**ЗАДАНИЕ 18**

Задача 1. Решить задачи с использованием класса Stack. Пусть символ # определен в текстовом редакторе как стирающий символ Backspace, т.е. строка abc#d##c в действительности является строкой ac. Дан текст, в котором встречается символ #. Преобразовать его с учетом действия этого символа.

Листинг программы:

namespace Task1

{

public class Program

{

public static void Main()

{

string s = "abc#d##c";

Console.WriteLine(ProcessString(s));

}

public static string ProcessString(string s)

{

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

foreach (char c in s)

{

if (c != '#')

{

stack.Push(c);

}

else if (stack.Count > 0)

{

stack.Pop();

}

}

return string.Join("", stack.Reverse().ToArray());

}

}

}

Таблица 18.1 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| abc#d##c | ac |

Анализ результатов:



Рисунок 18.1 – Результат работы программы

Задание 2. Решить следующие задачи с использованием класса Queue. Дан файл, содержащий числа. За один просмотр файла напечатать элементы файла в следующем порядке: сначала все числа, из интервала [a,b], потом все числа, меньшие a, потом все числа, большие b, сохраняя исходный порядок в каждой группе чисел.

Листинг программы:

namespace Task2

{

public class Program

{

public static void Main()

{

int a = 10, b = 20;

var queue1 = new Queue<int>();

var queue2 = new Queue<int>();

var queue3 = new Queue<int>();

foreach (var number in File.ReadLines("numbers.txt").Select(int.Parse))

{

if (number >= a && number <= b)

queue1.Enqueue(number);

else if (number < a)

queue2.Enqueue(number);

else

queue3.Enqueue(number);

}

File.WriteAllLines("output.txt", queue1.Concat(queue2).Concat(queue3).Select(n => n.ToString()));

}

}

}

Таблица 18.2 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| Задаются программой | Выписыватся в файл output.txt |

Анализ результатов:

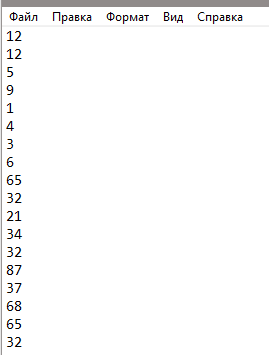


Рисунок 18.2 – Результат работы программы

Задание 3. Создать класс-прототип, с обобщенным методом, с методами добавления,

удаления, клонирования экземпляров класса.

В основной программе продемонстрировать:

функциональность созданного класса и всех его методов;

использование обнуляемых типов, использование абстрактных структур.

Листинг программы:

using System;

using System.Collections.Generic;

public abstract class Plant

{

public string Species { get; set; } // Вид

public double Height { get; set; } // Высота

public string Color { get; set; } // Цвет

public PlantType Type { get; set; } // Тип

public enum PlantType

{

Flower,

Tree,

Bush

}

public Plant(string species, double height, string color, PlantType type)

{

Species = species;

Height = height;

Color = color;

Type = type;

}

public override string ToString() => $"{Species} ({Height} м, {Color}, {Type})";

public abstract Plant Clone();

}

// Класс "Цветок"

public class Flower : Plant

{

public Flower(string species, double height, string color) : base(species, height, color, PlantType.Flower) { }

public override Plant Clone() => new Flower(Species, Height, Color);

}

// Класс "Дерево"

public class Tree : Plant

{

public Tree(string species, double height, string color) : base(species, height, color, PlantType.Tree) { }

public override Plant Clone() => new Tree(Species, Height, Color);

}

// Класс "Кустарник"

public class Bush : Plant

{

public Bush(string species, double height, string color) : base(species, height, color, PlantType.Bush) { }

public override Plant Clone() => new Bush(Species, Height, Color);

}

// Класс-прототип "Коллекция растений"

public class PlantCollection<T> where T : Plant

{

private List<T> plants = new List<T>();

public void AddPlant(T plant)

{

plants.Add(plant);

}

public List<T> FindPlants(Predicate<T> predicate)

{

return plants.FindAll(predicate);

}

public void SortPlants(Comparison<T> comparison)

{

plants.Sort(comparison);

}

public void PrintPlants(TextWriter writer)

{

foreach (var plant in plants)

{

writer.WriteLine(plant);

}

}

}

class Program

{

static void Main()

{

PlantCollection<Plant> plantCollection = new PlantCollection<Plant>();

// Добавление растений

plantCollection.AddPlant(new Flower("Роза", 0.5, "Красный"));

plantCollection.AddPlant(new Tree("Дуб", 20, "Зеленый"));

plantCollection.AddPlant(new Bush("Сирень", 3, "Фиолетовый"));

// Поиск цветов по цвету

List<Plant> redFlowers = plantCollection.FindPlants(plant => plant.Color == "Красный");

// Сортировка деревьев по высоте

plantCollection.SortPlants((t1, t2) => t1.Height.CompareTo(t2.Height));

// Печать информации о растениях в консоль

plantCollection.PrintPlants(Console.Out);

// Клонирование растения

Plant roseClone = redFlowers[0].Clone();

Console.WriteLine($"Клон розы: {roseClone}");

}

}

Таблица 18.3 – Входные и выходные данные

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| Задаются программой |  |

Анализ результатов:

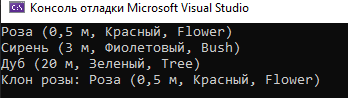


Рисунок 18.3– Результат работы программы