1 ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ НА ЯЗЫКАХ C++, C# И ПЛАТФОРМА MICROSOFT . NET FRAMEWORK

1.1 Решение задания по использованию библиотечных классов для математических вычислений и ввода-вывода

Для выполнения различных математических операций в библиотеке классов .NET предназначен класс Math. Предоставляет константы и статические методы для тригонометрических, логарифмических и иных общих математических функций. Он является статическим, поэтому все его методы также являются статическими. Также класс Math определяет две константы: Math.E и Math.PI.

Условные конструкции - один из базовых компонентов многих языков программирования, которые направляют работу программы по одному из путей в зависимости от определенных условий.

В языке C# используются следующие условные конструкции: if..else и switch..case.

Конструкция if/else проверяет истинность некоторого условия и в зависимости от результатов проверки выполняет определенный код.

После ключевого слова if ставится условие. И если это условие выполняется, то срабатывает код, который помещен далее в блоке if после фигурных скобок.

При несоблюдении условия можно добавить блок else.

Но при сравнении чисел мы можем насчитать три состояния: первое число больше второго, первое число меньше второго и числа равны. Используя конструкцию else if, мы можем обрабатывать дополнительные условия: Также мы можем соединить сразу несколько условий, используя логические операторы.

Условие: Написать программу, вычисляющую расстояние до линии горизонта от точки, расположенной на высоте h над поверхностью Земли. (Считать Землю идеальным шаром с радиусом R = 6350 км.)

Алгоритм: При помощи методов преобразуем строковое представление числа в эквивалентному ему числу. Возвели в нужную нам степень и получили результат.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace ConsoleApp2

{

class Program

{

public static void Main()

{

Console.Write("Введите высоту: ");

double h = double.Parse(Console.ReadLine());

if(h>0)

{

Console.WriteLine("Расстояние до точки: {0:#.####} " , Math.Sqrt( Math.Pow(6350+h,2) - Math.Pow(6350,2) ) );

}

Console.ReadKey();

}

}

}

Тестирование приложения:

Рисунок 1 – Исполнение программы

1.2 Решение задания с использованием процедур и функций. Перегрузка функций (методов) и операторов

В языке C# существует 2 способа передачи аргументов: передача по значению и передача по ссылке, и результат передачи объекта аргументу по ссылке будет отличаться от передачи значения простого типа.

Передача аргумента по значению — значение фактически копируется в формальный параметр метода. Соответственно, изменения, которые вносятся в параметр метода никак не влияют на аргумент, используемый для вызова. В случае с передачей по ссылке — всё наоборот. Так как передаётся не значение, а ссылка на объект, то изменения, которые вносятся в параметр метода, влияют и на аргумент, используемый для вызова.

Перегрузка функций - это определения нескольких функций с одинаковыми именами, но различными параметрами. Наборы параметров перегруженных функций могут отличаться порядком следования, количеством, типом.

Функции, имеющие одинаковые имена и различные сигнатуры и определенные в одной области видимости, называются перегруженными.

Выбор экземпляра функции при вызове включает сравнение типов и числа фактических аргументов с формальными параметрами в объявлении каждого экземпляра.

При вызове функций с несколькими аргументами правила соответствия применяются для каждого аргумента. При перегрузке такой функции выбирается тот экземпляр, для которого достигнуто либо точное соответствие, либо достигнуто соответствие при помощи преобразований.

Если при перегрузке функций есть один или несколько экземпляров с аргументами по умолчанию, то вызываться будет тот, для которого обеспечены все или некоторое подмножество аргументов, к которым применимы правила соответствия.

Условие: Осуществите в классе перегрузку метода del(тип1 i, тип2 j) c разными типами параметров так, что если тип1 integer, тип2 double, то выполняется действие (i+j)/(i-j), если тип1 double, тип2 integer, то выполнить действие i/j, если один из типов char, вывести на экран значения параметров, разделенные знаком “/”.

Алгоритм: При помощи методов «Дэл,» «Интежер,» «Дабл» выполнили ряд действий, указанных в условии задачи, так же подключили нестандартную библиотеки «$».

Листинг программы:

using System;

namespace csharp\_console {

class thread2174360 {

static void del(int i, double j) {

Console.WriteLine((i + j) / (i - j));

}

static void del(double i, int j) {

Console.WriteLine(i / j);

}

static void del(char i, int j) {

Console.WriteLine($"{i}/{j}");

}

static void del(int i, char j) {

Console.WriteLine($"{i}/{j}");

}

static void del(char i, double j) {

Console.WriteLine($"{i}/{j}");

}

static void del(double i, char j) {

Console.WriteLine($"{i}/{j}");

}

public static void Main(string[] args) {

del('6', 7);

del(7.5, '5');

}

}

}

Тестирование приложения:

Рисунок 2 – Исполнение программы

1.3 Обработка исключительных ситуаций

В языках программирования для обработки исключительных ситуаций предлагались самые разные подходы.

Всюду в тексте модуля, где синтаксически допускается использование блока, этот блок можно сделать охраняемым, добавив ключевое слово try. Вслед за try-блоком могут следовать catch-блоки, называемые блоками-обработчиками исключительных ситуаций, их может быть несколько, они могут и отсутствовать. Завершает эту последовательность finally-блок - блок финализации, который также может отсутствовать. Вся эта конструкция может быть вложенной - в состав try-блока может входить конструкция try-catch-finally.

Условие: Для заданного натурального и действительного подсчитать следующие суммы:

S=sinx+sin sinx+sin sinx+…… N раз

Алгоритм: Задаем промежуток n и x, при помощи структур возводим наше условие квадрат после чего с клавиатуры вводим N. При помощи исключений в программе при вводе отрицательного числа выведется сообщение об ошибке.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Зададим промежуток от n и x...");

Console.Write("n: ");

double s = 0;

double a = double.Parse(Console.ReadLine());

Console.Write("x:");

double b = double.Parse(Console.ReadLine());

try

{

if(a<1)

{

throw new Exception();

}

}

catch (Exception)

{

Console.WriteLine("Ошибка");

}

for (int i = 1; i <= a; i++)

{

s += Math.Pow(Math.Sin(b), i);

}

Console.WriteLine("Функция={0:#.####} " ,s);

Console.ReadKey();

}

}

}

Тестирование приложения:

Рисунок 3 – Исполнение программы

1.4 Решение задания с использованием массивов

В языке C#, соблюдая преемственность, сохранены одномерные массивы и массивы массивов. В дополнение к ним в язык добавлены многомерные массивы. Динамические многомерные массивы языка C# являются весьма мощной, надежной, понятной и удобной структурой данных, которую смело можно рекомендовать к применению не только профессионалам, но и новичкам, программирующим на C#. После этого краткого обзора давайте перейдем к более систематическому изучению деталей работы с массивами в C#.

Объявление массивов.

Рассмотрим, как объявляются одномерные массивы, массивы массивов и многомерные массивы.

Объявление одномерных массивов

Напомню общую структуру объявления:

[<атрибуты>] [<модификаторы>] <тип> <объявители>;

Забудем пока об атрибутах и модификаторах. Объявление одномерного массива выглядит следующим образом:

<тип>[] <объявители>;

Заметьте, в отличие от языка C++ квадратные скобки приписаны не к имени переменной, а к типу. Они являются неотъемлемой частью определения типа, так что запись T[] следует понимать как тип, задающий одномерный массив с элементами типа T.

Условие задачи: Для заданного одномерного массива Z из N элементов проверить, что для всех элементов массива выполняется условие . В рекурсивной функции каждый раз отделять первую треть от остальных двух третей рассматриваемой части массива и применять эту же функцию к одной или обеим частям при необходимости. Рекурсивные вызовы заканчивать, когда останется только один элемент в рассматриваемой части массива.

Например, для N=9: , (∧ – «и»)

Алгоритм решения: Ввелась размерность массива, произошло его заполнение. Были вычислены первые две трети от размера и оставшаяся часть. Произошел вызов рекуррентной функции для первой части, затем для второй и сложили полученные счётчики.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

static int sum = 0; //Счётчик

static int[] A; //Массив

static int Rec(int c, int n)

{

if (Math.Sin(A[c]) < Math.Cos(A[c]))

sum++;

if (c < n - 1)

Rec(c + 1, n);

return sum;

}

static void Main(string[] args)

{

Random rand = new Random();

Console.Write("Введите кол-во элементов: ");

int N = int.Parse(Console.ReadLine());

A = new int[N];

for (int i = 0; i < N; i++)

{

A[i] = rand.Next(-9, 10);

Console.Write("{0} ", A[i]);

}

Console.WriteLine();

int tret = (N / 3) \* 2;

Console.WriteLine("Первые две трети равны {0}.", tret);

int count1 = 0, count2 = tret;

int Count = Rec(count1, tret);

sum = 0;

Count += Rec(count2, N);

Console.WriteLine("Количество элементов, где его синус меньше косинуса = {0}", Count);

}

}

}

Тестирование приложения:

Рисунок 4 – Исполнение программы

1.5 Решение задания с использованием строк

Отличительной особенностью обратной польской нотации является то, что все аргументы (или операнды) расположены перед знаком операции. В общем виде запись выглядит следующим образом:

Запись набора операций состоит из последовательности операндов и знаков операций. Операнды в выражении при письменной записи разделяются пробелами.

Выражение читается слева направо. Когда в выражении встречается знак операции, выполняется соответствующая операция над двумя последними встретившимися перед ним операндами в порядке их записи. Результат операции заменяет в выражении последовательность её операндов и её знак, после чего выражение вычисляется дальше по тому же правилу.

Результатом вычисления выражения становится результат последней вычисленной операции.

Например, рассмотрим вычисление выражения 7 2 3 \* - (эквивалентное выражение в инфиксной нотации: 7-2\*3).

Первый по порядку знак операции — «\*», поэтому первой выполняется операция умножения над операндами 2 и 3 (они стоят последними перед знаком). Выражение при этом преобразуется к виду 7 6 - (результат умножения — 6, — заменяет тройку «2 3 \*»).

Второй знак операции — «-». Выполняется операция вычитания над операндами 7 и 6.

Вычисление закончено. Результат последней операции равен 1, это и есть результат вычисления выражения.

Очевидное расширение обратной польской записи на унарные, тернарные и операции с любым другим количеством операндов: при использовании знаков таких операций в вычислении выражения операция применяется к соответствующему числу последних встретившихся операндов

Особенности обратной польской записи следующие:

Порядок выполнения операций однозначно задаётся порядком следования знаков операций в выражении, поэтому отпадает необходимость использования скобок и введения приоритетов и ассоциативности операций.

