 МІНІСТЕРСТВО ОСВІТИ I НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Комп’ютерний практикум №17**

з дисципліни «Алгоритмізація та програмування»

на тему: «Дерева»

Варіант №17

**Виконав:**

студент гр. БС-81

Сєров О. В.

**Перевірив:**

доцент каф. БМК

к.т.н. Алхімова С.М.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис викладача)

Київ-2019

**Завдання:**

1. Ознайомитися з теоретичними відомостями створення, обробки та видалення двійкових дерев пошуку.
2. Відповідно до варіанту визначити структуру двійкового дерева пошуку та розробити функції, що необхідні для роботи з ним: функцію додавання елемента (пам’ять під черговий елемент контейнера слід виділяти динамічно), функцію друку всіх елементів контейнера на екран, функцію видалення контейнера.
3. Побудувати блок-схему алгоритму для вирішення задачі відповідно до свого варіанту.
4. Розробити програмний застосунок, в якому створюється контейнер і заповнюється вручну не менше, ніж 15-ма елементами, що в якості даних зберігають унікальні цілі числа, та реалізовано виконання індивідуального завдання, після чого всі створені контейнери видаляються (глобальні змінні не використовувати, функція main має бути призначена тільки для виклику функцій користувача):

***Розробити програму підрахунку вузлів двійкового дерева, що мають тільки по одному лівому нащадку.***

1. Скласти і захистити звіт по роботі.

**🞏 Комп’ютерний практикум без зауважень**

**🞏 Комп’ютерний практикум має зауваження:**

**🞏 несвоєчасний захист**

**🞏 присутні зауваження до блок-схеми:**

**🞏 блок-схема не відповідає коду**

**🞏 в блок-схемі присутній код**

**🞏 виконані не за стандартом:**

**🞏 блок умови 🞏 визначений процес (функція)**

**🞏 оператор вибору 🞏 перехід**

**🞏 цикл 🞏 розміри блоків**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 присутні зауваження до коду:**

**🞏 задача завдання вирішена хибно**

**🞏 код програми не компілюється**

**🞏 використано глобальні змінні**

**🞏 типи даних визначені хибно**

**🞏 недостатня декомпозиція на функції користувача**

**🞏 функція main містить лише виклик іншої функції**

**🞏 статичні змінні при роботі з масивами**

**🞏 оформлення коду**

**🞏 присутні зайві символи «{» та «}»**

**🞏 інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**🞏 результати виконання програми на рисунках не відповідають коду**

**🞏 невірні відповіді на запитання:**

**🞏 №1 🞏 №2 🞏 №3 🞏 №4 🞏 №5**

**🞏 №6 🞏 №7 🞏 №8 🞏 №9 🞏 №10**

**🞏 незнання теоретичного матеріалу**

**🞏 маються інші зауваження:**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Блок-схема:**









**Код:**

#include <iostream>

using namespace std;

struct Node

{

int data;

struct Node\* left, \* right;

};

Node\* new\_node(int item)

{

Node\* temp = new Node;

temp->data = item;

temp->left = NULL;

temp->right = NULL;

return temp;

}

void inorder\_traversal(Node\* root, unsigned short &counter)

{

if (root != NULL)

{

inorder\_traversal(root->left, counter);

if (root->left != NULL && root->right == NULL)

counter++;

cout << root->data << ' ';

inorder\_traversal(root->right, counter);

}

}

Node\* insert\_node(Node\* node, int data)

{

if (node == NULL)

return new\_node(data);

if (data < node->data)

node->left = insert\_node(node->left, data);

else

node->right = insert\_node(node->right, data);

return node;

}

Node\* delete\_node(Node\* root, int num\_to\_delete)

{

if (root == NULL)

return root;

if (root->data > num\_to\_delete)

{

root->left = delete\_node(root->left, num\_to\_delete);

return root;

}

else

if (root->data < num\_to\_delete)

{

root->right = delete\_node(root->right, num\_to\_delete);

return root;

}

if (root->left == NULL)

{

Node\* temp = root->right;

delete root;

return temp;

}

else

if (root->right == NULL)

