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. Методології програмування»

«Дерева»

Варіант 21

Виконав студент ІП-13 Макарчук Лідія Олександрівна
(шифр, прізвище, ім'я, по батькові)

Перевірив Вєчерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Київ 2022

Лабораторна робота №6

Дерева

Мета – вивчити особливості організації і оброки дерев.

Варіант 21

Завдання:

21. Побудувати дерево, елементами якого є символи. Знайти довжину шляху (число гілок) від кореня до значення символа, введеного з клавіатури. Різновид дерева вибрати самостійно.

Виконання завдання мовою C++

Код:

//laba6cpp.cpp (main)

```
#include <iostream>
#include <stdlib.h>
#include <vector>
#include "Functions.h"
#include "Tree.h"

using namespace std;

int main()
{
	srand(time(0));
	int n;
	init_arr_size(n);
	vector <char> arr = enter_char_arr(n);
	Tree tree;
	for (int i = 0; i < n; i++)
		tree.add_node(arr[i]);

	cout << "\nCreated tree:\n";
	tree.print_tree();
	char ch = init_char();
	int level = tree.search_char_level(ch);
	if (level != -1)
		cout << "level of char" << ch << " = " << level << "\n";
	else
		cout << "There is no char " << ch << "\n";
}
```

//Tree.h

```
#pragma once
#include <vector>

using std::vector;
class Node;

class Tree
{
	Node* root;
```

```
        void add_node_recursion(Node* &parent, char newChar);
        void print_node_pre_order(Node* &parent, int level);
        int search_recursion(Node*& parent, char ch, int level);
public:
        Tree();
        ~Tree();
        void add_node(char ch);
        void print_tree();
        int search_char_level(char ch);
};

class Node
{
        char data;
        Node* left;
        Node* right;
public:
        Node(char ch);
        ~Node();
        friend class Tree;
};
```

//Tree.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include "Tree.h"

using namespace std;

Tree::Tree()
{
        root = NULL;
}
Tree::~Tree()
{
        delete root;
}

void Tree::add_node(char ch)
{
        add_node_recursion(root, ch);
}

void Tree::add_node_recursion(Node* &parent, char newChar)
{
        if (parent == NULL)
                parent = new Node(newChar);
        else if (newChar < parent->data)
        {
                if (parent->left == NULL)
                        parent->left = new Node(newChar);
                else
                        add_node_recursion(parent->left, newChar);
        }
        else if (newChar > parent->data)
        {
                if (parent->right == NULL)
                        parent->right = new Node(newChar);
                else
                        add_node_recursion(parent->right, newChar);
        }
}

void Tree::print_tree()
```

```
{
	print_node_pre_order(root, 0);
}
void Tree::print_node_pre_order(Node* &parent, int level)
{
	char space = ' ';
	char under = '_';
	for (int i = 0; i < level; i++)
		cout << string(3, space) << "|";
	cout << string(2, under);
	if (parent != NULL)
	{
		cout << parent->data << "\n";
		print_node_pre_order(parent->right, level + 1);
		print_node_pre_order(parent->left, level + 1);
	}
	else
		cout << "\n";
}

int Tree::search_char_level(char ch)
{
	int n = search_recursion(root, ch, 0);
	return n;
}
int Tree::search_recursion(Node*& parent, char ch, int level)
{
	if (parent == NULL)
		return -1;
	else if (parent->data == ch)
		return level;
	else if (ch < parent->data)
		return search_recursion(parent->left, ch, level + 1);
	else
		return search_recursion(parent->right, ch, level + 1);
}