В отличие от инфиксной записи, невозможно использовать одни и те же знаки для записи унарных и бинарных операций. Так, в инфиксной записи выражение 5 \* (-3 + 8) использует знак «минус» как символ унарной операции (изменение знака числа), а выражение (10 - 15) \* 3 применяет этот же знак для обозначения бинарной операции (вычитание). Конкретная операция определяется тем, в какой позиции находится знак. Обратная польская запись не позволяет этого: запись 5 3 - 8 + \* (условный аналог первого выражения) будет интерпретирована как ошибочная, поскольку невозможно определить, что «минус» после 5 и 3 обозначает не вычитание; в результате будет сделана попытка вычислить сначала 5 - 3, затем 2 + 8, после чего выяснится, что для операции умножения не хватает операндов. Чтобы всё же записать это выражение, придётся либо переформулировать его (например, записав вместо выражения - 3 выражение 0 - 3), либо ввести для операции изменения знака отдельное обозначение, например, «±»: 5 3 ± 8 + \*.

Так же, как и в инфиксной нотации, в ОПН одно и то же вычисление может быть записано в нескольких разных вариантах. Например, выражение (10 - 15) \* 3 в ОПН можно записать как 10 15 - 3 \*, а можно — как 3 10 15 - \*

Из-за отсутствия скобок обратная польская запись короче инфиксной. За этот счёт при вычислениях на калькуляторах повышается скорость работы оператора (уменьшается количество нажимаемых клавиш), а в программируемых устройствах сокращается объём тех частей программы, которые описывают вычисления. Последнее может быть немаловажно для портативных и встроенных вычислительных устройств, имеющих жёсткие ограничения на объём памяти.

Условие: Написать программу формирования ОПЗ и расчета полученного выражения. Разработать удобный интерфейс ввода исходных данных и вывода результатов. Работу программы проверить на конкретном примере

(a+b)\*(c– d)/e

(a)7.4, (b)3.6, (c)2.8 ,(d)9.5, (e) 0.9, Результат -81.99.

Алгоритм решения: Пользователем ввелось выражение. Данные были записаны в новую строку: изначально числа, затем, исходя из приоритетных значений, символы – при помощи вспомогательной коллекции. В консоли вывелась конечная строка в обратной польской записи, исходя из которой при помощи чисел во вспомогательной коллекции использовались арифметические операции для получения новых величин до конечного результата, который был выведен на консоль.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

class Program

{

private static byte Prioritet(char v)

{

switch (v)

{

case '(':

return 0;

case ')':

return 1;

case '+':

return 2;

case '-':

return 2;

case '\*':

return 3;

case '/':

return 3;

default:

return 3;

}

}

private static bool operat(char v)

{

if (("+-/\*^()".IndexOf(v) != -1))

return true;

return false;

}

static void Main(string[] args)

{

Console.Write("Введите выражение: ");

string input = Console.ReadLine();

string output = string.Empty;

Stack<char> Operacii = new Stack<char>();

for (int i = 0; i < input.Length; i++)

{

if (Char.IsDigit(input[i]))

{

while (!operat(input[i]))

{

output += input[i];

i++;

if (i == input.Length)

break;

}

output = output + " ";

i--;

}

if (operat(input[i]))

{

if (input[i] == '(')

Operacii.Push(input[i]);

else if (input[i] == ')')

{

char s = Operacii.Pop();

while (s != '(')

{

output += s.ToString() + ' ';

s = Operacii.Pop();

}

}

else

{

if (Operacii.Count > 0)

if (Prioritet(input[i]) <= Prioritet(Operacii.Peek()))

output = output + Operacii.Pop().ToString() + " ";

Operacii.Push(char.Parse(input[i].ToString()));

}

}

}

while (Operacii.Count > 0)

output += Operacii.Pop() + " ";

Console.WriteLine("Выражение в обратной польской записи: " + output);

double result = 0;

Stack<double> StackReshenie = new Stack<double>();

for (int i = 0; i < output.Length; i++)

{

if (Char.IsDigit(output[i]))

{

string a = string.Empty;

while (output[i] != ' ' && !operat(output[i]))

{

a = a + output[i];

i++;

if (i == output.Length)

break;

}

StackReshenie.Push(double.Parse(a));

i--;

}

else if (operat(output[i]))

{

double a = StackReshenie.Pop();

double b = StackReshenie.Pop();

switch (output[i])

{

case '+':

result = b + a;

break;

case '-':

result = b - a;

break;

case '\*':

result = b \* a;

break;

case '/':

result = b / a;

break;

}

StackReshenie.Push(result);

}

}

Console.WriteLine("Ответ: " + StackReshenie.Peek());

Console.ReadKey();

}

}

}

Тестирование приложения:

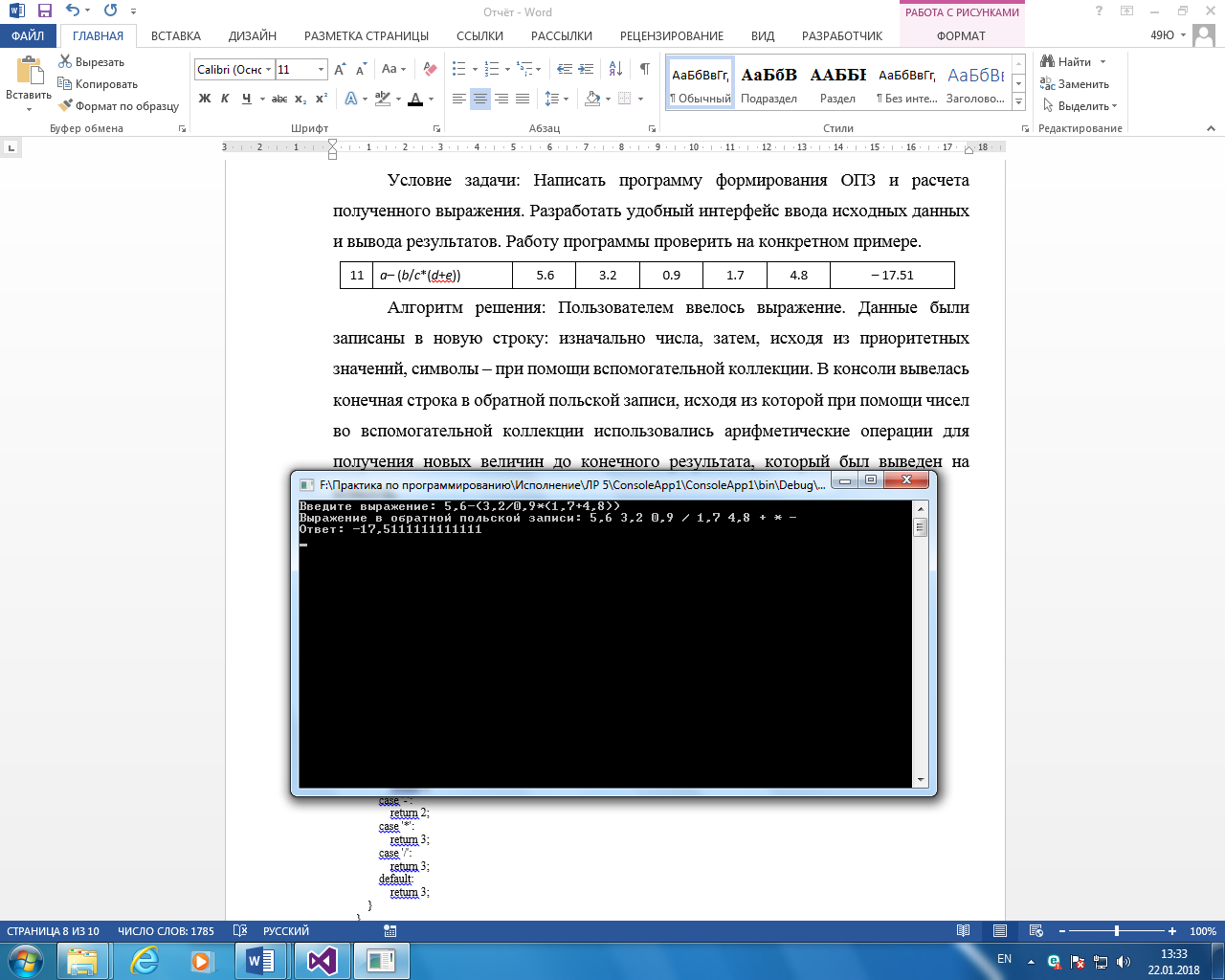


Рисунок 5 – Исполнение программы

1.6 Решение задания с использованием перечислений, структур, интерфейсов

Структура – это более простая версия классов. Все структуры наследуются от базового класса System.ValueType и являются типами значений, тогда как классы - ссылочные типы.

Структуры подходят для создания несложных типов, таких как точка, цвет, окружность. Если необходимо создать множество экземпляров подобного типа, используя структуры, мы экономим память, которая могла бы выделяться под ссылки в случае с классами.

Условие задачи: Описать структуру PEREVOZKI (тип самолета, количество рейсов, налет в часах, налет в тысячах километров). Введите данные в массив, состоящий из шести элементов типа PEREVOZKI. Выведите на экран информацию о самолетах, налет часов которых больше введенного с клавиатуры числа. Если такого самолета нет, то программа должна выдать соответствующее сообщение на экран.

Алгоритм решения: Задан массив структур из шести элементов. Пользователь заполнил необходимые поля, затем ввёл количество часов. Если были найдены с большим числом, чем введенное, списком на консоли вывелись сведения о данном самолете, иначе уведомление об отсутствии таковых.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ConsoleApp1

{

struct PEREVOZKI

{

public string type;

public int count, hours, km;

}

//Описать структуру PEREVOZKI(тип самолета, количество рейсов, налет в часах, налет в тысячах километров).

//Введите данные в массив, состоящий из шести элементов типа PEREVOZKI.

// Выведите на экран информацию о самолетах, налет часов которых больше введенного с клавиатуры числа.

