БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра программного обеспечения информационных технологий

Факультет ФНиДО

Специальность ПОИТ

Индивидуальная практическая работа № 2.1

«Бинарные деревья»

по дисциплине «Структуры и алгоритмы обработки данных»

Выполнила: Карпеко Н. Г.

Договор № 941 от 20.02.2019 г.

Минск 2020

**Цель работы**:

1. Научиться строить и обходить различными способами бинарные деревья, а также выполнять на них операции с данными.

2. Научиться прошивать бинарные деревья и обходить их заданным способом.

3. Научиться строить сбалансированные АВЛ-деревья и выполнять на них заданные операции с данными.

**Задание.** Ввести 10-15 целых чисел и построить из них с помощью указателей бинарное дерево поиска. Обойти его прямым, симметричным и обратным способами. Реализовать **процедуры** поиска, вставки и удаления элементов бинарного дерева поиска.

**Листинг программы:**

**program** tree;

**uses** crt;

**type**

PNode = ^TNode;

TNode = **Record**

Key: Integer;

LTag, RTag: Boolean;

Left, Right: PNode;

**End**;

//----------- Функция поиска элемента -----------------------

**function** Find(**const** Key: Integer; Node: PNode; **var** Rez: PNode): Boolean;

**begin**

Result := False;

Rez := nil;

**while** (**not** Result) **and** (Node <> nil) **do**

**begin**

Rez := Node;

**if** Node^.Key = Key **then**

Result := True

**else**

**if** Key < Node^.Key **then**

Node := Node^.Left

**else**

Node := Node^.Right;

**end**;

**end**;

//-------- Процедура добавления элемента -----------------------

**procedure** Add(**const** Key: Integer; **var** Node: PNode);

**var**

NodeTO, NewNode: PNode;

**begin**

**if not** Find(Key, Node, NodeTO) **then**

**begin**

**New**(NewNode);

NewNode^.Key := Key;

NewNode^.LTag := True;

NewNode^.RTag := True;

NewNode^.Left := nil;

NewNode^.Right := nil;

**if** Node = nil **then**

Node := NewNode

**else**

**if** Key < NodeTO^.Key **then**

NodeTO^.Left := NewNode

**else**

NodeTO^.Right := NewNode

**end**;

**end**;

//--- Процедура удаления элемента ----------------------

**procedure** Delete(**var** Node: PNode; Key: Integer);

**var**

DelNode: PNode;

{При первом вызове в качестве фактического параметра передается адрес

левой после удаляемой вершины}

**procedure** Ud(**var** R: PNode);

**begin**

**if** R^.Right = nil **then**

**begin**

DelNode^.Key := R^.Key;

DelNode := R;

R := DelNode^.Left;

**end**

**else**

Ud(R^.Right);

**end**; //Ud()---

**begin** //Delete() -----

**if** Node = nil **then** //Первый случай удаления

Writeln(' Звено с заданным ключом в дереве отсутствует.')

**else**

**if** Key < Node^.Key **then**

Delete(Node^.Left, Key)

**else**

**if** Key > Node^.Key **then**

Delete(Node^.Right, Key)

**else**

**begin**

DelNode := Node;

**if** DelNode^.Right = nil **then** //второй случай удаления

Node := DelNode^.Left

**else**

**if** DelNode^.Left = nil **then**

Node := DelNode^.Right

**else** //Третий случай удаления

Ud(DelNode^.Left)//переход в левое для удаляемой вершины поддерева}

**end**

**end**;

//----- Процедура вывода дерева на экран -----------

**procedure** Print\_Tree(**var** Node: PNode; level: integer);

**var**

i: integer;

**begin**

**if** Node <> nil **then**

**begin**

Print\_Tree(Node^.Right, level + 1);

**for** i := 0 **to** level **do**

write(' ');

//writeln(Node^.Key, Node^.LTag, Node^.RTag);

writeln(Node^.Key);

