Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра інформаційних систем та мереж

ЗВІТ

про виконання лабораторних робіт №5

**“ Робота з динамічними структурами**.**”**

з дисципліни "Алгоритми та структури даних ”

В-15

Виконала студентка групи СА-32

Катеринчук Тетяна Андріївна

Прийняв доцент

Щербак С.С.

*2017*

**Мета роботи:** Набуття навичок опрацювання таких динамічних структур як звязні списки і дерева.

**Завдання:**

Розробити програми, які виконують операції вказані в індивідуальному завданні.

* Програму для роботи з двонапрааленими звязними списками. Кожен елемент списку містить зсилки на наступний і попередній елемент в списку. Програма повинна забезпечувати ввід і побудову списку.
* Програму для роботи для роботи з деревами. Кожен елемент дерева містить зсилку на батьківський елемент і зсилки на елементи-нащадки (необмежена кількість). Програма повинна забезпечувати ввід і побудову дерева.
* Кожен елемент списку містить інформаційне поле(атрибут) деякого простого типу: символ, стрічка, число.
* Всі операції над динамічними структурами повинні супроводжуватись відповідним виводом на екран.

**Індивідуальне завдання:**

|  |  |  |
| --- | --- | --- |
| 15 | Вставка нових елементів на непарні позиції в списку. | Видалення елемента з дерева за вказаним значенням інформаційного атрибуту. (Нащадки видаленого елемента стають нащадками батьківського елемента видаленого) |

**Текст програми**

Main.cpp

#include <iostream>

#include "MyTree.h"

#include "MyList.h"

using namespace std;

int main()

{

setlocale(LC\_CTYPE, "ukr");

int menu = true;

while (menu) {

int nNodeTree, temp, nList;

cout << "\n1 - опрацювання списка\n2 - опрацювання дерева\n0 - вихiд\n";

cin >> menu;

if (menu == 1) {

cout << "Введiть кiлькiсть елементiв списка: ";

cin >> nList;

List \*list = NULL;

for (int i = 0; i < nList; i++) {

cout << "Елемент " << i + 1 << " = ";

cin >> temp;

AddElement(temp, list);

}

cout << "Елементи списка: \n";

OutputList(list);

cout << endl;

int addElement, nAddElement;

cout << "Введiть на яку позицію (непарну) ви хочете вставити елемент: "; cin >> nAddElement;

cout << "Введiть елемент: "; cin >> addElement;

AddOddElement(list, addElement, nAddElement, 1);

OutputList(list);

cout << endl;

}

if (menu == 2) {

cout << "Введiть кiлькiсть вузлiв: ";

cin >> nNodeTree;

Node \*tree = NULL;

for (int i = 0; i < nNodeTree; i++) {

cout << "Вузол " << i + 1 << " = ";

cin >> temp;

AddNode(temp, tree);

}

int choice = true;

while (choice != 0) {

OutputTree(tree);

cout << "1 - ввести новий вузол\n2 - Видалити вузол\n0 - вихiд в меню\n";

cin >> choice;

if (!choice)

return 0;

switch (choice) {

case 1:

cout << "Введiть новий вузол: ";

cin >> temp;

cout << endl;

AddNode(temp, tree);

break;

case 2:

cout << "Введiть вузол, який ви хочете видалити: ";

cin >> temp;

cout << endl;

DeleteNode(tree, tree, temp);

break;

case 0:

break; break;

default:

break;

}}}}

return 0;

}

MyList.h

#pragma once

#include <iostream>

using namespace std;

struct List

{

int element;

List \*back, \*next;

};

void SupstituteElement(List \*&list, int element);

void AddElement(int element, List \*&list)

{

if (list == NULL)

{

list = new List;

list->element = element;

list->back = list->next = NULL;

}

else if (list != NULL)

{

if (list->next == NULL) {

list->next = new List;

list->next->next = NULL;

list->next->back = list;

list->next->element = element;

}

else if (list->next != NULL)

AddElement(element, list->next);

}

}

void OutputList(List \*&list) {

if (list != NULL) {

cout << list->element << " ";

OutputList(list->next);

}

}

void AddOddElement(List \*&list, int element, int position, int n) {

if (list != NULL) {

if (n != position) {

AddOddElement(list->next, element, position, n + 1);

}

else if (n == position) {

SupstituteElement(list, element);

}

}

}

void SupstituteElement(List \*&list, int element) {

if (list != NULL) {

int temp = list->element;

list->element = element;

SupstituteElement(list->next, temp);

}

else {

AddElement(element, list);

}

}

MyTree.h

#pragma once

#include <iostream>

using namespace std;

struct Node

{

int top;

Node \*left, \*right;

};

void DeleteOneNode(Node \*&fatherNode, Node \*tree);

int DeleteTwoNode(Node \*&fatherNode, Node \*tree);

void OutputTree(Node \*&tree)

{

if (tree != NULL && (tree->left != NULL || tree->right != NULL))

{

cout << "\t " << tree->top << endl;

if (tree->left != NULL) {

cout << "\t" << tree->left->top;

if (tree->right != NULL)

cout << "\t" << tree->right->top;

}

else {

cout << "\t\t" << tree->right->top;

}

cout << endl << endl << endl;

OutputTree(tree->left);

OutputTree(tree->right);

}

}

void AddNode(int top, Node \*&tree)

{

if (NULL == tree)

{

tree = new Node;

tree->top = top;

tree->left = tree->right = NULL;

}

if (top<tree->top)

{

if (tree->left != NULL) AddNode(top, tree->left);

else

{

tree->left = new Node;

tree->left->left = tree->left->right = NULL;

tree->left->top = top;

}

}

if (top>tree->top)

{

if (tree->right != NULL) AddNode(top, tree->right);

else

{

tree->right = new Node;

tree->right->left = tree->right->right = NULL;

tree->right->top = top;