// Если такого самолета нет, то программа должна выдать соответствующее сообщение на экран.

public class Program

{

static void Main(string[] args)

{

PEREVOZKI[] p = new PEREVOZKI[6];

Console.WriteLine("Заполнение данных:");

for (int i = 0; i < 6; i++)

{

Console.WriteLine("Книга №{0}:", i + 1);

Console.Write("Введите тип самолета: ");

p[i].type = Console.ReadLine();

Console.Write("Введите количество рейсов: ");

p[i].count = int.Parse(Console.ReadLine());

Console.Write("Введите налет в часах: ");

p[i].hours = int.Parse(Console.ReadLine());

Console.Write("Введите налет в тысячах километров: ");

p[i].km = int.Parse(Console.ReadLine());

}

Console.Write("Введите налет часов: ");

int vvod = int.Parse(Console.ReadLine());

int count = 0;

bool b = false;

for (int i = 0; i < 6; i++)

{

if (p[i].hours > vvod)

{

count++;

Console.WriteLine("{0}) тип самолета '{1}', количество рейсов {2}, налет в часах {3} и налет в тысячах километров {4}", count, p[i].type, p[i].count, p[i].hours, p[i].km);

b = true;

}

}

if (b == false)

{

Console.WriteLine("Нет таких самолетов.");

}

Console.ReadKey();

}

}

}

Тестирование приложения:

Рисунок 6 – Исполнение программы

1.7 Реализация механизма наследования

Наследование – это тот аспект ООП, который способствует многократному использованию кода. Основная идея наследования состоит в том, что новые классы могут использовать и/или расширять функциональность других классов. При этом поддерживается концепция иерархической классификации, имеющей направление сверху вниз. Используя наследование, объект должен определить только те качества, которые делают его уникальным в пределах своего класса.

Синтаксис:

class имя\_класса : имя\_родительского\_класса

{тело\_класса}

В семействе классов базовый класс часто является обобщенным (в нашем примере таков класс сотрудника Employee). Его задача состоит в определении общих полей, свойств и методов для классов-наследников, а создание экземпляров обобщенного класса не имеет смыла. В этом случае в C# используют ключевое слово abstract.

Назначение абстрактного класса заключается в предоставлении общего определения для базового класса, которое могут совместно использовать несколько производных классов. Создавать экземпляры абстрактного класса нельзя. Абстрактные классы могут определять абстрактные методы. Для этого перед типом возвращаемого значения метода необходимо поместить ключевое слово abstract. Абстрактные методы не имеют реализации, поэтому определение такого метода заканчивается точкой с запятой вместо обычного блока метода. Классы, производные от абстрактного класса, должны реализовывать все абстрактные методы. Если абстрактный класс наследует виртуальный метод из базового класса, абстрактный класс может переопределить виртуальный метод с помощью абстрактного метода.

Условие задачи 1: в соответствии с вариантом задания самостоятельно разработать класс и программу, иллюстрирующую его возможности – «корабль».

Алгоритм решения: Задан объекта класса без использования параметров, вызвана его функция вывода данных на консоль. Пользователем введены переменные и создался объект с параметрами, где использовались его функции.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

namespace ConsoleApp5

{

public class Ship

{

public string name, color, appointment;

public int weight, crew;

private int year;

public Ship()

{

name = "Элизабет";

color = "Коричневый";

appointment = "Торговый";

crew = 48;

weight = 13;

year = 1997;

}

public Ship(string name, string color, string appointment, int crew, int weight, int year)

{

this.name = name;

this.color = color;

this.appointment = appointment;

this.crew = crew;

this.weight = weight;

this.year = year;

}

public string Color

{

get { return color; }

set { color = value; }

}

public int Weight

{

get { return weight; }

set

{

weight = value;

if ((value <= 100000) && (value >= 30000))

Console.WriteLine("Средний вес");

}

}

public void print()

{

Console.WriteLine("Корабль:\nНазвание: {0};\nЦвет: {1};\nНазначение: {2};\nЧисленность экипажа: {3};\nВес: {4};\nГод выпуска: {5};\n",

name, color, appointment, crew, weight, year);

}

public static Ship operator ++(Ship obj1)

{

obj1.weight += 1;

return obj1;

}

}

public class Program

{

static void Main(string[] args)

{

Ship pp = new Ship();

pp.print();

string name, color, appointment;

int weight, year, crew;

Console.WriteLine("Заполнение данных корабля:");

Console.Write("Введите название: ");

name = Console.ReadLine();

Console.Write("Введите цвет: ");

color = Console.ReadLine();

Console.Write("Введите назначение: ");

appointment = Console.ReadLine();

Console.Write("Введите численность экипажа: ");

crew = int.Parse(Console.ReadLine());

Console.Write("Введите вес: ");

weight = int.Parse(Console.ReadLine());

Console.Write("Введите год выпуска: ");

year = int.Parse(Console.ReadLine());

Ship p\_p = new Ship(name, color, appointment, crew, weight, year);

Console.WriteLine();

p\_p.print();

p\_p.Color = color;

p\_p.Weight = weight;

p\_p.weight++;

Console.WriteLine("-----------------------------------");

p\_p.print();

Console.ReadKey();

}

}

}

Тестирование приложения:

Рисунок 7 – Исполнение программы

Условие задачи 2: построить иерархию классов в соответствии с вариантом задания – «корабль, пароход, парусник, корвет».

Алгоритм решения: Пользователь заполнил переменные, которые затем были инициализированы в массив класса и информация о них вывелась на консоль.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.IO;

namespace ConsoleApp5

{

abstract class Ship

{

public string name, color, appointment;

public int weight, crew, year;

public Ship()

{

name = "Элизабет";

color = "Коричневый";

appointment = "Торговый";

crew = 48;

weight = 13;

year = 1997;

}

public Ship(string name, string color, string appointment, int crew, int weight, int year)

{

this.name = name;

this.color = color;

this.appointment = appointment;

this.crew = crew;

this.weight = weight;

this.year = year;

}

public string Color

{

get { return color; }

set { color = value; }

}

public int Weight

{

get { return weight; }

set

{

weight = value;

if ((value <= 100000) && (value >= 30000))

Console.WriteLine("Средний вес");

}

}

public abstract void print();

public static Ship operator ++(Ship obj1)

{

obj1.weight += 1;

return obj1;

}

}

class SailingShip : Ship

{

public string sail;

public SailingShip(string sail, string name, string color, string appointment, int crew, int weight, int year)

:base (name, color, appointment, crew, weight, year)

{

this.sail = sail;

}

public override void print()

{

Console.WriteLine("Корабль:\nНазвание: {0};\nЦвет: {1};\nНазначение: {2};\nЧисленность экипажа: {3};\nВес: {4};\nГод выпуска: {5};\nВид паруса: {6}.\n",

name, color, appointment, crew, weight, year, sail);

}

}

class Steamship : Ship

{

public string engine;

public Steamship(string engine, string name, string color, string appointment, int crew, int weight, int year)

: base(name, color, appointment, crew, weight, year)

{

this.engine = engine;

}

public override void print()

{ }

public void printer()

{

Console.WriteLine("Корабль:\nНазвание: {0};\nЦвет: {1};\nНазначение: {2};\nЧисленность экипажа: {3};\nВес: {4};\nГод выпуска: {5};",

name, color, appointment, crew, weight, year);

Console.WriteLine("Двигатель: {0};", engine);

}

}

class Corvette : Steamship

{

public string country;

public Corvette(string country, string engine, string name, string color, string appointment, int crew, int weight, int year)

: base(engine, name, color, appointment, crew, weight, year)

{

this.country = "Великобритания";

}

public override void print()

{

printer();

Console.WriteLine("Страна: {0}.\n", country);

}

}

public class Program

{

static void Main(string[] args)

{

string name, color, appointment, sail, engine, country;

int weight, year, crew;

Corvette tb = new Corvette("Франция", "UG-1836E", "Изольда", "Серебро", "Военный", 262, 80000, 2012);

tb.print();

Console.WriteLine("Введите количество парусников: ");

int n = int.Parse(Console.ReadLine());

SailingShip[] j = new SailingShip[n];

for (int i = 0; i < n; i++)

{

Console.WriteLine("Заполнение данных парусника:");

Console.Write("Введите название: ");

name = Console.ReadLine();

Console.Write("Введите цвет: ");

color = Console.ReadLine();

Console.Write("Введите назначение: ");

appointment = Console.ReadLine();

Console.Write("Введите численность экипажа: ");

crew = int.Parse(Console.ReadLine());

Console.Write("Введите вес: ");

weight = int.Parse(Console.ReadLine());

Console.Write("Введите год выпуска: ");

year = int.Parse(Console.ReadLine());

Console.Write("Введите вид паруса: ");

sail = Console.ReadLine();

j[i] = new SailingShip(sail, name, color, appointment, crew, weight, year);

Console.WriteLine();

}

j[0].weight++;

for (int i = 0; i < n; i++)

{

j[i].print();

}

Console.WriteLine("Введите количество пароходов: ");

int nn = int.Parse(Console.ReadLine());

Steamship[] jj = new Steamship[n];

for (int i = 0; i < n; i++)

{

Console.WriteLine("Заполнение данных парохода:");

Console.Write("Введите название: ");

name = Console.ReadLine();

Console.Write("Введите цвет: ");

color = Console.ReadLine();

Console.Write("Введите назначение: ");

appointment = Console.ReadLine();

Console.Write("Введите численность экипажа: ");

crew = int.Parse(Console.ReadLine());

Console.Write("Введите вес: ");

weight = int.Parse(Console.ReadLine());

Console.Write("Введите год выпуска: ");

year = int.Parse(Console.ReadLine());

Console.Write("Введите двигатель: ");

engine = Console.ReadLine();

jj[i] = new Steamship(engine, name, color, appointment, crew, weight, year);

Console.WriteLine();

}

for (int i = 0; i < n; i++)

{

j[i].print();

}

Console.ReadKey();

}

}

}

Тестирование приложения:

Рисунок 8 – Исполнение программы

1.8 Решение задания с использованием делегатов и событий

Делегат – это объект, имеющий ссылку на метод. Делегат позволяет выбрать вызываемый метод во время выполнения программы. Фактически значение делегата – это адрес области памяти, где находится точка входа метода.

Делегат позволяет указать в коде программы вызов метода, но фактически вызываемый метод определяется во время работы программы, а не во время компилирования.

Делегат объявляется с помощью ключевого слова delegate, за которым указывается тип возвращаемого значения, имя делегата и список параметров вызываемых методов. Примеры объявления классов делегатов:

delegate int ClassDelegate(int key);

delegate void XXX(int intKey, float fKey);

Характерной особенностью делегата является возможность его использования для вызова любого метода, который соответствует сигнатуре делегата. Это дает возможность определить во время выполнения программы, какой из методов должен быть вызван. Вызываемый метод может быть методом экземпляра, ассоциированным с объектом, либо статическим методом, ассоциированным с классом. Метод можно вызвать только тогда, когда его сигнатура (тип возвращаемого значения, набор параметров) соответствует сигнатуре делегата.

Условие задачи: в соответствии с вариантом задания самостоятельно разработать класс и программу, иллюстрирующую его возможности – «корабль».