Print\_Tree(Node^.Left, level + 1);

**end**;

**end**;

//--- Процедура прямого обхода дерева -------

**procedure** prym\_print(**var** Node: PNode);

**begin**

**if** Node <> nil **then**

**begin**

write('(', Node^.Key, ') ');

prym\_print(Node^.Left);

write(Node^.Key, ' ');

prym\_print(Node^.Right);

write(Node^.Key, ' ');

**end**

**else**

write('0 ');

**end**;

//--- Процедура симметричного обхода дерева -------

**procedure** sim\_print(**var** Node: PNode);

**begin**

**if** Node <> nil **then**

**begin**

write(Node^.Key, ' ');

sim\_print(Node^.Left);

write('(', Node^.Key, ') ');

sim\_print(Node^.Right);

write(Node^.Key, ' ');

**end**

**else**

write('0 ');

**end**;

//--- Процедура обратного обхода дерева -------

**procedure** obr\_print(**var** Node: PNode);

**begin**

**if** Node <> nil **then**

**begin**

write(Node^.Key, ' ');

obr\_print(Node^.Left);

write(Node^.Key, ' ');

obr\_print(Node^.Right);

write('(', Node^.Key, ') ');

**end**

**else**

write('0 ');

**end**;

**VAR**

Node: PNode;

s: string;

delnode, addnode: integer;

**BEGIN**

**repeat**

writeln(' Выберите желаемое действие: ');

writeln(' 1 (добавить элемент), 2 (удалить элемент), 3 (выход)');

// writeln;

readln(s);

**begin**

Node := nil;

Add(8, Node);

Add(6, Node);

Add(4, Node);

Add(2, Node);

Add(7, Node);

Add(5, Node);

Add(13, Node);

Add(9, Node);

Add(11, Node);

Add(16, Node);

**if** s = '1' **then**

**begin**

write(' Введите значение добавляемой вершины: ');

readln(addnode);

Add(addnode, Node);

**end**;

**if** s = '2' **then**

**begin**

write(' Введите значение удаляемой вершины: ');

readln(delnode);

Delete(Node, delnode);

writeln(' Элемент ', delnode, ' удален.');writeln;

**end**;

writeln(' Бинарное дерево поиска:');

Print\_Tree(Node, 0); writeln;

writeln(' Симметричный обход:');

sim\_print(Node); writeln;

writeln(' Прямой обход:');

prym\_print(Node); writeln;

writeln(' Обратный обход:');

obr\_print(Node); writeln;

writeln('---------------------------------------');

