# Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 6 з дисципліни

«Основи програмування-2.

Базові конструкції»

# «ДЕРЕВА»

Варіант 20

Виконав студент Лошак Віктор Іванович

(шифр, прізвище, ім'я, по батькові)

Перевірив Вітковська Ірина Іванівна

( прізвище, ім'я, по батькові)

Київ 2022

**Лабораторна робота 6**

**Мета:**

Вивчити особливості організації і обробки дерев.

**Тема:** Текстовий файл містить програму мовою С++. Надрукувати в порядку зростання номери всіх рядків, що містять кожен ідентифікатор програми. Для збереження ідентифікаторів використати структуру типу дерева, елементами якого є ідентифікатор і номер рядка де вони зустрічаються.

**Математична модель:**

Для імплементації механізму визначення ідентифікаторів в програмі використаємо метод GetSqueezedStr що прийматиме в якості параметрів два об’єкти індексованого типу з елементами типу string де перший масив відповідає можливим початкам шуканого підрядка а други можлививм закінченням. Оскільки ми знаємо синтаксис що відповідає оголошенню користувацького типу, використаємо цей метод також для знаходження користувацьких типів що були введені користувачем. Для коректного зчитування чимволів з клавіатури використаємо метод ReadMultilineInp. Для визначення пар рядок- ідентифікатори застосуємо метод GetLineIdentifyerTree що визначає наявність конкретно ідентифікатора а не випадкового співпадіння за допомогою перевірки символів що знаходяться справа та зліва від субрядка що є підозрюваним. Для створення структури типу дерево і зберігання знайдених пар використаємо клас BinaryNode що міститиме посилання на двох нащадків а також поля значень Key, Value які відповідають ключу та значенню вузла дерева відповідно. Для знаходження масиву послідовно заповненого елементами дерева за зростанням використаємо алгоритм серединного обходу бінарного дерева.

**1)Приклад коду на C#**

**Program.cs**

namespace Lab6 // Note: actual namespace depends on the project name.

{

internal class Program

{

static void Main(string[] args)

{

//reading input

string fileName = "SomeProgram";

Console.WriteLine("Enter the code of program in C/C++ (do not create identifiers with the same name but different nested level) :");

List<string> input = new List<string>(ReadMultilineInp());

Console.WriteLine();

//writing input to file

using (StreamWriter writer = new StreamWriter(File.Open(fileName, FileMode.Create)))

{

foreach (string line in input)

{

writer.WriteLine(line);

}

}

//input.Clear();

//using (StreamReader reader = new StreamReader(File.Open(fileName, FileMode.Open)))

//{

// while(!reader.EndOfStream)

// {

// input.Add(reader.ReadLine());

// }

//}

//creating the list of types(the word that goes before identifyer in it`s definition is type)

List<string> identifyerStartDef = new List<string>

{

"int ",

"long ",

"double ",

"float ",

"bool ",

"char ",

"string ",

"auto ",

"void "

};

//creating the list of strings that go after identidyer in it`s definition

List<string> identifyerEndDef = new List<string> { "=", ";", "(", };

//spotting custom types in the input

List<string> typesStartDef = new List<string> { "class", "struct", "enum", };

List<string> CustomTypes = GetSqueezedStr(input, typesStartDef, new List<string>());

foreach (var item in CustomTypes)

{

Console.WriteLine($"Type: {item}");

}

//adding custom types to all other types

identifyerStartDef.AddRange(CustomTypes);

//adding pointers to all other types

int ptrDimensAbleToRead = 5;

int identifyerStartDefLength = identifyerStartDef.Count;

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

{

for (int j = 0; j < ptrDimensAbleToRead; j++)

{

identifyerStartDef.Add(identifyerStartDef[i] + new String('\*', j) + " ");

}

}

//getting all identifyers in program

List<string> allIdentifyers = GetSqueezedStr(new List<string>(input), identifyerStartDef, identifyerEndDef);

foreach (var item in allIdentifyers)

{

Console.WriteLine($"Identifyer: {item}");

}

//generating Binary tree drom given identifyers and printing it

BinaryNode root = GetLineIdentifyerTree(input, allIdentifyers);