Алгоритм решения: Произошла инициализация делегата по статической целочисленной функции, его вызов. Объявлены объекты классов, на которых использовано событие и подписка, затем его вызов, отписка и повторный вызов. Дальнейший алгоритм повторяет предыдущую задачу.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.IO;

namespace ConsoleApp5

{

public delegate void info\_print();

class Ship

{

public event info\_print event\_print;

public string name, color, appointment;

public int weight, crew;

private int year;

public Ship()

{

name = "Элизабет";

color = "Коричневый";

appointment = "Торговый";

crew = 48;

weight = 13;

year = 1997;

}

public Ship(string name, string color, string appointment, int crew, int weight, int year)

{

this.name = name;

this.color = color;

this.appointment = appointment;

this.crew = crew;

this.weight = weight;

this.year = year;

}

public string Color

{

get { return color; }

set { color = value; }

}

public int Weight

{

get { return weight; }

set

{

weight = value;

if ((value <= 100000) && (value >= 30000))

Console.WriteLine("Средний вес");

}

}

public void print()

{

Console.WriteLine("Корабль:\nНазвание: {0};\nЦвет: {1};\nНазначение: {2};\nЧисленность экипажа: {3};\nВес: {4};\nГод выпуска: {5};\n",

name, color, appointment, crew, weight, year);

}

public static Ship operator ++(Ship obj1)

{

obj1.weight += 1;

return obj1;

}

public void Ship\_e\_p()

{

Console.WriteLine("Обработчик события вызван.");

}

public void events\_now()

{

event\_print();

}

public void events\_job()

{

Console.WriteLine("Обработчик события.");

}

}

public class Program

{

public delegate int Del(int DT);

static int on\_date\_time(int dt)

{

return dt + 1;

}

static void Main(string[] args)

{

Del d = new Del(on\_date\_time);

Console.WriteLine("Делегат создан {0} числа", d(1));

Console.WriteLine();

Ship pp = new Ship();

pp.print();

Console.WriteLine("Подписка на событиe активирована: ");

pp.event\_print += new info\_print(pp.events\_job);

pp.event\_print += new info\_print(pp.Ship\_e\_p);

pp.events\_now();

Console.WriteLine();

Console.WriteLine("Подписка на событие отменена: ");

pp.event\_print -= new info\_print(pp.events\_job);

pp.events\_now();

Console.WriteLine();

string name, color, appointment;

int weight, year, crew;

Console.WriteLine("Заполнение данных корабля:");

Console.Write("Введите название: ");

name = Console.ReadLine();

Console.Write("Введите цвет: ");

color = Console.ReadLine();

Console.Write("Введите назначение: ");

appointment = Console.ReadLine();

Console.Write("Введите численность экипажа: ");

crew = int.Parse(Console.ReadLine());

Console.Write("Введите вес: ");

weight = int.Parse(Console.ReadLine());

Console.Write("Введите год выпуска: ");

year = int.Parse(Console.ReadLine());

Ship p\_p = new Ship(name, color, appointment, crew, weight, year);

Console.WriteLine();

p\_p.print();

p\_p.Color = color;

p\_p.Weight = weight;

p\_p.weight++;

p\_p.print();

Console.ReadKey();

}

}

}

Тестирование приложения:

Рисунок 9 – Исполнение программы

1.9 Решение задания с использованием асинхронных делегатов и многопоточностью

С помощью языка C# можно создавать приложения, которые выполняют несколько задач одновременно. Задачи, которые потенциально могут задержать выполнение других задач, выполняются в отдельных потоках; такой способ организации работы приложения называется многопоточностью или свободным созданием потоков.

Приложения, использующие многопоточность, более оперативно реагируют на действия пользователя, поскольку пользовательский интерфейс остается активным, в то время как задачи, требующие интенсивной работы процессора, выполняются в других потоках. Многопоточность также эффективна при создании масштабируемых приложений, поскольку пользователь может добавлять потоки при увеличении рабочей нагрузки.

Основной функционал для использования потоков в приложении сосредоточен в пространстве имен System.Threading. В нем определен класс, представляющий отдельный поток - класс Thread.

Класс Thread определяет ряд методов и свойств, которые позволяют управлять потоком и получать информацию о нем.

Асинхронные делегаты позволяют вызывать методы, на которые эти делегаты указывают, в асинхронном режиме. В теме проделегаты говорилось, что делегаты могут вызываться как с помощью метода Invoke, так и в асинхронном режиме с помощью пары методов BeginInvoke/EndInvoke.

Условие задачи: На основе предыдущего задания продемонстрируйте многопоточность приложения. Добавьте асинхронный делегат и продемонстрируйте его работу – «корабль».

Алгоритм решения: Произошла инициализация делегата по статической целочисленной функции, его вызов. Объявлены объекты классов, на которых использовано событие и подписка, затем его вызов, отписка и повторный вызов. Дальнейший алгоритм повторяет предыдущую задачу.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.IO;

using System.Threading;

namespace ConsoleApp5

{

public delegate void info\_print();

class Ship

{

public event info\_print event\_print;

public string name, color, appointment;

public int weight, crew;

private int year;

public Ship()

{

name = "Элизабет";

color = "Коричневый";

appointment = "Торговый";

crew = 48;

weight = 13;

year = 1997;

}

public Ship(string name, string color, string appointment, int crew, int weight, int year)

{

this.name = name;

this.color = color;

this.appointment = appointment;

this.crew = crew;

this.weight = weight;

this.year = year;

}

public string Color

{

get { return color; }

set { color = value; }

}

public int Weight

{

get { return weight; }

set

{

weight = value;

if ((value <= 100000) && (value >= 30000))

Console.WriteLine("Средний вес");

}

}

public void print()

{

Console.WriteLine("Корабль:\nНазвание: {0};\nЦвет: {1};\nНазначение: {2};\nЧисленность экипажа: {3};\nВес: {4};\nГод выпуска: {5};\n",

name, color, appointment, crew, weight, year);

}

public static Ship operator ++(Ship obj1)

{

obj1.weight += 1;

return obj1;

}

public void Ship\_e\_p()

{

Console.WriteLine("Обработчик события вызван.");

}

public void events\_now()

{

event\_print();

}

public void events\_job()

{

Console.WriteLine("Обработчик события.");

}

}

public class Program

{

public delegate int Del(int DT);

static int on\_date\_time(int dt)

{

return dt + 1;

}

static void ThreadFunction(Object input)

{

int flag = (int)input;

if (flag == 1)

{

Console.WriteLine("Это первый поток!");

}

else

{

Console.WriteLine("Это второй поток!");

}

}

static void EggsFunction()

{

for (int i = 0; i < 3; i++)

{

Console.Write(" text ");

Thread.Sleep(15);

}

}

static void Main(string[] args)

{

Thread thread1 = new Thread(ThreadFunction);

thread1.Start(1);

Thread thread2 = new Thread(ThreadFunction);

thread2.Start(2);

Thread thread3 = new Thread(EggsFunction);

thread3.Start();

Del d = new Del(on\_date\_time);

Console.WriteLine("Делегат создан {0} числа", d(1));

Console.WriteLine();

Ship pp = new Ship();

pp.print();

Console.WriteLine("Подписка на событиe активирована: ");

pp.event\_print += new info\_print(pp.events\_job);

pp.event\_print += new info\_print(pp.Ship\_e\_p);

pp.events\_now();

Console.WriteLine();

Console.WriteLine("Подписка на событие отменена: ");

pp.event\_print -= new info\_print(pp.events\_job);

pp.events\_now();

Console.WriteLine();

string name, color, appointment;

int weight, year, crew;

Console.WriteLine("Заполнение данных корабля:");

Console.Write("Введите название: ");

name = Console.ReadLine();

Console.Write("Введите цвет: ");

color = Console.ReadLine();

Console.Write("Введите назначение: ");

appointment = Console.ReadLine();

Console.Write("Введите численность экипажа: ");

crew = int.Parse(Console.ReadLine());

Console.Write("Введите вес: ");

weight = int.Parse(Console.ReadLine());

Console.Write("Введите год выпуска: ");

year = int.Parse(Console.ReadLine());

Ship p\_p = new Ship(name, color, appointment, crew, weight, year);

Console.WriteLine();

p\_p.print();

p\_p.Color = color;

p\_p.Weight = weight;

p\_p.weight++;

p\_p.print();

Console.ReadKey();

}

}

}

Тестирование приложения:

Рисунок 10 – Исполнение программы

1.10 Программирование обобщенных классов

Язык С# позволяет создавать собственные классы-прототипы и их разновидности – интерфейсы, структуры, делегаты, события.

Обобщенные классы могут входить в иерархию классов аналогично необобщенным классам. Следовательно, обобщенный класс может действовать как базовый или производный класс. Главное отличие между иерархиями обобщенных и необобщенных классов заключается в том, что в первом случае аргументы типа, необходимые обобщенному базовому классу, должны передаваться всеми производными классами вверх по иерархии аналогично передаче аргументов конструктора.

Помимо обобщенных классов и методов, в C# допускаются обобщенные интерфей­сы. Такие интерфейсы указываются аналогично обобщенным классам.

Иногда требуется иметь отдельный метод, параметризованный каким-либо типом данных. Примером такого метода может служить метод сортировки данных.

Как и методы, делегаты также могут быть обобщенными. Ниже приведена общая форма объявления обобщенного делегата.

delegate возвращаемый\_тип имя\_делегата<список\_параметров\_типа>(список\_аргументов);

Условие задачи: На основе предыдущего задания создайте обобщенный класс, обобщенный метод, обобщенный интерфейс и обобщенный делегат – «корабль».