{

Node\* temp = root->left;

delete root;

return temp;

}

else

{

Node\* two\_child\_parent = root->right;

Node\* replacement = root->right;

while (replacement->left != NULL)

{

two\_child\_parent = replacement;

replacement = replacement->left;

}

two\_child\_parent->left = replacement->right;

root->data = replacement->data;

delete replacement;

return root;

}

}

void binary\_tree(void)

{

cout << "This program is working with binary tree and count nodes with only left child\n\n";

Node\* root = NULL;

int\* array = new int[15];

int rnd = 0;

for (int i = 0; i < 15; i++)

{

rnd = rand() % 50;

root = insert\_node(root, rnd);

array[i] = rnd;

}

unsigned short only\_left\_subtree = 0;

cout << "\nInorder traversal of the tree:\n";

inorder\_traversal(root, only\_left\_subtree);

cout << "\nNumber of the nodes with only left child: " << only\_left\_subtree << '\n';

cout << "\nDelete node " << array[14];

root = delete\_node(root, array[14]);

only\_left\_subtree = 0;

cout << "\nInorder traversal of the new tree:\n";

inorder\_traversal(root, only\_left\_subtree);

cout << "\nNumber of the nodes with only left child: " << only\_left\_subtree << '\n';

for (int i = 13; i >= 0; i--)

root = delete\_node(root, array[i]);

cout << "\nAfter deleting, the tree traversal:\n";

inorder\_traversal(root, only\_left\_subtree);

cout << "Clear!\n\n";

delete[] array;

}

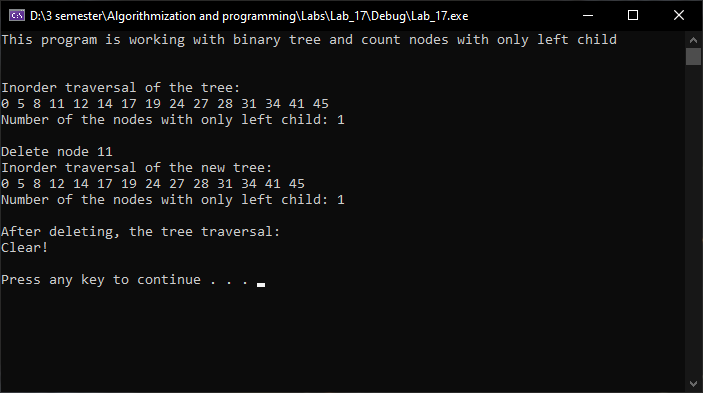
void main()

{

binary\_tree();

system("pause");

}

**Результати: **

**Контрольні питання:**

1. ***Дайте визначення дереву.***

Дерево — в інформатиці та програмуванні одна з найпоширеніших структур даних. Формально дерево визначається як скінченна множина Т з одним або більше вузлами.

Дерево представляє собою структуру даних, яка підтримує багато операцій з динамічними множинами, включаючи пошук елемента, визначення мінімального і максимального значення, попереднього і наступного елемента, операції вставка та видалення. Таким чином, дерево пошуку може використовуватися і як словник, і як черга з пріоритетами.

1. ***Що таке бінарне дерево, бінарне дерево пошуку?***

*Бінарне дерево* – це структура даних, кожен елемент якої окрім самих даних містить покажчики на два наступних елементи структури. Один з цих наступних елементів умовно називається лівим, а інший правим.

Кожен елемент дерева називається вузлом або листом дерева. Перший вузол дерева (з якого дерево власне починається) називається коренем.

Фрагмент дерева разом з вузлом, від якого він починається, називається піддеревом або віткою.

Множина всіх вузлів, рівновіддалених від кореня, називається рівнем. Вузол, з якого не починається жодна вітка, називається кінцевим або термінальним вузлом.

Оскільки дерево бінарне, кожен вузол може породжувати два вузли наступного рівня. Породжені вузли є дочірніми по відношенню до вузла, що їх породив. Породжуючий вузол є батьківським по відношенню до своїх дочірніх вузлів. Батьківський вузол разом із своїми дочірніми складає ланку.