}

}

}

int DeleteNode(Node \*tree, Node \*&fatherNode, int deleteNode) {

if (tree != NULL) {

if (tree->top == deleteNode) {

if (tree->left == NULL&&tree->right == NULL) {

if (fatherNode->left != NULL) {

fatherNode->left = NULL;

}

else if (fatherNode->right != NULL) {

fatherNode->right = NULL;

}

}

else if (tree->left != NULL&&tree->right == NULL) {

if (fatherNode->left != NULL) {

if (fatherNode->left->top == tree->top) {

DeleteOneNode(fatherNode->left, tree->left);

}

}

else if (fatherNode->right != NULL) {

if (fatherNode->right->top == tree->top) {

DeleteOneNode(fatherNode->right, tree->left);

}

}

}

else if (tree->left == NULL&&tree->right != NULL) {

if (fatherNode->left != NULL) {

if (fatherNode->left->top == tree->top)

DeleteOneNode(fatherNode->left, tree->right);

}

if (fatherNode->right != NULL) {

if (fatherNode->right->top == tree->top) {

DeleteOneNode(fatherNode->right, tree->right);

}

}

}

else {

if (fatherNode->left != NULL) {

if (fatherNode->left->top == tree->top) {

Node\* tempTree = tree->left;

DeleteOneNode(fatherNode->left, tree->right);

DeleteTwoNode(fatherNode->left, tempTree);

free(tempTree);

}

}

if (fatherNode->right != 0) {

if (fatherNode->right->top == tree->top) {

Node\* tempTree = tree->left;

DeleteOneNode(fatherNode->right, tree->right);

DeleteTwoNode(fatherNode->right, tempTree);

free(tempTree);

}

}

}

return 0;

}

else if (deleteNode<tree->top) {

DeleteNode(tree->left, tree, deleteNode);

}

else if (deleteNode > tree->top) {

DeleteNode(tree->right, tree, deleteNode);

}

else {

cout << "Даного вузла немає у дереві!\n";

return 0;

}

}

else {

return 0;

}

}

int DeleteTwoNode(Node \*&fatherNode, Node \*tree) {

if (fatherNode == NULL) {

DeleteOneNode(fatherNode, tree);

return 0;

}

DeleteTwoNode(fatherNode->left, tree);

return 0;

}

void DeleteOneNode(Node \*&fatherNode, Node \*tree) {

if (tree != NULL) {

if (fatherNode == NULL) {

fatherNode = new Node;

fatherNode->left = new Node;

fatherNode->right = new Node;

fatherNode->left = NULL;

fatherNode->right = NULL;

}

fatherNode->top = tree->top;

fatherNode->left = tree->left;

fatherNode->right = tree->right;

DeleteOneNode(fatherNode->right, tree->right);

DeleteOneNode(fatherNode->left, tree->left);

}

}

**Результат програми**

Створення списку та вставка елемента на непарну позицію

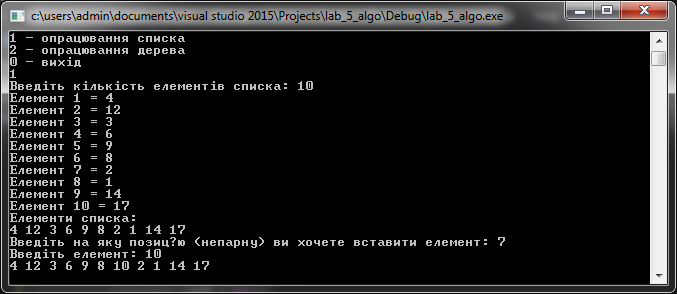


Рис.1 Вставка елемента в список на непарну позицію

Створення дерева та видалення вузла

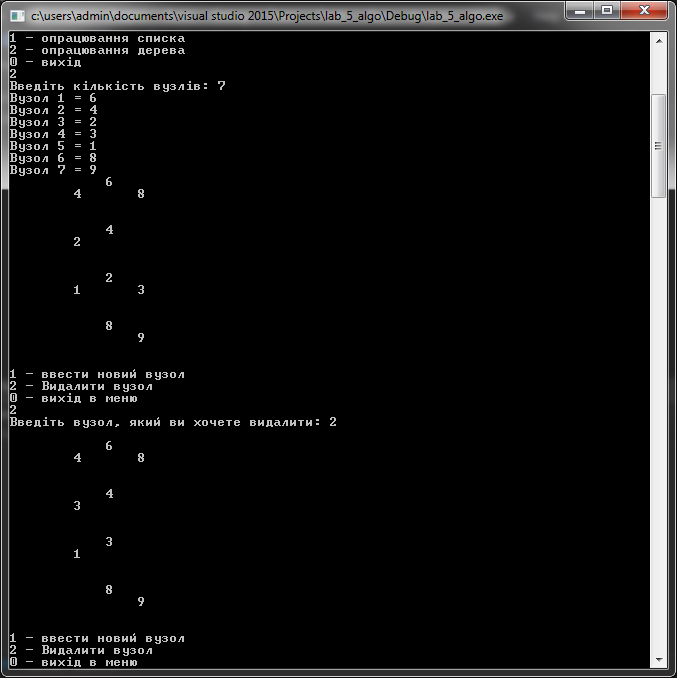


Рис. 2 Видалення вузла в дереві

**Висновок:** При виконання даної лабараторної роботи я набула навичок опрацювання таких динамічних структур як звязні списки і дерева. Створила програму яка складається з двох частин. Перша частина це створення двонаправленого списку, кожний елемент якої має посилання на попередній та наступний елемент та вставка елемента в непарну позицію в списку. Друга частина це робота з деревами: створення дерева, додавання та видалення елемента.