Алгоритм решения: Произошла инициализация обобщенного делегата по статической целочисленной функции, его вызов. Объявлен целочисленный объект типа int32 обобщенного класса, реализующего обобщенный интерфейс; вызвана его функция. Создан список строк, выведенный на консоль; использован обобщенный метод сортировки списка и его повторное отображение на экране. Дальнейший алгоритм повторяет предыдущую задачу.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.IO;

using System.Threading;

namespace ConsoleApp5

{

public delegate void info\_print();

class Ship

{

public event info\_print event\_print;

public string name, color, appointment;

public int weight, crew;

private int year;

public Ship()

{

name = "Элизабет";

color = "Коричневый";

appointment = "Торговый";

crew = 48;

weight = 13;

year = 1997;

}

public Ship(string name, string color, string appointment, int crew, int weight, int year)

{

this.name = name;

this.color = color;

this.appointment = appointment;

this.crew = crew;

this.weight = weight;

this.year = year;

}

public string Color

{

get { return color; }

set { color = value; }

}

public int Weight

{

get { return weight; }

set

{

weight = value;

if ((value <= 100000) && (value >= 30000))

Console.WriteLine("Средний вес");

}

}

public void print()

{

Console.WriteLine("Корабль:\nНазвание: {0};\nЦвет: {1};\nНазначение: {2};\nЧисленность экипажа: {3};\nВес: {4};\nГод выпуска: {5};\n",

name, color, appointment, crew, weight, year);

}

public static Ship operator ++(Ship obj1)

{

obj1.weight += 1;

return obj1;

}

public void Ship\_e\_p()

{

Console.WriteLine("Обработчик события вызван.");

}

public void events\_now()

{

event\_print();

}

public void events\_job()

{

Console.WriteLine("Обработчик события.");

}

}

public interface ISort<T>

where T : struct

{

void ReWrite();

}

class MyObj<T> : ISort<T> where T : struct

{

public int longOb { get; set; }

T[] myarr;

public MyObj(int i)

{

longOb = i;

}

public MyObj(int i, T[] arr)

{

longOb = i;

myarr = new T[i];

for (int j = 0; j < arr.Length; j++)

myarr[j] = arr[j];

}

public void ReWrite()

{

Console.WriteLine("Тип: {0}", typeof(T));

Console.WriteLine("Распределение кораблей по областям: ");

foreach (T t in myarr)

Console.Write("{0}; ", t);

Console.WriteLine("\n");

}

public static int SumInt(int a, int b)

{

return a + b;

}

}

delegate T MyDel<T>(T obj1, T obj2);

public class Program

{

public delegate int Del(int DT);

static int on\_date\_time(int dt)

{

return dt + 1;

}

static void ThreadFunction(Object input)

{

int flag = (int)input;

if (flag == 1)

{

Console.WriteLine("Это первый поток!");

}

else

{

Console.WriteLine("Это второй поток!");

}

}

static void EggsFunction()

{

for (int i = 0; i < 3; i++)

{

Console.Write(" text ");

Thread.Sleep(15);

}

}

static void Sort<T>(ref T[] a) where T : IComparable<T>

{

T buf;

int n = a.Length;

for (int i = 0; i < n; i++)

{

int im = i;

for (int j = i + 1; j < n; j++)

if (a[j].CompareTo(a[im]) < 0) im = j;

buf = a[i]; a[i] = a[im]; a[im] = buf;

}

}

static void Main(string[] args)

{

MyDel<int> del1 = MyObj<int>.SumInt;

Console.WriteLine("Количество военных и торговых кораблей = " + del1(136, 74));

int[] MyArrByte = new int[6] { 41, 50, 183, 566, 87, 65 };

MyObj<int> ByteConst = new MyObj<int>(MyArrByte.Length, MyArrByte);

ByteConst.ReWrite();

Console.WriteLine("Названия кораблей: ");

string[] a = { "Меркант", "Абильгейт", "Кэрвэт", "Фельхарг", "Тур", "Айлива", "Стонергам" };

foreach (string x in a) Console.Write(x + "; ");

Console.WriteLine();

Console.WriteLine("Упорядочивание названий в алфавитном порядке: ");

Sort<string>(ref a);

foreach (string x in a) Console.Write(x + "; ");

Console.WriteLine();

Console.WriteLine();

Thread thread1 = new Thread(ThreadFunction);

thread1.Start(1);

Thread thread2 = new Thread(ThreadFunction);

thread2.Start(2);

Thread thread3 = new Thread(EggsFunction);

thread3.Start();

Del d = new Del(on\_date\_time);

Console.WriteLine("Делегат создан {0} числа", d(1));

Console.WriteLine();

Ship pp = new Ship();

pp.print();

Console.WriteLine("Подписка на событиe активирована: ");

pp.event\_print += new info\_print(pp.events\_job);

pp.event\_print += new info\_print(pp.Ship\_e\_p);

pp.events\_now();

Console.WriteLine();

Console.WriteLine("Подписка на событие отменена: ");

pp.event\_print -= new info\_print(pp.events\_job);

pp.events\_now();

Console.WriteLine();

string name, color, appointment;

int weight, year, crew;

Console.WriteLine("Заполнение данных корабля:");

Console.Write("Введите название: ");

name = Console.ReadLine();

Console.Write("Введите цвет: ");

color = Console.ReadLine();

Console.Write("Введите назначение: ");

appointment = Console.ReadLine();

Console.Write("Введите численность экипажа: ");

crew = int.Parse(Console.ReadLine());

Console.Write("Введите вес: ");

weight = int.Parse(Console.ReadLine());

Console.Write("Введите год выпуска: ");

year = int.Parse(Console.ReadLine());

Ship p\_p = new Ship(name, color, appointment, crew, weight, year);

Console.WriteLine();

p\_p.print();

p\_p.Color = color;

p\_p.Weight = weight;

p\_p.weight++;

p\_p.print();

Console.ReadKey();

}

}

}

Тестирование приложения:

Рисунок 11 – Исполнение программы

1.11 Работа с файлами

Центральную часть потоковой С#-системы занимает класс Stream пространства имен System.IO. Класс Stream представляет байтовый поток и является базовым для всех остальных потоковых классов. Из класса Stream выведены такие байтовые классы потоков как:

* FileStream - байтовый поток, разработанный для файлового ввода-вывода
* BufferedStream - заключает в оболочку байтовый поток и добавляет буферизацию, которая во многих случаях увеличивает производительность программы;
* MemoryStream - байтовый поток, который использует память для хранения данных.

Чтобы создать байтовый поток, связанный с файлом, создается объект класса FileStream. При этом в классе определено несколько конструкторов.

Чтобы создать символьный поток нужно поместить объект класса Stream (например, FileStream ) "внутрь" объекта классаStreamWriter или объекта класса StreamReader. В этом случае байтовый поток будет автоматически преобразовываться в символьный.

Класс StreamWriter предназначен для организации выходного символьного потока.

Условие задачи 1: Создать файл и записать в него обратные натуральные числа 1,1/2,...,1/n. Вывести на экран все компоненты файла с порядковым номером, кратным 3.

Условие задачи 2: Дан текстовый файл. Найти самую длинную строку и ее длину.

Алгоритм решения: Пользователем введен интервал и шаг последовательности. Произошло создание файла и запись в него чисел, которые затем перебором по байтам переведены в вещественный тип и, при удовлетворении условия на положительность, вывелись на консоль. При выполнении следующего задания введено число, затем файл был открыт и циклом при совпавшем введенном числе с длиной строки выводится она выводилась на экран.

Листинг программы 1:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите n: ");

int n = int.Parse(Console.ReadLine());

FileStream f = new FileStream("Numbers\_text.txt", FileMode.Create);

BinaryWriter fOut = new BinaryWriter(f);

for (double i = 0; i <= n; i++)

{

fOut.Write(1 / i);

}

fOut.Close();

Console.WriteLine("Вывод из файла: ");

f = new FileStream("Numbers\_text.txt", FileMode.Open);

BinaryReader fIn = new BinaryReader(f);

long m = f.Length;

for (long i = 16; i < m; i += 24)

{

f.Seek(i, SeekOrigin.Begin);

double foo = fIn.ReadDouble();

if (foo > 0)

Console.Write("{0} ", foo);

}

fIn.Close();

f.Close();

Console.ReadKey();

}

}

}

Тестирование приложения:

Рисунок 12 – Исполнение программы

Листинг программы 2:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

int max = 0;

string strmax = "";

Console.WriteLine("Вывод из файла: ");

FileStream fSymb = new FileStream("Symbol\_text.txt", FileMode.Open);

StreamReader fO = new StreamReader(fSymb);

String lines;

while ((lines = fO.ReadLine()) != null)

{

if (lines.Length > max)

{

strmax = lines;

max = lines.Length;

}

}

Console.WriteLine("Строка: " + strmax);

Console.WriteLine("Длина: " + max);

}

}

}

Тестирование приложения:

Рисунок 13 – Исполнение программы

1.12 Создание библиотек

В .Net приложения можно развертывать простым копированием всех необходимых сборок в одну папку на жестком диске, но на практике обычно выделяются отдельные подкаталоги для связанного содержимого, например, библиотек. В файлах конфигурации .NET можно указать подкаталоги, в которых среда выполнения будет искать частные сборки при запуске приложения.

В C# определен ряд директив препроцессора, оказывающих влияние на интерпретацию исходного кода программы компилятором. Эти директивы определяют порядок интерпретации текста программы перед ее трансляцией в объектный код в том исходном файле, где они появляются. Все директивы препроцессора начинаются со знака #. Кроме того, каждая директива препроцессора должна быть выделена в отдельную строку кода.

Условие задачи: Создайте из класса, определенного в задании 1 дня 9, библиотеку (сборку). Подпишите ее. Создайте приложение, которое будет использовать Вашу сборку. Настройте ее с помощью конфигурационного файла. Добавьте директивы #pragma, #line.

Алгоритм решения: Задан объекта класса без использования параметров, вызвана его функция вывода данных на консоль. Пользователем введены переменные и создался объект с параметрами, где использовались его функции.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using ClassLibrary1;

namespace ConsoleApp5

{

public class Program

{

static void Main(string[] args)

{

#line hidden

Ship pp = new Ship();

pp.print();

#pragma warning disable 168

string name, color, appointment;

int weight, year, crew;

Console.WriteLine("Заполнение данных корабля:");

Console.Write("Введите название: ");

name = Console.ReadLine();

Console.Write("Введите цвет: ");

color = Console.ReadLine();

Console.Write("Введите назначение: ");

appointment = Console.ReadLine();

Console.Write("Введите численность экипажа: ");

crew = int.Parse(Console.ReadLine());

Console.Write("Введите вес: ");

weight = int.Parse(Console.ReadLine());

Console.Write("Введите год выпуска: ");

year = int.Parse(Console.ReadLine());

Ship p\_p = new Ship(name, color, appointment, crew, weight, year);

Console.WriteLine();

p\_p.print();

p\_p.Color = color;

p\_p.Weight = weight;

p\_p.weight++;

p\_p.print();

Console.ReadKey();

}

}

}

Листинг библиотеки:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary1

{

public class Ship

{

public string name, color, appointment;

public int weight, crew;

private int year;

public Ship()

{

name = "Элизабет";

color = "Коричневый";

appointment = "Торговый";

crew = 48;

weight = 13;

year = 1997;

}

public Ship(string name, string color, string appointment, int crew, int weight, int year)

{

this.name = name;

this.color = color;

this.appointment = appointment;

this.crew = crew;

this.weight = weight;

this.year = year;

}

public string Color

{

get { return color; }

set { color = value; }

}

public int Weight

{

get { return weight; }

set

{

weight = value;

if ((value <= 100000) && (value >= 30000))

Console.WriteLine("Средний вес");

}

}

public void print()

{

Console.WriteLine("Корабль:\nНазвание: {0};\nЦвет: {1};\nНазначение: {2};\nЧисленность экипажа: {3};\nВес: {4};\nГод выпуска: {5};\n",

name, color, appointment, crew, weight, year);

}

public static Ship operator ++(Ship obj1)

{

obj1.weight += 1;

return obj1;

}

}

}

Тестирование приложения:

Рисунок 14 – Исполнение программы

1.13 Решение задач с помощью структур данных, коллекций и классов-прототипов. Библиотечные классы коллекций

Во многих приложениях требуется создавать группы связанных объектов и управлять ими. Существует два способа группировки объектов: создать массив объектов и создать коллекцию.

