторы +, -, * (скаляр), ==, !=.

12. Разработать класс, инкапсулирующий трехмерный вектор.

Перегрузить операторы +, -, * (скаляр), ==, !=.

13. Разработать класс, инкапсулирующий вектор произвольной размерности. Перегрузить операторы +, * (скаляр), * (вектор), ==,

!=.

14. Разработать класс, инкапсулирующий количество (вещественное число) с единицей измерения. Перегрузить операторы +, -, ==, !=. Разработать статический класс для конвертации единиц измерения.

15. Разработать классы Student (ФИО, группы) и Students (список студентов). В Students перегрузить операторы + (Student), -

(Student). В Student перегрузить операторы == и !=.

16. Разработать классы Book (автор, название, isbn) и Books (список книг). В Books перегрузить операторы + (Book), - (Book). В Book перегрузить операторы == и !=.

Список используемых источников

1. Белов В.Г. Основы программирования на языке C++ Builder [Текст]: учеб. пособие / В.Г. Белов, Т.М. Белова; Юго-Зап. гос. ун-т. – Курск, 2015. – 160 с.

2. Белов В.Г. Основы программирования на языке C++ Builder [Электронный ресурс]: учеб. пособие / В.Г. Белов, Т.М. Белова; ЮгоЗап. гос. ун-т. – Курск, 2015. – 160 с.

3. Архангельский, А.Я. Программирование в C++ Builder [Текст] /

4. А.Я. Архангельский. – М.: Изд-во БИНОМ, 2010. – 1304 с.

5. Дэвид Р. Мюссер. C++ и STL. Справочное руководство [Текст] / Дэвид Р. Мюссер, Жилмер Дж. Дердж, Атул Сейни. – М.: Вильямс, 2010. – 432 с.

6. Культин, Н. C++ Builder [Текст] / Н. Культин. – СПб.: БХВПетербург, 2012. – 464 с.

7. Лафоре, Р. Объектно-ориентированное программирование в C++ [Текст] / Р. Лафоре. – СПб.: ПИТЕР, 2013. – 924 с.

8. Прата, С. Язык программирования C++. Лекции и упражне-ния [Текст] / С. Прата. – М.: Вильямс, 2012. – 1244 с.

9. http://mycsharp.ru/post/21/2013_06_12_rabota_s_fajlami_v_si-sharp_klassy_streamreader_i_streamwriter.html

10. https://metanit.com/sharp/tutorial/5.3.php

11. https://professorweb.ru/my/csharp/thread_and_files/level3/3_2.php