**end**;

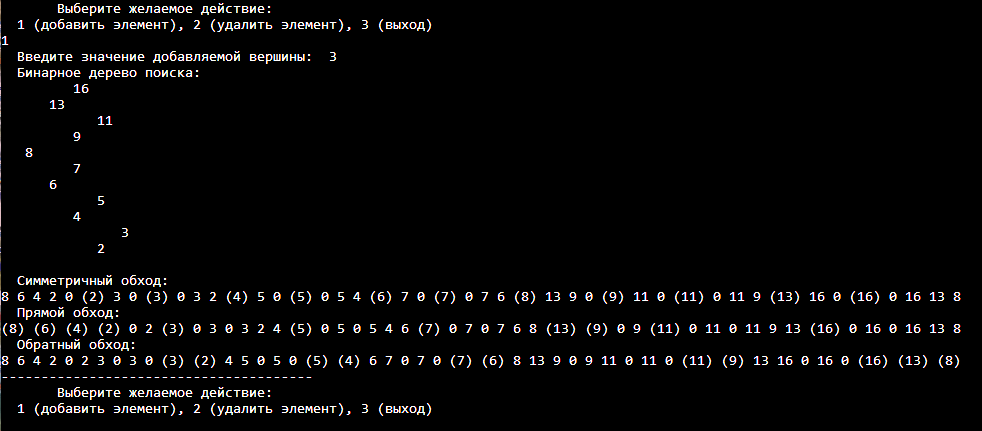
**if** s = '3' **then exit**

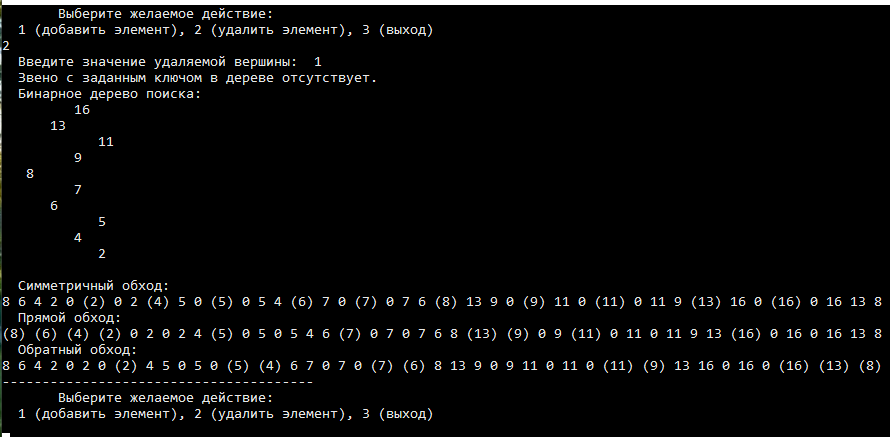
**until** s = '3';

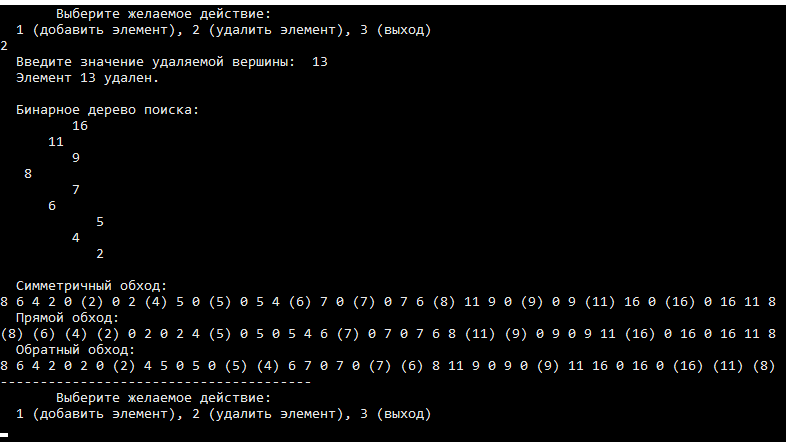
readln;

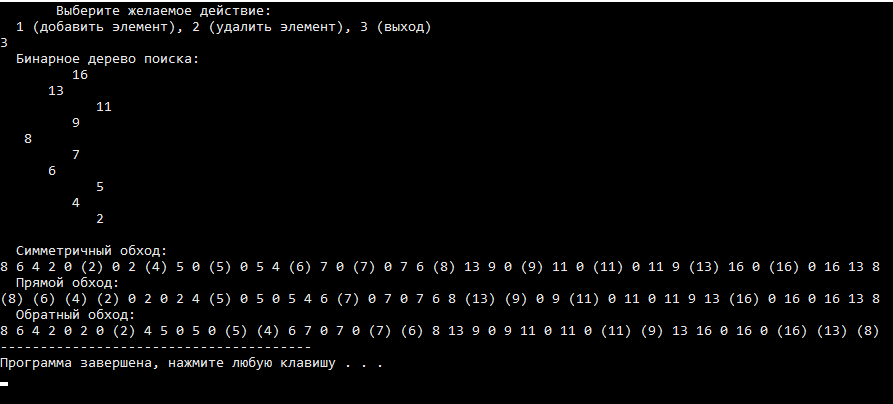
**END**.