Массивы удобнее всего использовать для создания и работы с фиксированным числом строго типизированных объектов.

Коллекции предоставляют более гибкий способ работы с группами объектов. В отличие от массивов, коллекция, с которой вы работаете, может расти или уменьшаться динамически при необходимости. Некоторые коллекции допускают назначение ключа любому объекту, который добавляется в коллекцию, чтобы в дальнейшем можно было быстро извлечь связанный с ключом объект из коллекции.

Коллекция является классом, поэтому необходимо объявить экземпляр класса перед добавлением в коллекцию элементов.

Если коллекция содержит элементы только одного типа данных, можно использовать один из классов в пространстве имен System.Collections.Generic. Универсальная коллекция обеспечивает строгую типизацию, так что в нее нельзя добавить другие типы данных. При извлечении элемента из универсальной коллекции не нужно определять или преобразовывать его тип данных.

Условие задачи: Разработать оконное приложение, используя коллекцию LinkedList<T> для данных гражданина страны.

Алгоритм решения: Создан класс, помещенный в коллекцию LinkedList. При открытии формы произошла инициализация объектов. Нажатие на кнопку один вызвало добавление записи, считавших информацию из текстовых полей для добавления элемента в коллекцию и DataGrid. Нажатие на кнопку два вызвало сообщение, при нажатии на которую если выражено согласие, произошло удаление элемента из коллекции и DataGrid.

Листинг формы:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApplication1337

{

public partial class Form1 : Form

{

private DataGridViewColumn dataGridViewColumn1 = null;

private DataGridViewColumn dataGridViewColumn2 = null;

private DataGridViewColumn dataGridViewColumn3 = null;

private DataGridViewColumn dataGridViewColumn4 = null;

private DataGridViewColumn dataGridViewColumn5 = null;

private LinkedList<Grazdanin> JournalList = new LinkedList<Grazdanin>();

public Form1()

{

InitializeComponent();

initDataGridView();

}

private void addJournal(string number, string name, string redaction, string author, string price)

{

Grazdanin s = new Grazdanin(number, name, redaction, author, price);

JournalList.AddLast(s);

textBox1.Text = "";

textBox2.Text = "";

textBox3.Text = "";

textBox4.Text = "";

textBox5.Text = "";

showListInGrid();

}

private void deleteJournal(int elementIndex)

{

var arr = JournalList.ToArray();

JournalList.Remove(arr[elementIndex]);

showListInGrid();

}

private void showListInGrid()

{

dataGridView1.Rows.Clear();

foreach (Grazdanin s in JournalList)

{

DataGridViewRow row = new DataGridViewRow();

DataGridViewTextBoxCell cell1 = new DataGridViewTextBoxCell();

DataGridViewTextBoxCell cell2 = new DataGridViewTextBoxCell();

DataGridViewTextBoxCell cell3 = new DataGridViewTextBoxCell();

DataGridViewTextBoxCell cell4 = new DataGridViewTextBoxCell();

DataGridViewTextBoxCell cell5 = new DataGridViewTextBoxCell();

cell1.ValueType = typeof(string);

cell1.Value = s.getNumber();

cell2.ValueType = typeof(string);

cell2.Value = s.getName();

cell3.ValueType = typeof(string);

cell3.Value = s.getRedaction();

cell4.ValueType = typeof(string);

cell4.Value = s.getAuthor();

cell5.ValueType = typeof(string);

cell5.Value = s.getPrice();

row.Cells.Add(cell1);

row.Cells.Add(cell2);

row.Cells.Add(cell3);

row.Cells.Add(cell4);

row.Cells.Add(cell5);

dataGridView1.Rows.Add(row);

}

}

private void initDataGridView()

{

dataGridView1.DataSource = null;

dataGridView1.Columns.Add(getDataGridViewColumn1());

dataGridView1.Columns.Add(getDataGridViewColumn2());

dataGridView1.Columns.Add(getDataGridViewColumn3());

dataGridView1.Columns.Add(getDataGridViewColumn4());

dataGridView1.Columns.Add(getDataGridViewColumn5());

dataGridView1.AutoResizeColumns();

}

private DataGridViewColumn getDataGridViewColumn1()

{

if (dataGridViewColumn1 == null)

{

dataGridViewColumn1 = new DataGridViewTextBoxColumn();

dataGridViewColumn1.Name = "";

dataGridViewColumn1.HeaderText = "Номер паспорта";

dataGridViewColumn1.ValueType = typeof(string);

dataGridViewColumn1.Width = dataGridView1.Width / 3;

}

return dataGridViewColumn1;

}

private DataGridViewColumn getDataGridViewColumn2()

{

if (dataGridViewColumn2 == null)

{

dataGridViewColumn2 = new DataGridViewTextBoxColumn();

dataGridViewColumn2.Name = "";

dataGridViewColumn2.HeaderText = "ФИО";

dataGridViewColumn2.ValueType = typeof(string);

dataGridViewColumn2.Width = dataGridView1.Width / 3;

}

return dataGridViewColumn2;

}

private DataGridViewColumn getDataGridViewColumn3()

{

if (dataGridViewColumn3 == null)

{

dataGridViewColumn3 = new DataGridViewTextBoxColumn();

dataGridViewColumn3.Name = "";

dataGridViewColumn3.HeaderText = "Страна";

dataGridViewColumn3.ValueType = typeof(string);

dataGridViewColumn3.Width = dataGridView1.Width / 3;

}

return dataGridViewColumn3;

}

private DataGridViewColumn getDataGridViewColumn4()

{

if (dataGridViewColumn4 == null)

{

dataGridViewColumn4 = new DataGridViewTextBoxColumn();

dataGridViewColumn4.Name = "";

dataGridViewColumn4.HeaderText = "Судимость";

dataGridViewColumn4.ValueType = typeof(string);

dataGridViewColumn4.Width = dataGridView1.Width / 3;

}

return dataGridViewColumn4;

}

private DataGridViewColumn getDataGridViewColumn5()

{

if (dataGridViewColumn5 == null)

{

dataGridViewColumn5 = new DataGridViewTextBoxColumn();

dataGridViewColumn5.Name = "";

dataGridViewColumn5.HeaderText = "Заработок";

dataGridViewColumn5.ValueType = typeof(string);

dataGridViewColumn5.Width = dataGridView1.Width / 3;

}

return dataGridViewColumn5;

}

private void button1\_Click(object sender, EventArgs e)

{

addJournal(textBox1.Text, textBox2.Text, textBox3.Text, textBox4.Text, textBox5.Text);

}

private void button2\_Click(object sender, EventArgs e)

{

int selectedRow = dataGridView1.SelectedCells[0].RowIndex;

DialogResult dr = MessageBox.Show("Убрать выделенного гражданина из таблицы?", "",

MessageBoxButtons.YesNo);

if (dr == DialogResult.Yes)

{

try

{

deleteJournal(selectedRow);

}

catch (Exception)

{

}

}

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void panel1\_Paint(object sender, PaintEventArgs e)

{

}

}

}

Тестирование приложения:

Рисунок 15 – Заполненные данные

Рисунок 16 – Демонстрация данных

Рисунок 17 – Удаление данных

Рисунок 18 – Демонстрация обновленных данных

1.14 Операции с указателями. Регулярные выражения

В .Net приложения можно развертывать простым копированием всех необходимых сборок в одну папку на жестком диске, но на практике обычно выделяются отдельные подкаталоги для связанного содержимого, например, библиотек. В файлах конфигурации .NET можно указать подкаталоги, в которых среда выполнения будет искать частные сборки при запуске приложения.

В C# определен ряд директив препроцессора, оказывающих влияние на интерпретацию исходного кода программы компилятором. Эти директивы определяют порядок интерпретации текста программы перед ее трансляцией в объектный код в том исходном файле, где они появляются. Все директивы препроцессора начинаются со знака #. Кроме того, каждая директива препроцессора должна быть выделена в отдельную строку кода.

Условие задачи: Обязательно должно быть реализовано выполнение следующих функций:

- добавления элемента в конец, начало, середину (до и после введенного номера);

- удаления элемента из начала, середины, конца;

- печати содержимого списка;

- задания конкретного варианта (поиск по заданным полям).

Для решения задач обязательно использовать функции. Все необходимые данные для функций должны передаваться им в качестве параметров. Использование глобальных переменных в функциях не допускается.

Для задания полей даты и времени логических полей, полей ввода целых, вещевственных или логических данных использовать регулярные выражения.

Учетная запись посещения спорткомплекса имеет поля: фамилия клиента, код и вид спортивного занятия, фамилия тренера, дата и время начала тренировки, количество минут, тариф. Поиск - по фамилии клиента.

Алгоритм решения: Была открыта консоль, пользователь ввел число, соответствующее необходимому номеру меню. Введены данные и помещены в класс, указатель выставлен на начальный. Осуществлен просмотр данных по указателю, перемещающемуся от начального элемента к последнему с выводом на экран. Для поиска пользователь ввел фамилию клиента и перебором каждого элемента при совпадении выводятся все данные. Добавление элемента осуществляется справа и слева от элемента по введенному ключу, далее данные заполнены данными и указатели переопределяются с добавлением нового. Удаление элемента произошло после нахождения элемента по его ключу, после чего указатели были переопределены.

