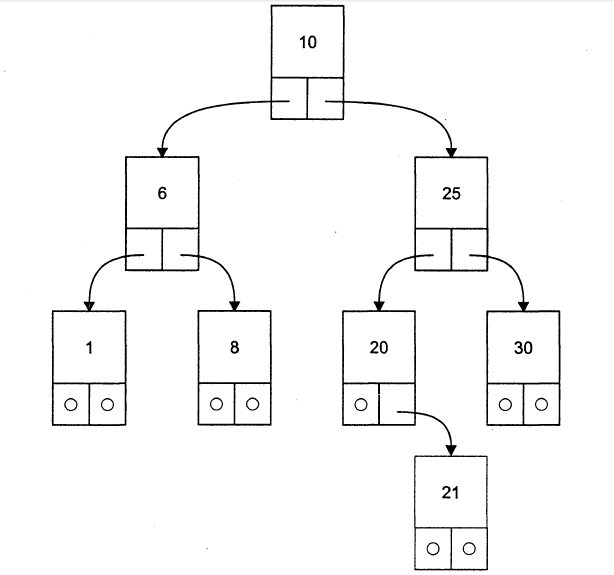
**8.3. Бинарные деревья**

Бинарное дерево — это динамическая структура данных, состоящая из узлов, каждый из которых содержит, кроме данных, не более двух ссылок на различные бинарные деревья. На каждый узел имеется ровно одна ссылка. Начальный узел называется *корнем* дерева. Узел, не имеющий поддеревьев, называется *листом.* Исходящие узлы называются предками, входящие — потомками. *Высота дерева* определяется количеством уровней, на которых располагаются его узлы.

Если дерево организовано таким образом, что для каждого узла все ключи его левого поддерева меньше ключа этого узла, а все ключи его правого поддерева — больше, оно называется *деревом поиска.* Одинаковые ключи не допускаются. В дереве поиска можно найти элемент по ключу, двигаясь от корня и переходя на левое или правое поддерево в зависимости от значения ключа в каждом узле. Такой поиск гораздо эффективнее поиска по списку, поскольку время поиска определяется высотой дерева.



Дерево является рекурсивной структурой данных, поскольку каждое поддерево также является деревом. Действия с такими структурами изящнее всего описываются с *помощью рекурсивных алгоритмов.* Например, функцию обхода всех узлов дерева в общем виде можно описать так:

function way\_\_around ( дерево ){

way\_around ( левое поддерево )

посещение корня

way\_around ( правое поддерево )

}

Можно обходить дерево и в другом порядке, например, сначала корень, потом поддеревья, но приведенная функция позволяет получить на выходе отсортированную последовательность ключей, поскольку сначала посещаются вершины с меньшими ключами, расположенные в левом поддереве. Результат обхода дерева, изображенного на рисунке:

1 6 8 10 20 21 25 30

Если в функции обхода первое обращение идет к правому поддереву, результат обхода будет другим: 30 25 21 20 10 8 6 1

Таким образом, деревья поиска можно применять для сортировки значений. При обходе дерева узлы не удаляются.

Для бинарных деревьев определены операции:

• включения узла в дерево;

• поиска по дереву;

• обхода дерева;

• удаления узла.

Для каждого рекурсивного алгоритма можно создать его нерекурсивный эквивалент.

В приведенной ниже программе реализована нерекурсивная *функция поиска по дереву с включением* и рекурсивная функция обхода дерева. Первая функция осуществляет поиск элемента с заданным ключом. Если элемент найден, она возвращает указатель на него, а если нет — включает элемент в соответствующее место дерева и возвращает указатель на него. Для включения элемента необходимо помнить пройденный по дереву путь на один шаг назад и знать, выполняется ли включение нового элемента в левое или правое поддерево его предка. Программа формирует дерево из массива целых чисел и выводит его на экран.

#include <iostream.h>

struct Node{

int d;

Node \*left;

Node \*right;

};

Node \*first(int d);

Node \*search\_Insert(Node \*root, 1nt d);

void print\_tree(Node \*p, int level);

//---------------------------------------------------

int main(){

int b[] = {10, 25, 20, 6, 21, 8, 1, 30};

Node \*root = first(b[0]);

for (int i = 1; i<8; i++) search\_Insert(root, b[i]);

print\_tree(root, 0);

return 0:

}

//---------------------------------------------------

// Формирование первого элемента дерева

Node \* first(int d){

Node \*pv = new Node;

pv->d = d;

pv->left = 0;

pv->right = 0;

return pv;

}

//----------------------------------------------------

// Поиск с включением

Node \* search\_Insert(Node \*root, int d){

Node \*pv = root, \*prev;

boo! found = false;

while (pv && !found){

prev = pv;

if (d == pv->d) found = true;

else if (d < pv->d) pv=pv->left;

else pv=pv->right;

}

if (found) return pv;

// Создание нового узла:

Node \*pnew = new Node;

pnew->d = d;

pnew->left = 0;

pnew->right = 0;

if (d < prev->d)

// Присоединение к левому поддереву предка:

prev->left == pnew;

else

// Присоединение к правому поддереву предка:

prev->right = pnew;

return pnew;

} '

// ----------------------------------------------------

// Обход дерева

void print\_tree(Node \*p, int level){

if (p){

print\_tree(p->left, level +1); // вывод левого поддерева

for (int i = 0; i<level; i++) cout « " ";

cout « p->d « endl; // вывод корня поддерева

print\_tree(p->right, level +1); // вывод правого поддерева

}

}

Текущий указатель для поиска по дереву обозначен **pv,** указатель на предка **pv** обозначен **prev,** переменная pnew используется для выделения памяти под включаемый в дерево узел. Рекурсии удалось избежать, сохранив всего одну переменную **(prev)** и повторив при включении операторы, определяющие, к какому поддереву присоединяется новый узел.

Результат работы программы для дерева, изображенного на рисунке:

1

6

8

10

20

21

25

30

Рассмотрим подробнее *функцию обхода дерева.* Вторым параметром в нее передается целая переменная, определяющая, на каком уровне находится узел. Корень находится на уровне 0. Дерево печатается по горизонтали так, что корень находится слева. Перед значением узла для имитации структуры дерева выводится количество пробелов, пропорциональное уровню узла. Если закомментировать цикл печати пробелов, отсортированный по возрастанию массив будет выведен в столбик. Заметьте, что функция обхода дерева длиной всего в несколько строк может напечатать дерево любого размера.

*Удаление узла из дерева* представляет собой не такую простую задачу, поскольку удаляемый узел может быть корневым, содержать две, одну или ни одной ссылки на поддеревья. Для узлов, содержащих меньше двух ссылок, удаление тривиально. Чтобы сохранить упорядоченность дерева при удалении узла с двумя ссылками, его заменяют на узел с самым близким к нему ключом. Это может быть самый левый узел его правого поддерева или самый правый узел левого поддерева.