МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Лабораторна робота № 3**

з дисципліни « Алгоритми і структури даних »

*назва дисципліни*

на тему:« ДВОРІВНЕВІ СТРУКТУРИ. СПИСКИ СПИСКІВ. ВБУДОВАНІ КОЛЕКЦІЇ»

Виконав: студент 2 курсу групи № 622п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Зайченко Ярослав Ігорович

(прізвище й ініціали студента)

Прийняв: канд. техн. наук, доцент на кафедрі 603 Волобуєва Ліна Олексіївна

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2023

**Мета роботи** − вивчення вбудованих типів вбудованих колекцій в узагальненій та в неузагальненій реалізаціях на прикладі створення списку підсписків.

Постановка задачі

1. Описати вказаний абстрактний тип даних список підсписків і основні операції роботи з ним на абстрактному і логічному рівнях:

* початкова ініціалізація структури (список порожній).
* додавання елементу в список/підсписок
* видалення елементу зі списку/підсписку.
* перегляд першого елементу в списку/підсписку.
* перевірка списку/підсписку- порожній він або не порожній
* виведення структури на екран.

1. Відповідно до варіанту створити структуру даних для організації дворівневої структури даних (списки підсписків).
2. Для реалізації структури використовувати вбудовані типи в узагальненій і неузагальнених реалізаціях (Stack, Stack , Queue, Queue , ArrayList, List , LinkedList). Як елементи структур використати наступні типи: (class або struct), що мають поле ключа та поле інформації для підсписків (нижній рівень) і поле ключа для списку (верхній рівень).
3. Створити додаток, який демонструє роботу операцій, перелічених вище (див. п.1.). Для створення додатку дозволяється використовувати лише ті операції, що притаманні заданій структурі. Наприклад, заборонено отримувати доступ до елементу із довільною позицією у черзі.
4. Оформити звіт.

Зміст звіту

Зміст звіту з лабораторної роботи №3 має включати наступні підрозділи:

1. Абстрактне представлення структури.
2. Представлення даних в графічному виді.
3. Проектування необхідних класів і структур.
4. Алгоритм для кожного методу, використовуючи один з наступних видів представлення алгоритмів: псевдокод або блок-схему.
5. Лістинг програми та екранні форми результату роботи програми.
6. Висновки

Варіант 9 індивідуального завдання до лабораторної роботи 3:

Список типу ArrayList та підсписки Stack <T>

хід роботи

Опис програми:

Мова програмування: С#, операційна система Windows 11 Prо, Версія 23H2, Збірка ОС 22621.1325, процесор: Apple Silicon M1 Pro 3.20 GHz (ядер: 4), компілятор: Microsoft Visual Studio Community 2022 (64-розрядна версія ARM).

Абстрактне представлення структури

Стек (Stack) – АТД з доступом до елементів за правилом «останній прийшов – першим вийшов» (Last In – First Out, LIFO)

Опис АТД стек:

Top - тип стек, el – елемент стеку.

Операції:

* 1. Створити (New):  → Up\_Sp
  2. Додавання елементу в підсписок (Push): Up\_Sp x i x D\_St → Up\_Sp;
  3. Додавання елементу в список (Add): Up\_Sp x D\_St → Up\_Sp
  4. Видалення елементу з підсписку (Pop): Up\_Sp x i x D\_St → Up\_Sp;
  5. Видалення елементу зі списку (Remove(At)): Up\_Sp x i → Up\_Sp;
  6. Перший елемент в підсписку (Peek): Up\_Sp x i x D\_St → El\_Down (int)