List<BinaryNode> ascendingNodes = new List<BinaryNode>();

GetTreeNodesAscending(root, ascendingNodes);

for (int i = 0; i < ascendingNodes.Count; i++)

{

Console.Write($"String number: {ascendingNodes[i].Key}\tIdentifyers: ");

foreach (var str in ascendingNodes[i].Value)

{

Console.Write(str + " ");

}

Console.WriteLine();

}

}

public static void GetTreeNodesAscending(BinaryNode root, List<BinaryNode> ascendingNodes)

{

if (root.LeftChild != null)

{

GetTreeNodesAscending(root.LeftChild, ascendingNodes);

}

ascendingNodes.Add(root);

if(root.RightChild != null)

{

GetTreeNodesAscending(root.RightChild, ascendingNodes);

}

}

private static BinaryNode GetLineIdentifyerTree(List<string> input, List<string> allIdentifyers)

{

BinaryNode root = new BinaryNode();

allIdentifyers = new List<string>(allIdentifyers.OrderByDescending(p => p.Length).Select(p => p.Trim())); //sorting and trimming strings of identifyers

for (int i = 0; i < input.Count; i++)// for every string in input

{

input[i] = input[i].Trim();

for (int j = 0; j < allIdentifyers.Count(); j++)//we look for the first substring in the string which corresponds to any identifyer from allIdentifyers list

{

int index;

string temp = input[i];

while ((index = temp.IndexOf(allIdentifyers[j].Trim())) != -1) //if we have found such substring we check charecters on the left and on the right from it

{

bool prevCharIsLetter = false;

bool nextCharIsLetter = false;

char? prevChar = null;

char? nextChar = null;

if (index - 1 >= 0)

{

prevChar = temp[index - 1];

if ((65 <= prevChar && prevChar <= 90) || (97 <= prevChar && prevChar <= 122))

prevCharIsLetter = true;

}

if (index + allIdentifyers[j].Length < temp.Length)

{

nextChar = temp[index + allIdentifyers[j].Length];

if ((65 <= nextChar && nextChar <= 90) || (97 <= nextChar && nextChar <= 122))

nextCharIsLetter = true;

}

if ((!prevCharIsLetter || prevChar == null) && (!nextCharIsLetter || nextChar == null)) //if these charecters are not letters or dont exist(beginng or end of string)

{

if (root.GetNode(i + 1) != null)

{

if (!root.GetNode(i + 1).Value.Contains(allIdentifyers[j]))//add new element to the tree if string wasn`t added before

root.GetNode(i + 1).Value.Add(allIdentifyers[j]);

}

else

{

root.AddNode(new BinaryNode(i + 1, new List<string> { allIdentifyers[j] }));//add new element to the list of identifyers found in the string

}

input[i] = input[i].Remove(index, allIdentifyers[j].Length);

temp = input[i];

}

else

{

temp = temp.Substring(index + allIdentifyers[j].Length);//trimming the current string till the end of found substring

}

}//repeating for other identifyers

}

}

return root;

}

//if specifyed arrays of elements which can precede and elements which can go after the wanted squeezed string it returns an array of substrings from which satisfy condition

private static List<string> GetSqueezedStr(List<string> input, List<string> StartDef, List<string> EndDef)

{

List<string> identifyerList = new List<string>();

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

{

if (input[i].Length > 0)

{

bool LineContainsIdentifyer = false;

for (int j = 0; j < StartDef.Count; j++)

{

int index;

string newTrimmedStr;

while ((index = input[i].IndexOf(StartDef[j])) != -1)

{

newTrimmedStr = input[i].Substring(index + StartDef[j].Length).Trim();

input[i] = newTrimmedStr;

LineContainsIdentifyer = true;

}

}

if (LineContainsIdentifyer)

{

List<string> sameLineIdentifyerList = new List<string>(input[i].Split(","));

for (int j = 0; j < sameLineIdentifyerList.Count; j++)

{

int topTrim = sameLineIdentifyerList[j].Length;

for (int k = 0; k < EndDef.Count; k++)

{

int index = sameLineIdentifyerList[j].IndexOf(EndDef[k]);

topTrim = (topTrim > index && index != -1) ? index : topTrim;