Листинг программы:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Text.RegularExpressions;

namespace hyin9

{

class Tur

{

public string Name;

public string PriceDay;

public double FIO;

public double Date;

public double Days;

public double PriceTravel;

public double Course;

public DateTime Counter;

public int Key;

public Tur Next;

public Tur()

{

this.Next = null;

}

public Tur(string Name, string PriceDay, double FIO, double Date,

double Days, double PriceTravel, double Course, DateTime Counter, int Key)

{

this.Name = Name;

this.PriceDay = PriceDay;

this.FIO = FIO;

this.PriceTravel = PriceTravel;

this.Date = Date;

this.Course = Course;

this.Days = Days;

this.Counter = Counter;

this.Key = Key;

this.Next = null;

}

}

class LineList

{

public string Name;

public string PriceDay;

public double FIO;

public double Date;

public double Days;

public double PriceTravel;

public double Course;

public DateTime Counter;

public int Key;

public Tur head = null;

public Tur current = null;

public LineList() { }

public void Menu()

{

Console.Clear();

Console.WriteLine("Главное меню:\n");

Console.WriteLine(" 1. Заполнить список.");

Console.WriteLine(" 2. Просмотреть список.");

Console.WriteLine(" 3. Добавить новый элемент справа.");

Console.WriteLine(" 4. Добавить новый элемент слева.");

Console.WriteLine(" 5. Поиск элемента.");

Console.WriteLine(" 6. Удалить элемент.");

Console.WriteLine(" 7. Очистить список.");

Console.WriteLine(" 8. Выход.\n");

Console.WriteLine("\nВЫБЕРИТЕ НОМЕР НУЖНОГО ЗАДАНИЯ!\n");

}

public void FirstElem() // вставка элемента в начало списка

{

Console.Clear();

Console.Write("Введите количество сельскохозяйственных продуктов: ");

int n = int.Parse(Console.ReadLine());

for (int i = 0; i < n; i++)

{

Console.WriteLine("Заполнение {0} сельскохозяйственного продукта: \n");

Console.Write("Введите ключ: ");

Key = int.Parse(Console.ReadLine());

Console.Write("Введите наименование района: ");

Name = Console.ReadLine();

Console.Write("Введите наименование продукта: ");

PriceDay = Console.ReadLine();

Console.Write("Введите площадь: ");

FIO = double.Parse(Console.ReadLine());

Console.Write("Введите урожайность: ");

Days = double.Parse(Console.ReadLine());

Console.Write("Введите цену: ");

Date = double.Parse(Console.ReadLine());

Console.Write("Введите потери при транспартировке: ");

PriceTravel = double.Parse(Console.ReadLine());

Console.Write("Введите стоимость продукта: ");

Course = double.Parse(Console.ReadLine());

Console.Write("Введите предполагаемую дату сбора: ");

Counter = DateTime.Parse(Console.ReadLine());

string text = Convert.ToString(Key) + " " + Name + " " +

Convert.ToString(PriceDay) + " " + FIO +

" " + Convert.ToString(Days) + " " + Convert.ToString(Date) + " " +

Convert.ToString(PriceTravel) + " " + Convert.ToString(Counter) + " " + Convert.ToString(Course);

Regex theReg = new Regex(@"(?<key>\d\b\s)" +

@"(?<lnots>\^[А-ЯЁ][а-яё]+ [А-ЯЁ][а-яё]+$)" +

@"(?<sac>\^d+\b\s)" +

@"(?<kosa>\^[А-ЯЁ][а-яё]+[А-ЯЁ][а-яё]+$)" +

@"(?<lnot>\^d\b\s)" +

@"(?<dot>\d([0-9]{1,4}).\d([0-9]{1,2}).\d([0-9]{1,2})\b\s)" +

@"(?<stot>\d+\b\s)" +

@"(?<nom>\d+\b\s)" +

@"(?<rate>\d+\b)");

MatchCollection thematches1 = theReg.Matches(text);

foreach (Match sss in thematches1)

{

if (!theReg.IsMatch(Convert.ToString(sss))) { Console.WriteLine("ошибка"); };

Console.WriteLine("\ntheMatch: {0}", sss.ToString());

Console.WriteLine("key: {0}", sss.Groups["key"]);

Console.WriteLine("lnots: {0}", sss.Groups["lnots"]);

Console.WriteLine("sac: {0}", sss.Groups["sac"]);

Console.WriteLine("kosa: {0}", sss.Groups["kosa"]);

Console.WriteLine("lnot: {0}", sss.Groups["lnot"]);

Console.WriteLine("dot: {0}", sss.Groups["dot"]);

Console.WriteLine("stot: {0}", sss.Groups["stot"]);

Console.WriteLine("nom: {0}", sss.Groups["nom"]);

Console.WriteLine("rate: {0}", sss.Groups["rate"]);

}

Tur temp = new Tur(Name, PriceDay, FIO, Date, Days,

PriceTravel, Course, Counter, Key);

if (head == null)

head = temp;

else

{

temp.Next = head;

head = temp;

}

}

}

public void Show()

{

Console.Clear();

Console.WriteLine("Однонаправленный список: ");

if (head == null)

Console.WriteLine("список пуст!");

else

{

Tur temp = head;

while (temp != null)

{

Console.Write("Ключ - {0}, наименование района - {1} (где выращивают), наименование продукта - {2}, площадь - {3} (га), урожайность - {4} (кг/га), цена за 1 кг - {5}, потери при транспортировке - {6}%, стоимость продукта - {7}р., предполагаемая дата сбора - {8}\n",

temp.Key, temp.Name, temp.PriceDay, temp.FIO,

temp.Days, temp.Date, temp.PriceTravel,

temp.Course, temp.Counter); // вернуть найденный элемент

temp = temp.Next; // переход к след. элементу

}

Console.WriteLine("");

}

}

public void FindElem()

{

Console.WriteLine("Введите фамилию клиента для поиска: ");

Name = Console.ReadLine();

Tur temp = head;

Console.WriteLine("Результаты поиска: ");

while (temp != null)

{

if (temp.Name == Name)

{

Console.Write("Ключ - {0}, наименование района - {1} (где выращивают), наименование продукта - {2}, площадь - {3} (га), урожайность - {4} (кг/га), цена за 1 кг - {5}, потери при транспортировке - {6}%, стоимость продукта - {7}р., предполагаемая дата сбора - {8}\n",

temp.Key, temp.Name, temp.PriceDay, temp.FIO,

temp.Days, temp.Date, temp.PriceTravel,

temp.Course, temp.Counter); // вернуть найденный элемент

temp = temp.Next; // переход к след. элементу

}

}

}

public void Right\_Element()

{

Console.Clear();

if (head == null)

{

Console.WriteLine("Заполните список!");

}

else

{

Console.WriteLine("После какого элемента вставить новый? ");

int m = int.Parse(Console.ReadLine());

current = head;

for (int i = 1; i < m; i++)

current = current.Next;

Console.WriteLine("Заполнение {0} сельскохозяйственного продукта: \n");

Console.Write("Введите ключ: ");

Key = int.Parse(Console.ReadLine());

Console.Write("Введите наименование района: ");

Name = Console.ReadLine();

Console.Write("Введите наименование продукта: ");

PriceDay = Console.ReadLine();

Console.Write("Введите площадь: ");

FIO = double.Parse(Console.ReadLine());

Console.Write("Введите урожайность: ");

Days = double.Parse(Console.ReadLine());

Console.Write("Введите цену: ");

Date = double.Parse(Console.ReadLine());

Console.Write("Введите потери при транспартировке: ");

PriceTravel = double.Parse(Console.ReadLine());

Console.Write("Введите стоимость продукта: ");

Course = double.Parse(Console.ReadLine());

Console.Write("Введите предполагаемую дату сбора: ");

Counter = DateTime.Parse(Console.ReadLine());

Tur temp = new Tur(Name, PriceDay, FIO, Date, Days,

PriceTravel, Course, Counter, Key);

temp.Next = current.Next;

current.Next = temp;

current = temp;

}

}

public void Left\_Element()

{

Console.Clear();

if (head == null)

{

Console.WriteLine("Заполните список!");

}

else

{

Console.WriteLine("Перед каким элементом вставить новый? ");

int m = int.Parse(Console.ReadLine());

current = head;

for (int i = 1; i < m - 1; i++)

current = current.Next;

Console.WriteLine("Заполнение {0} сельскохозяйственного продукта: \n");

Console.Write("Введите ключ: ");

Key = int.Parse(Console.ReadLine());

Console.Write("Введите наименование района: ");

Name = Console.ReadLine();

Console.Write("Введите наименование продукта: ");

PriceDay = Console.ReadLine();

Console.Write("Введите площадь: ");

FIO = double.Parse(Console.ReadLine());

Console.Write("Введите урожайность: ");

Days = double.Parse(Console.ReadLine());

Console.Write("Введите цену: ");

Date = double.Parse(Console.ReadLine());

Console.Write("Введите потери при транспартировке: ");

PriceTravel = double.Parse(Console.ReadLine());

Console.Write("Введите стоимость продукта: ");

Course = double.Parse(Console.ReadLine());

Console.Write("Введите предполагаемую дату сбора: ");

Counter = DateTime.Parse(Console.ReadLine());

Tur temp = new Tur(Name, PriceDay, FIO, Date, Days,

PriceTravel, Course, Counter, Key);

temp.Next = current.Next;

current.Next = temp;

current = temp;

}

}

public void Delete\_Elements()

{

Console.Clear();

if (head != null)

{

Show();

Tur temp = head;

Tur prev = head;

int i = 1;

Console.WriteLine("Введите позицию в списке (число): ");

int pos = int.Parse(Console.ReadLine());

while (pos != i)

{

prev = temp;

temp = temp.Next;

i++;

}

if (head == temp)

head = head.Next;

else

prev.Next = temp.Next;

}

else

Console.WriteLine("Список и так пустой! Удалять нечего.");

}

public void Clear\_list()

{

Tur temp;

while (head != null)

{

temp = head.Next;

head = temp;

}

}

}

class Program

{

static void Main(string[] args)

{

LineList List = new LineList();

List.Menu();

int c = int.Parse(Console.ReadLine());

while (c != 8)

{

switch (c)

{

case 1: List.FirstElem(); break;

case 2: List.Show(); break;

case 3: List.Right\_Element(); break;

case 4: List.Left\_Element(); break;

case 5: List.FindElem(); break;

case 6: List.Delete\_Elements(); break;

case 7: List.Clear\_list(); break;

case 8: break;

default: Console.Write("Команды с таким номером нет!"); break;

}

do Console.Write("\nДля продолжения нажмите клавишу Enter....");

while (Console.ReadKey().Key !=

ConsoleKey.Enter);

List.Menu();

c = int.Parse(Console.ReadLine());

}

}

}

}

Тестирование приложения:

Рисунок 19 – Исполнение программы

2 Разработка Windows-приложений средствами библиотеки классов Microsoft .NET Framework

Оконные приложения строятся по принципам событийно-управляемого программирования (event-driven programming) - стиля программирования, при котором поведение компонента системы определяется набором возможных внешних событий и ответных реакций компонента на них. Такими компонентами в Windows являются окна.

С каждым окном в Windows связана определенная функция обработки событий – оконная функция. События для окон называются сообщениями. Сообщение относится к тому или иному типу, идентифицируемому определенным кодом (32-битным целым числом), и сопровождается парой 32-битных параметров (WPARAM и LPARAM), интерпретация которых зависит от типа сообщения.

Задача любого оконного приложения - создать главное окно и сообщить Windows функцию обработки событий для этого окна. Все самое интересное для приложения будет происходить именно в функции обработки событий главного окна.