Графічне представлення структури

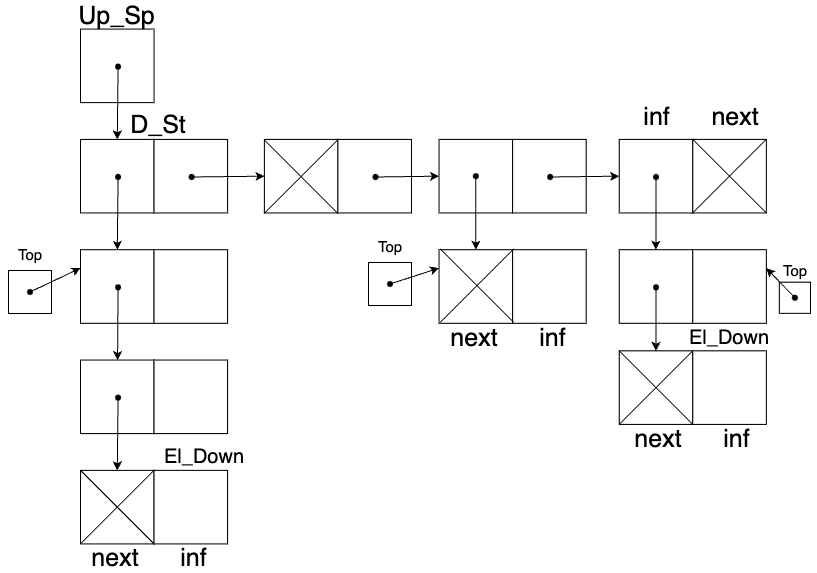


Рисунок 1 – Структура зі списком ArrayList та підсписком Stack <T>

Проектування необхідних класів і структур.

Клас El\_Down потрібен для ініціалізації елементів в підсписку; містить інформацію та наступний елемент.

public class El\_Down // елемент підсписку

{

public int key;

public string inf;

public El\_Down(int key, string inf)

{

this.key = key;

this.inf = inf;

}

public override string ToString()

{

string res = "Значення=" + key.ToString() + "; inf=" + inf + " ";

return res;

}

}

Клас Down містить в собі інформацію про перший та останній елементи стеку. Також містить методи для реалізації дій із даними, які зазначені у постановці завдання (фрагмент коду наведено нижче).

public class MyStack

{

Elem \_Top;

public bool Push(Elem el)

public bool Pop()

public Tuple<MyStack, MyStack> SplitStack()

public override string ToString()

public bool IsEmpty()

public int CountEl()

public Elem Peek()

}

Методи зображені у вигляді блок-схем

(Див. рисунки 4 –10). Вони мають назви відповідно до їхнього функціоналу.

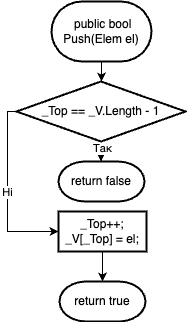


Рисунок 4 – Додавання символу в стек

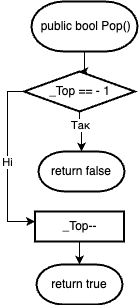


Рисунок 5 – Метод для видалення символів зі стеку

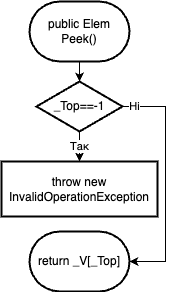


Рисунок 6 – Знаходження першого елементу

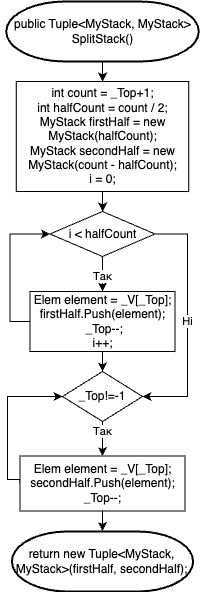


Рисунок 7 – Розбиття списку на два рівні циклічних підсписки

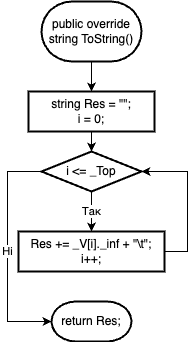


Рисунок 8 – Метод для перетворення елементів стеку в рядковий тип

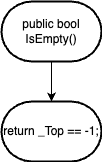


Рисунок 9 – Перевірка наповненості стека

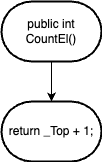


Рисунок 10 – Кількість елементів в стеку

Тестування програми

Додавання елементів (1 a 2 b 3 c) до стека:

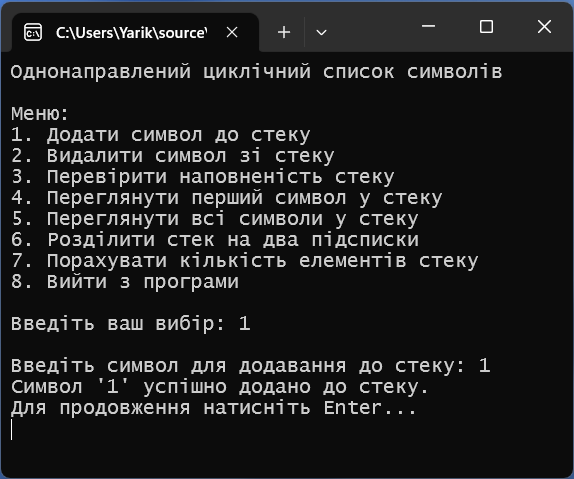


Рисунок 11 – Процес додавання елементів до стеку

Переглянемо створений стек символів. Рисунок 10.

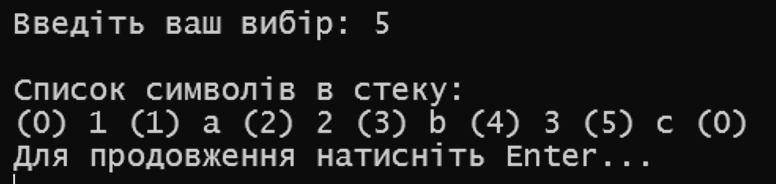


Рисунок 12 – Виведення на екран створеного стеку символів

Видаляємо перший елемент в стеку та виводимо його знову на екран (Рисунки 13 - 14)

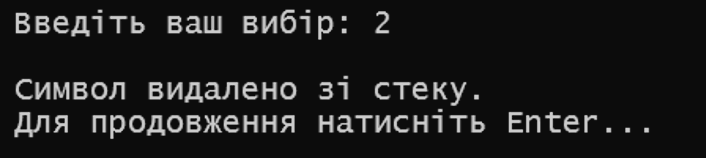


Рисунок 13 – Видалення першого елементу в стеку

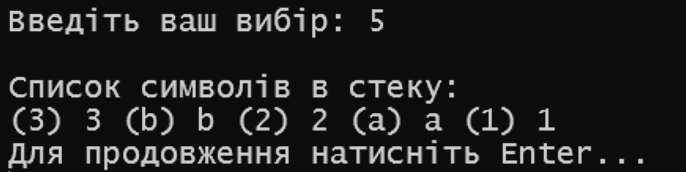


Рисунок 14 – Оновлений стек з видаленим першим елементом

Розіб’ємо стек на 2 підсписки. Результат на Рисунку 15

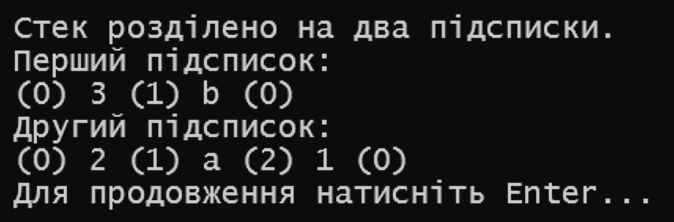


Рисунок 15 – Виведені на екран елементи двох підсписків стеку

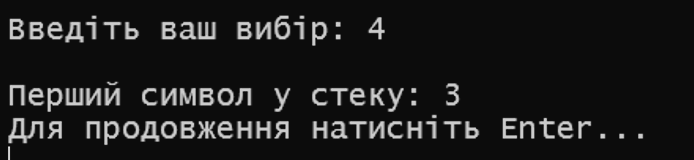


Рисунок 16 – Виведений на екран перший елемент стеку без видалення

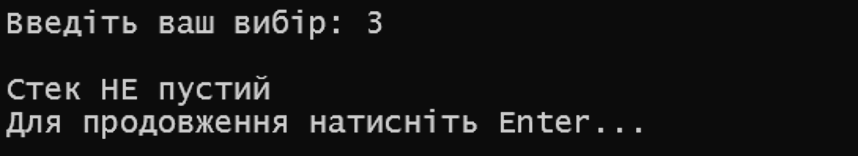


Рисунок 17 – Виведений на екран результат перевірки наповненості стеку

Порахуємо кількість елементів в списку. Результат на Рисунку 14.