}

sameLineIdentifyerList[j] = sameLineIdentifyerList[j].Remove(topTrim).Trim();

bool foundSameIdenifyer = false;

for (int k = 0; k < identifyerList.Count; k++)

{

if (identifyerList[k] == sameLineIdentifyerList[j])

foundSameIdenifyer = true;

}

if (!foundSameIdenifyer && sameLineIdentifyerList[j] != "")

{

identifyerList.Add(sameLineIdentifyerList[j] + " ");

}

}

}

}

}

return identifyerList;

}

public static string[] ReadMultilineInp()

{

ConsoleKeyInfo key;

bool contInp;

string temp = "";

//added line numeration

int lineCounter = 1;

Console.Write(lineCounter++.ToString() + " ");

//

do

{

contInp = true;

key = Console.ReadKey();

switch (key.Key)

{

case ConsoleKey.Enter:

temp += '\n';

Console.CursorTop++;

Console.Write(lineCounter++.ToString() + " ");

break;

case ConsoleKey.E when key.Modifiers == ConsoleModifiers.Control:

Console.Write(' ');

Console.CursorLeft--;

//Console.CursorLeft--;

contInp = false;

break;

case ConsoleKey.Backspace:

if (Console.CursorLeft > 1)

{

Console.Write(' ');

Console.CursorLeft--;

if (temp.Length > 0)

temp = temp.Remove(temp.Length - 1);

}

else

{

Console.CursorLeft++;

}

break;

default:

temp += key.KeyChar.ToString();

break;

}

}

while (contInp);

return temp.Split('\n');

}

}

}

////Code for testing

//

//Testing deleting of BinaryNodes

//while (root.RightChild != null || root.LeftChild != null)

//{

// Console.Write($"String number: {root.Key}\tIdentifyers: ");

// foreach (var str in root.Value)

// {

// Console.Write(str + " ");

// }

// Console.WriteLine();

// root.RemoveNode((int)root.Key);

//}

**BinaryNode.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab6

{

internal class BinaryNode

{

#region Fields

public BinaryNode LeftChild { get; set; }

public BinaryNode RightChild { get; set; }

public int? Key { get; set; }

public List<string> Value { get; set; }

#endregion

#region Constructors

public BinaryNode(int? key=null, List<string> value=null)

{

LeftChild = null;

RightChild = null;

Key = key;

Value = value;

}

#endregion

#region Methods

public void AddNode(BinaryNode binaryNode)//tested working

{

string editable;

if (Key != null)

{

if (binaryNode.Key == Key) throw new ArgumentException("Can`t add node with same Key");

else if (binaryNode.Key > Key)

{

editable = "RightChild";

}

else

{

editable = "LeftChild";

}

if (editable == "RightChild")

{

if (RightChild != null)

RightChild.AddNode(binaryNode);

else

RightChild = binaryNode;

}

else if (editable == "LeftChild")

{

if (LeftChild != null)

LeftChild.AddNode(binaryNode);

else

LeftChild = binaryNode;

}

}

else

{

this.ChangeNodeTo(binaryNode);

}

}

public void ChangeNodeTo(BinaryNode binaryNode)

{

Key = binaryNode.Key;

Value = binaryNode.Value;

LeftChild = binaryNode.LeftChild;

RightChild = binaryNode.RightChild;

}

public BinaryNode? GetNode(int key)//tested-working

{

if (key == Key)

return this;

else if (key < Key && LeftChild != null)

return LeftChild.GetNode(key);

else if (key > Key && RightChild != null)

return RightChild.GetNode(key);

else

return null;

}

public void RemoveNode(int key)

{

if (key == Key)

{

BinaryNode replacementNode;

if (LeftChild != null)

{

replacementNode = LeftChild.GoToSubtree(LeftChild.RightChild);

this.ChangeNodeTo(replacementNode);

replacementNode.ChangeNodeTo(new BinaryNode());

}

else if (RightChild != null)

{

replacementNode = RightChild.GoToSubtree(RightChild.LeftChild);

this.ChangeNodeTo(replacementNode);

replacementNode.ChangeNodeTo(new BinaryNode());

}

else

{

this.ChangeNodeTo(new BinaryNode());