Сумарна кількість рівнів дерева називається висотою дерева.

Двійкове дерево пошуку – теж бінарне дерево, але будується за певними правилами:

* у кожного вузла не більше двох дітей;
* будь-яке значення менше значення вузла стає лівою дитиною або дитиною лівої дитини;
* будь-яке значення більше або рівне значенню вузла стає правою дитиною або дитиною правої дитини.

1. ***Який тип інформації зручно представляти за допомогою бінарних дерев?***

За допомогою бінарних дерев зручно представляти ієрархічні структури, великий об’єм даних, до яких потрібен швидкий доступ.

1. ***Що таке обхід дерева, коли його застосовують?***

Обхід дерева – це спосіб дослідження його вузлів, при якому кожен з них проходиться точно один раз.

1. ***Навести схему довільного бінарного дерева пошуку та вказати порядок прямого, симетричного та зворотного обходу.***



Прямий: 6 → 2 → 1 → 4 → 3 → 5 → 7

Симетричний: 1 → 2 → 3 → 4 → 5 → 6 → 7

Зворотній: 1 → 3 → 5 → 4 → 2 → 7 → 6

1. ***Які алгоритми пошуку елементів зручно застосовувати підчас роботи із деревами?***

Пошук в ширину (BFS) йде з початкової вершини, відвідує спочатку всі вершини, що знаходяться на відстані одного ребра від початкової, потім відвідує всі вершини на відстані двох ребер від початкової і так далі. Алгоритм пошуку в ширину є за своєю природою нерекурсивним (ітеративним). Для його реалізації застосовується структура даних черги (FIFO).

Пошук в глибину (DFS) йде з початкової вершини, відвідуючи ще не відвідані вершини без огляду на віддаленість від початкової вершини. Алгоритм пошуку в глибину за своєю природою є рекурсивним. Для емуляції рекурсії в ітеративному варіанті алгоритму застосовується структура даних стек.

1. ***Чи можна побудувати повне бінарне дерево із елементів 5,1,2,8,6,10,3,9,4,7?***

Ні, є незаповнені рівні крім останнього та вузли з одним нащадком.

1. ***Що таке досконале дерево, чим воно відрізняється від повного?***

Збалансоване дерево – це бінарне дерево пошуку з логарифмічною висотою, операції пошуку в ньому, вставки та видалення виконуються за логарифмічний час.

У досконалому дереві всі листки лежать на однаковій глибині, а у повному – не обов’язково.

1. ***Поясніть принцип роботи рекурсивних функцій для відображення елементів бінарного дерева.***

*Алгоритм*:

Якщо дерево порожнє – зупинитися.

Інакше:

Рекурсивно обійти ліве піддерево Т.

Застосувати функцію f до кореневого вузла.

Рекурсивно обійти праве піддерево Т.

1. ***Поясніть принцип видалення елементів бінарного деревау випадку, коли цей елемент не має нащадків, має одного нащадка, має двох нащадків.***

*Алгоритм*:

* Якщо дерево T порожнє – зупинитися;
* Інакше порівняти K з ключем X кореневого вузла n.
  + Якщо K > X, рекурсивно видалити K з правого піддерева Т;
  + Якщо K < X, рекурсивно видалити K з лівого піддерева Т;
  + Якщо K = X, то необхідно розглянути три випадки.
    - Якщо обох нащадків немає, то видаляємо поточний вузол і обнуляємо посилання на нього у батьківського вузла;
    - Якщо одного з нащадків немає, то значення полів дитини m ставимо замість відповідних значень кореневого вузла, затираючи його старі значення, і звільняємо пам'ять, займану вузлом m;
    - Якщо обидва нащадка присутні, то
      * Якщо лівий вузол m правого піддерева відсутній (n → right → left)
        + Копіюємо з правого вузла у вузол що видаляється поля K, V і посилання на правий вузол правого нащадка.
      * Інакше
        + Візьмемо найлівіший вузол m, правого піддерева n → right;
        + Скопіюємо дані (крім посилань на дочірні елементи) з m в n;
        + Рекурсивно видалимо вузол m.