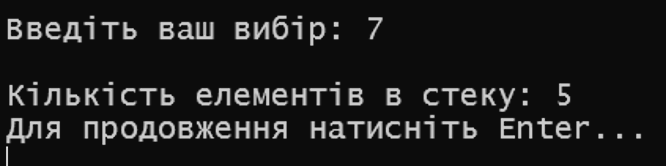


Рисунок 18 – кількість елементів в черзі

Результат співпадає з ручним рахунком.

Після видалення всіх елементів стеку при перевірці наповнення програма повідомляє, що стек порожній. Демонстрація на рисунках 19-20.

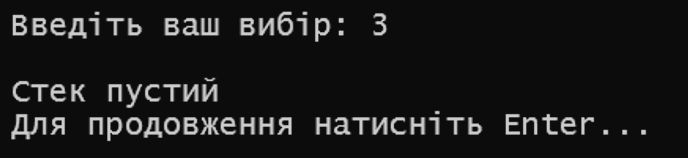


Рисунок 19 – перевірка елементів в пустій черзі

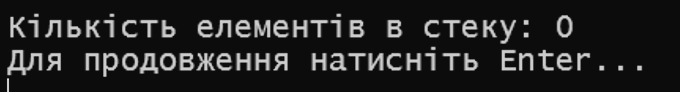


Рисунок 20 – перевірка кількості елементів в пустому стеку

ВИСНОВОК

У ході виконання лабораторної роботи я:

* Ознайомився з базовими структурами даних;
* Отримав навички програмування алгоритмів, що їх оброблюють;
* Навів приклади роботи зі стеком;
* Написав програму для виконання операцій зі стеком;
* Виконав тестування написаної програми;
* Написав звіт.

ДОДАТОК А

Машинний лістинг програми

Program.cs:

using System;

using System.Text;

namespace ASD\_Lab\_2\_vector\_

{

internal class Program

{

static void PressEnter()

{

while (true)

{

Console.WriteLine("Для продовження натисніть Enter...");

var key = Console.ReadKey(intercept: true);

if (key.Key == ConsoleKey.Enter)

{

Console.Clear();

break; // Выход из цикла, если нажата клавиша Enter

}

else Console.WriteLine("Була натиснута інша клавіша");

}

}

static void Main(string[] args)

{

Encoding.RegisterProvider(CodePagesEncodingProvider.Instance);

Console.InputEncoding = System.Text.Encoding.GetEncoding(1251);

Console.OutputEncoding = System.Text.Encoding.GetEncoding(1251);

MyStack stack = new MyStack(10); // Создаем стек с начальной емкостью 10

Console.WriteLine("Однонаправлений циклічний список символів\n");

while (true)

{

Console.WriteLine("Меню:");

Console.WriteLine("1. Додати символ до стеку");

Console.WriteLine("2. Видалити символ зі стеку");

Console.WriteLine("3. Перевірити наповненість стеку");

Console.WriteLine("4. Переглянути перший символ у стеку");

Console.WriteLine("5. Переглянути всі символи у стеку");

Console.WriteLine("6. Розділити стек на два підсписки");

Console.WriteLine("7. Порахувати кількість елементів стеку");

Console.WriteLine("8. Вийти з програми");

Console.Write("\nВведіть ваш вибір: ");

char mode = Console.ReadKey().KeyChar;

Console.WriteLine("\n");

switch (mode)

{

case '1':

Console.Write("Введіть символ для додавання до стеку: ");

char input = Console.ReadKey().KeyChar;

Elem newElem = new Elem(input);

if (stack.Push(newElem))

{

Console.WriteLine($"\nСимвол '{input}' успішно додано до стеку.");

}

else

{

Console.WriteLine("\nСтек переповнений. Неможливо додати символ.");

}

PressEnter();

break;

case '2':

if (stack.Pop())

{

Console.WriteLine("Символ видалено зі стеку.");

}

else

{

Console.WriteLine("Стек порожній. Неможливо видалити символ.");

}

PressEnter();

break;

case '3':

if (stack.IsEmpty())

Console.WriteLine("Стек пустий");

else Console.WriteLine("Стек НЕ пустий");