Node::Node(char ch)
{
	data = ch;
	left = NULL;
	right = NULL;
}
Node::~Node()
{
	delete left;
	delete right;
}
```

//Functions.h

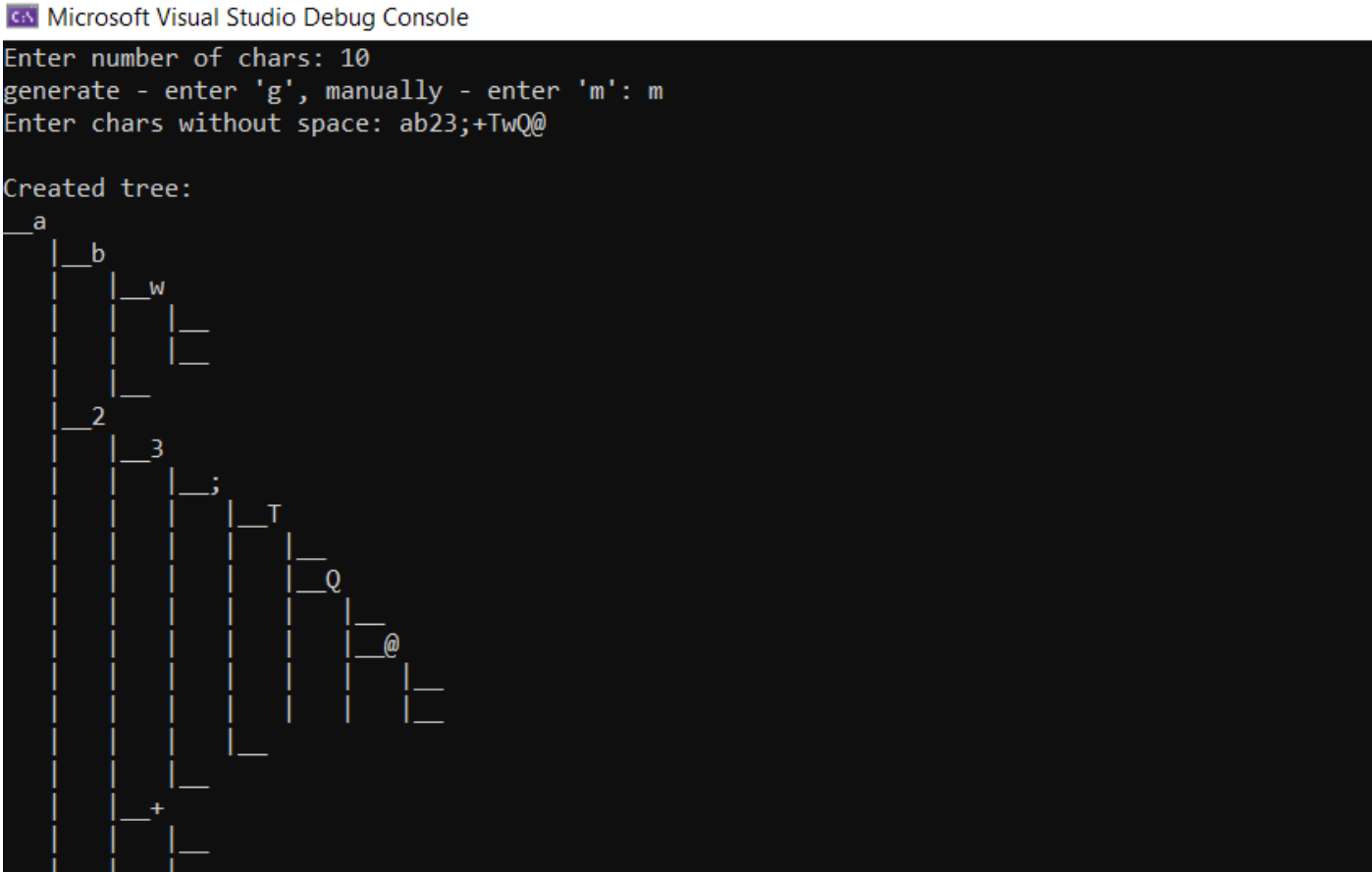
```
#pragma once
#include <iostream>
#include <vector>

using std::vector;

void init_arr_size(int& n);
vector <char> generate_chars(int n);
vector <char> init_chars(int n);
void print_arr(vector <char> arr);
vector <char> enter_char_arr(int n);
char init_char();
```

//Functions.cpp

```cpp
#include <iostream>
#include <stdlib.h>
#include <string>
#include "Functions.h"

using namespace std;

vector <char> enter_char_arr(int n)
{
	vector <char> arr;
	cout << "generate - enter 'g', manually - enter 'm': ";
	char mode;
	cin >> mode; cin.ignore();
	while (mode != 'g' && mode != 'm')
	{
		cout << "Try again! ";
		cin >> mode; cin.ignore();
	}
	if (mode == 'm')
		arr = init_chars(n);
	else
	{
		arr = generate_chars(n);
		cout << "Created array:\n";
		print_arr(arr);
	}
	return arr;
}

void init_arr_size(int& n)
{
	cout << "Enter number of chars: ";
	cin >> n; cin.ignore();
	while (n<1)
	{
		cout << "Try again: ";
		cin >> n; cin.ignore();
	}
}
vector <char> generate_chars(int n)
{
	int rangeMin = 32;
	int rangeMax = 126;
	vector <char> arr(n);
	for (int i=0;i<n;i++)
		arr[i] = rand() % (rangeMax - rangeMin + 1) + rangeMin;
	return arr;
}

void print_arr(vector <char> arr)
{
	for (int i = 0; i < arr.size(); i++)
		cout << arr[i];
	cout << "\n";
}

vector <char> init_chars(int n)
{
	vector <char> arr(n);
	cout << "Enter chars without space: ";
	string str;
	getline(cin, str);
	for (int i = 0; i <n; i++)
		arr[i] = str[i];
	return arr;
}
char init_char()
{
```

```
        char ch;
        cout << "\nEnter char to search: ";
        string str;
        getline(cin, str);
        while (str.length() < 1)
        {
                cout << "Try again! ";
                getline(cin, str);
        }
        ch = str[0];
        return ch;
}
```

Тестування програми:



Висновок: Під час виконання лабораторної роботи я вивчила особливості організації і оброки дерев. У результаті я створила бінарне дерево пошуку, елементами якого є символи та знайшла рівень символу, який був введений з клавіатури.