}

}

else

{

if (key < Key && LeftChild != null) LeftChild.RemoveNode(key);

else if (key > Key && RightChild != null) RightChild.RemoveNode(key);

else throw new ArgumentException("Can`t remove the node--wasn`t found");

}

}

public BinaryNode GoToSubtree(BinaryNode subtree, int steps = -1)//tested working

{

BinaryNode nextNode;

if (subtree == RightChild || subtree == LeftChild)

nextNode = subtree;

else

throw new ArgumentException("wrong parameter passed as a type of subtree");

if (steps == 0)

return this;

else if (steps == -1)

{

if (nextNode != null)

return nextNode.GoToSubtree(subtree, steps);

else

return this;

}

else if (steps > 0 && nextNode != null)

return nextNode.GoToSubtree(subtree, steps - 1);

else

throw new ArgumentException("incrorrect steps parameter");

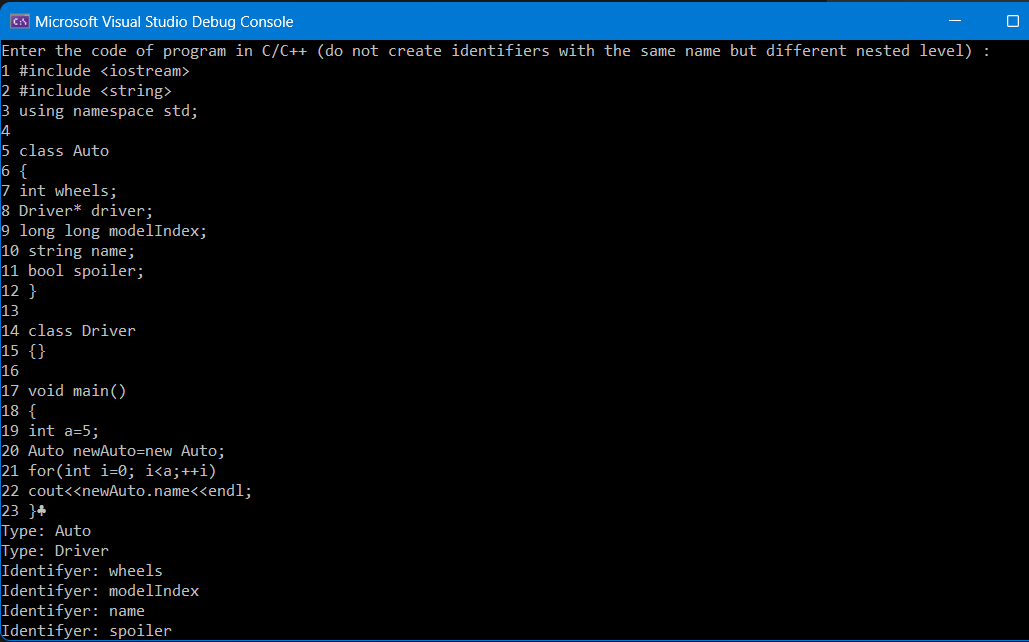
}

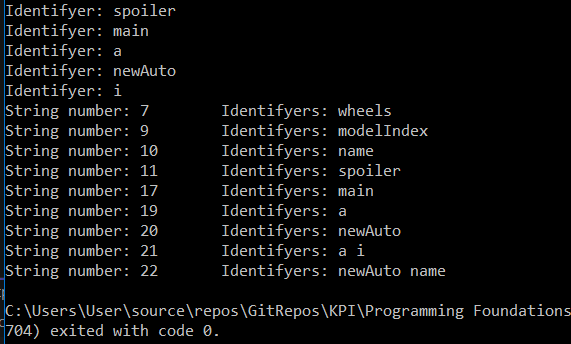
#endregion

}

}

**Виконання коду на C#:**





**Висновок:**

Отже в ході даної лабораторної роботи ми дослідили рекурсивні алгоритми обходження бінарних дерев в компільованій мові C# а також навчилися працювати з консоллю і розробили алгоритми обробки користувацького вводу. Також поглибили свої знання про роботу з текстовими файлами. Дослідили функціонал бібліотек та ближче познайомилися з концепцією ООП.