**Результат работы программы:**









**Контрольные вопросы:**

1. Дайте определение абстрактному типу данных «дерево».

Дерево - это граф, который характеризуется следующими свойствами:

1. Существует единственный элемент (узел или вершина), на который не ссылается никакой другой элемент - и который называется корнем.

2. Начиная с корня и следуя по определенной цепочке указателей, содержащихся в элементах, можно осуществить доступ к любому элементу структуры.

3. На каждый элемент, кроме корня, имеется единственная ссылка, т.е. каждый элемент адресуется единственным указателем.

1. Дайте определение бинарному дереву и бинарному дереву поиска.

Бинарное дерево – это упорядоченное дерево, каждая вершина которого имеет не более двух поддеревьев, причем для каждого узла выполняется правило: в левом поддереве содержатся только ключи, имеющие значения, меньшие, чем значение данного узла, а в правом поддереве содержатся только ключи, имеющие значения, большие, чем значение данного узла.

Бинарное дерево поиска — это бинарное дерево, обладающее дополнительными свойствами: значение левого потомка меньше значения родителя, а значение правого потомка больше значения родителя для каждого узла дерева. То есть, данные в бинарном дереве поиска хранятся в отсортированном виде. При каждой операции вставки нового или удаления существующего узла отсортированный порядок дерева сохраняется.

1. Назовите три основных способа обхода бинарных деревьев и укажите порядок следования вершин, соответствующий каждому из обходов.

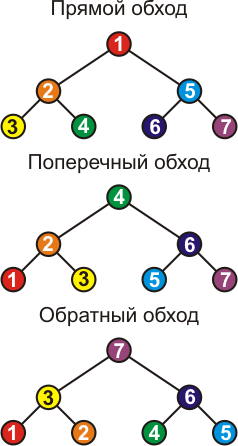
Есть три способа обхода: Прямой (preorder), Поперечный (inorder), Обратный (postorder).

1. Прямой обход: сначала обходится данная вершина, левое поддерево данной вершины, затем правое поддерево данной вершины.

2. Симметричный обход: сначала обходится левое поддерево данной вершины, затем данная вершина, затем правое поддерево данной вершины. Вершины при этом будут следовать в неубывающем (по ключам key) порядке.

3. Обратный обход: сначала обходится левое поддерево данной вершины, затем правое, затем данная вершина.

На рисунке 1 порядок обхода вершин указан номерами.



Симметричный обход

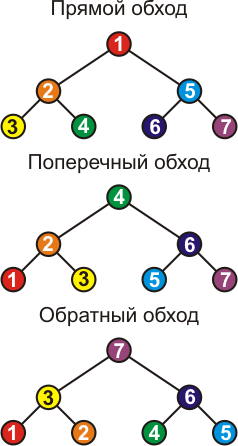


Рисунок 1.

1. Перечислите достоинства и недостатки прошивки бинарных деревьев.

Операция замены **пустых** **указателей** на **нити**, называется **прошивка**.

Достоинства прошитых деревьев: быстрый обход,нет необход-ти в стеке, м определить предшественника и преемника вершины.

Недостатки: включение новой вершины в дер занимает больше вр, т.к. надо поддерживать 2 типа связей: структурные и по нитям. Поэтому использовать прошитые деревья где изменения в дер редкие, а обходы частые.

1. Чем АВЛ-дерево отличается от идеально сбалансированного бинарного дерева?

АВЛ-дерево – дерево, у которого высота поддеревьев для каждой вершины отличается не более чем на 1.

Отличие АВЛ-дер от обычного дер поиска: при включении или удалении элементов надо поддерживать сбалансированность дерева в целом: в каждый узел дерева добавляется 1 вспомогательное поле, содержащее информацию о равновесности поддеревьев (показатель сбалансированности узла). Его значениями могут быть:

1. 0 – высоты правого и левого поддеревьев =ы;
2. 1 – высота правого поддер больше;
3. -1 – высота левого поддер больше.