PressEnter();

break;

case '4':

try

{

Elem firstElement = stack.Peek();

Console.WriteLine("Перший символ у стеку: " + (char)firstElement.\_inf);

}

catch (InvalidOperationException)

{

Console.WriteLine("Стек порожній. Неможливо переглянути перший символ.");

}

PressEnter();

break;

case '5':

Console.WriteLine("Список символів в стеку:");

Console.WriteLine(stack.Print());

PressEnter();

break;

case '6':

Tuple<MyStack, MyStack> splitResult = stack.SplitStack();

MyStack firstHalf = splitResult.Item1;

MyStack secondHalf = splitResult.Item2;

Console.WriteLine("Стек розділено на два підсписки.");

Console.WriteLine("Перший підсписок:");

Console.WriteLine(firstHalf.ToString());

Console.WriteLine("Другий підсписок:");

Console.WriteLine(secondHalf.ToString());

PressEnter();

break;

case '7':

Console.WriteLine("Кількість елементів в стеку: " + stack.CountEl());

PressEnter();

break;

case '8':

Environment.Exit(0);

break;

default:

Console.WriteLine("Невідома опція. Будь ласка, виберіть інший варіант.");

PressEnter();

break;

}

}

}

}

}

ДОДАТОК Б

Машинний лістинг створеного класу

MyStack.cs

using System;

namespace ASD\_Lab\_2\_vector\_

{

public struct Elem

{

public char \_inf;

public Elem(char inf)

{

this.\_inf = inf;

}

}

public class MyStack

{

private int \_Top;

private Elem[] \_V;

public MyStack(int count)

{

\_Top = -1;

\_V = new Elem[count];

}

public bool Push(Elem el)

{

if (\_Top == \_V.Length - 1)

{

return false; // Стек переполнен

}

\_Top++;

\_V[\_Top] = el;

return true;

}

public bool Pop()

{

if (\_Top == -1)

{

return false; // Стек пуст

}

\_Top--;

return true;

}

public override string ToString()

{

string Res = "";

for (int i = 0; i <= \_Top; i++)

{

Res += \_V[i].\_inf + "\t";

}

return Res;

}

public string Print()

{

if (\_Top == -1)

{

return "Стек пустий";

}

string Res = "";

for (int i = 0; i <= \_Top; i++)

{

Res += $"({\_V[i].\_inf}) {\_V[i].\_inf} ";

}

Res += $"({\_V[0].\_inf}) {\_V[0].\_inf}"; // Последний элемент указывает на первый

return Res;

}

public string Print1()

{

if (\_Top == -1)

{

return "Стек пустий";

}

string Res = "";

for (int i = \_Top; i >= 0; i--)

{

Res += $"({\_V[i].\_inf}) {\_V[i].\_inf} ";

}

return Res;

}

public Elem Peek()

{

if (\_Top == -1)

{

throw new InvalidOperationException("Стек пуст. Невозможно выполнить Peek.");

}

return \_V[\_Top];

}

public bool IsEmpty()

{

return \_Top == -1;

}

public int CountEl()

{

return \_Top + 1;

}

public Tuple<MyStack, MyStack> SplitStack()

{

if (IsEmpty())

{

// Стек пуст, возвращаем два пустых стека

return new Tuple<MyStack, MyStack>(new MyStack(0), new MyStack(0));

}

int count = CountEl();

int halfCount = count / 2;

MyStack firstHalf = new MyStack(halfCount);

MyStack secondHalf = new MyStack(count - halfCount);

// Перемещаем элементы из исходного стека в первую половину

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

{

Elem element = \_V[\_Top]; // Извлекаем элемент из вершины стека

firstHalf.Push(element); // Добавляем элемент в первую половину

\_Top--; // Уменьшаем указатель вершины

}

// Перемещаем оставшиеся элементы из первой половины во вторую половину

while (!IsEmpty())

{

Elem element = \_V[\_Top]; // Извлекаем элемент из вершины стека

secondHalf.Push(element); // Добавляем элемент во вторую половину

\_Top--; // Уменьшаем указатель вершины

}

return new Tuple<MyStack, MyStack>(firstHalf, secondHalf);

}

}

}