В Windows программа пассивна. После запуска она ждет, когда ей уделит внимание операционная система. Операционная система делает это посылкой сообщений. Сообщения могут быть разного типа, они функционируют в системе достаточно хаотично, и приложение не знает, какого типа сообщение придет следующим. Логика построения Windows-приложения должна обеспечивать корректную и предсказуемую работу при поступлении сообщений любого типа.

Условие задачи 1: Создать оконное приложение для вычисления значений функций:

|  |  |
| --- | --- |
|  |  |

Алгоритм решения: Кнопка 1 преобразовывает текстовую информацию, введенную пользователем в вещественные значения. Исходя из условия вычисляется соответствующее значение. Предусмотрены исключительные ситуации. Найденные значения записываются в таблицу и текстовые поля. Кнопка 2 позволяет закрыть программу, кнопка 3 – переход к следующему заданию. Компонент PictureBox отображает задание.

Открылась форма, пользователь заполнил данные и нажал на «Выполнить задание». На экране отобразился результат.

Листинг формы:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

double xn, xk, xh, x, y, a, ymax, ymin, yt;

int n, i;

if ((textBox3.Text != "") && (textBox4.Text != "") && (textBox5.Text != "") && (textBox6.Text != ""))

{

xn = Convert.ToDouble(textBox3.Text);

xk = Convert.ToDouble(textBox4.Text);

xh = Convert.ToDouble(textBox5.Text);

a = Convert.ToDouble(textBox6.Text);

dataGridView1.Columns.Clear();

dataGridView1.ColumnCount = 2;

dataGridView1.Rows.Add(Math.Ceiling((xk - xn) / xh) + 1);

dataGridView1.Columns[0].Name = " X";

dataGridView1.Columns[1].Name = " Y";

i = 0;

x = xn;

ymax = -1.8e307;

ymin = 1.8e307;

while (x <= xk)

{

if (x <= 0)

{

y = 1.53 \* Math.Pow(x, 2);

}

else if (x <= a)

{

try

{

if (Math.Log10(x \* a) < 0)

{

throw new Exception();

}

y = x + (Math.Log10(x \* a));

}

catch (Exception)

{

dataGridView1.Rows[i].Cells[0].Value = Convert.ToString(x);

dataGridView1.Rows[i].Cells[1].Value = "Область недопустимых значений";

x += xh;

i++;

dataGridView1.Rows.Add(1);

continue;

}

}

else

{

y = 1 / Math.Sqrt(x + 2.3 \* a);

}

dataGridView1.Rows[i].Cells[0].Value = Convert.ToString(x);

yt = Math.Ceiling(y \* 100) / 100;

dataGridView1.Rows[i].Cells[1].Value = Convert.ToString(yt);

string str = Convert.ToString(dataGridView1.Rows[i].Cells[1].Value);

if (str == "NaN")

{

dataGridView1.Rows[i].Cells[1].Value = "Требуются доп. исследования";

}

if (y > ymax)

ymax = Math.Ceiling(y \* 100) / 100;

if (y < ymin)

ymin = Math.Ceiling(y \* 100) / 100;

x += xh;

i++;

dataGridView1.Rows.Add(1);

}

textBox1.Text = Convert.ToString(ymax);

textBox2.Text = Convert.ToString(ymin);

dataGridView1.Rows.RemoveAt(i);

}

else

{

MessageBox.Show("Заполните, пожалуйста, данные", "Ошибка ввода данных",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

}

private void button2\_Click(object sender, EventArgs e)

{

Application.Exit();

}

}

}

Тестирование приложения:

Рисунок 20 – Демонстрация рабочего окна

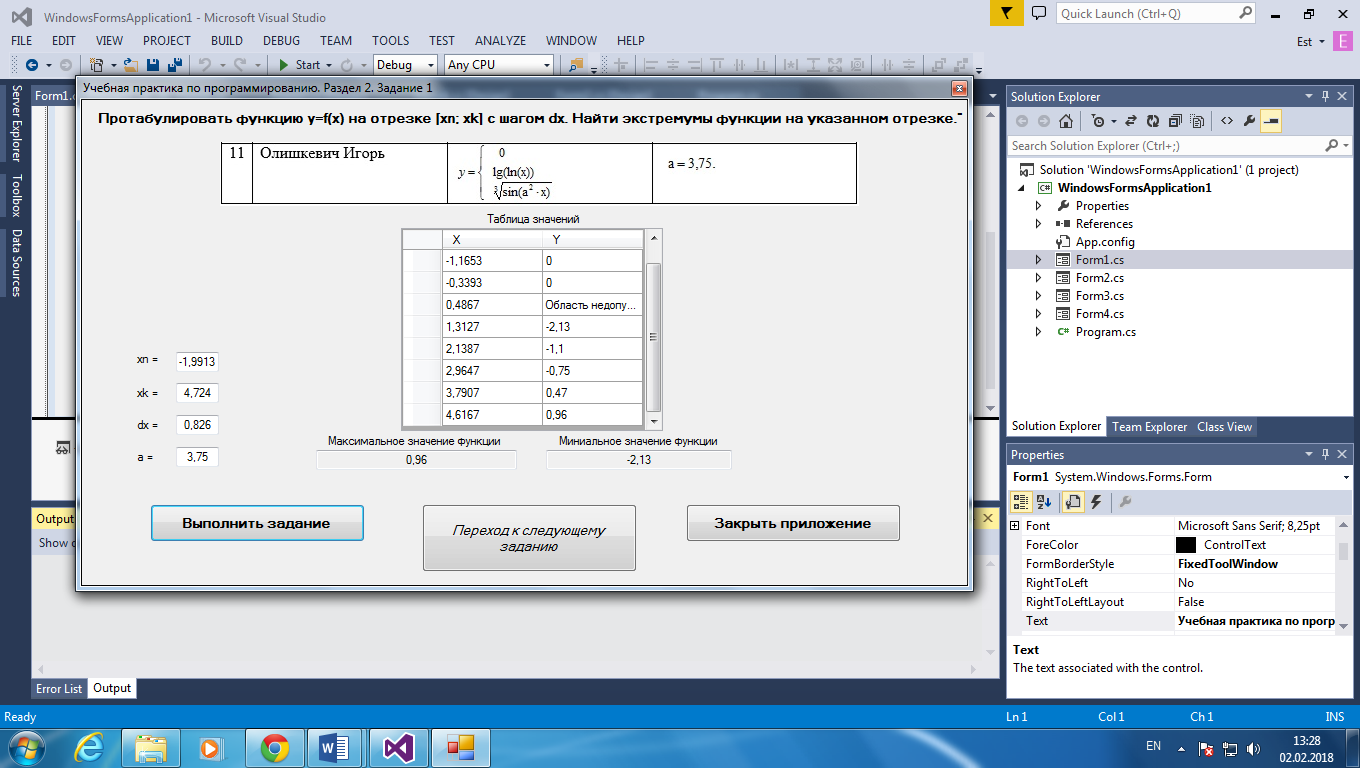


Рисунок 21 – Демонстрация полученных значений

Условие задачи 2: Создать оконное приложение для вычисления значений функций:

1. Вычислить среднегеометрическое значение четных по значению элементов последнего столбца массива.

2. Определить наибольший по абсолютной величине среди ненулевых элементов первой строки массива.

Алгоритм решения: Кнопка 1 преобразовывает текстовую информацию, введенную пользователем в целые значения, затем создает таблицу соответствующего размера. Кнопка 3 считывает значения из таблицы, затем производит вычисления. Предусмотрены исключительные ситуации. Найденные значения записываются в таблицу и текстовые поля. Кнопка 2 позволяет закрыть программу, кнопка 4 – переход к форме «Справка». Компонент TabControl отображает задание на первой вкладке, на второй позволяет выполнить задание. Компонент MainMenu по клику на вкладку «О раработчике» открывает форму с контактной информацией исполнителя.

Открылась форма, пользователь открыл таблицу, заполнил данные и нажал на кнопку выполнения. На экране отобразился результат.

Листинг основной формы:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form2 : Form

{

public int n, m;

public Form2()

{

InitializeComponent();

}

private void button2\_Click(object sender, EventArgs e)

{

Application.Exit();

}

private void button1\_Click(object sender, EventArgs e)

{

if ((textBox1.Text != "") && (textBox2.Text != ""))

{

m = Convert.ToInt32(textBox1.Text);

n = Convert.ToInt32(textBox2.Text);

dataGridView1.Columns.Clear();

dataGridView1.ColumnCount = n;

dataGridView1.RowCount = m;

}

else

{

MessageBox.Show("Заполните, пожалуйста, данные", "Ошибка ввода данных",

MessageBoxButtons.OK, MessageBoxIcon.Exclamation);

}

}

private void button4\_Click(object sender, EventArgs e)

{

this.Hide();

Form4 f4 = new Form4();

f4.ShowDialog();

}

private void toolStripMenuItem2\_Click(object sender, EventArgs e)

{

this.Hide();

Form3 f3 = new Form3();

f3.ShowDialog();

}

private void button3\_Click(object sender, EventArgs e)

{

int count = 0;

double sum = 0, sr = 0;

int size = Convert.ToInt32(Math.Round(Convert.ToDouble(m) / 4)) + 1;

double max = -1000000;

double[,] A = new double[m, n];

for (int i = 1; i <= m; i++)

for (int j = 1; j <= n; j++)

{

A[i - 1, j - 1] = Convert.ToDouble(this.dataGridView1.Rows[i - 1].Cells[j - 1].Value);

if (A[i - 1, j - 1] % 2 == 0 && j == n)

{

sum \*= Math.Pow(A[i - 1, j - 1], 2);

count++;

}

if ((i == 1) && (A[i - 1, j - 1] != 0) && (max < A[i - 1, j - 1]))

{

max = Math.Abs(A[i - 1, j - 1]);

}

}

sr = Math.Pow(sum, 1.0 / count);

if (checkBox1.Checked == true)

{

this.textBox3.Text = Convert.ToString(sr);

}

else

if (checkBox1.Checked == true)

{

this.textBox3.Text = Convert.ToString("нет элементов");

}

if (checkBox2.Checked == true)

{

this. textBox4.Text = Convert.ToString(max);

}

else

if (checkBox2.Checked == true)

{

this.textBox4.Text = Convert.ToString("нет элементов");

}

return;

}

}

}

Листинг формы контактной информации разработчика:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form3 : Form

{

public Form3()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

this.Hide();

Form2 f2 = new Form2();

f2.Show();

}

}

}

Листинг формы справки:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form4 : Form

{

public Form4()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

this.Hide();

Form2 f2 = new Form2();

f2.Show();

}

}

}

Тестирование приложения:

Рисунок 26 – Демонстрация условия

Рисунок 27 – Демонстрация выполнения

Рисунок 28 – Демонстрация формы «О разработчике»

Рисунок 29 – Демонстрация формы «Справка»