Лекция 7.
Бинарные деревья

Рекурсивная функция просмотра списка

```
void prosmotr_2(struct element *tek)
 {
  if( tek != NULL )
      {
          printf("%d", tek->d);
          prosmotr_2(tek->link);
       }
 }
```

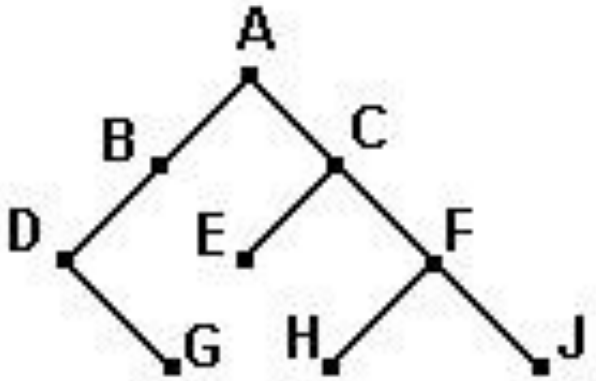
Рекурсивная функция построения списка

```
1.  void  vvod_2(struct elemeny **t)
2.  {
3.      int i;
4.      scanf("%d", &i);
5.      if (i != 0)
6.      {
7.         *t=(struct element *)malloc(sizeof(struct element));
8.          (*t)->d = i;
9.         vvod_2(&((*t)->link));
10.     }
11.     else
12.        *t=NULL;
13.   }
```

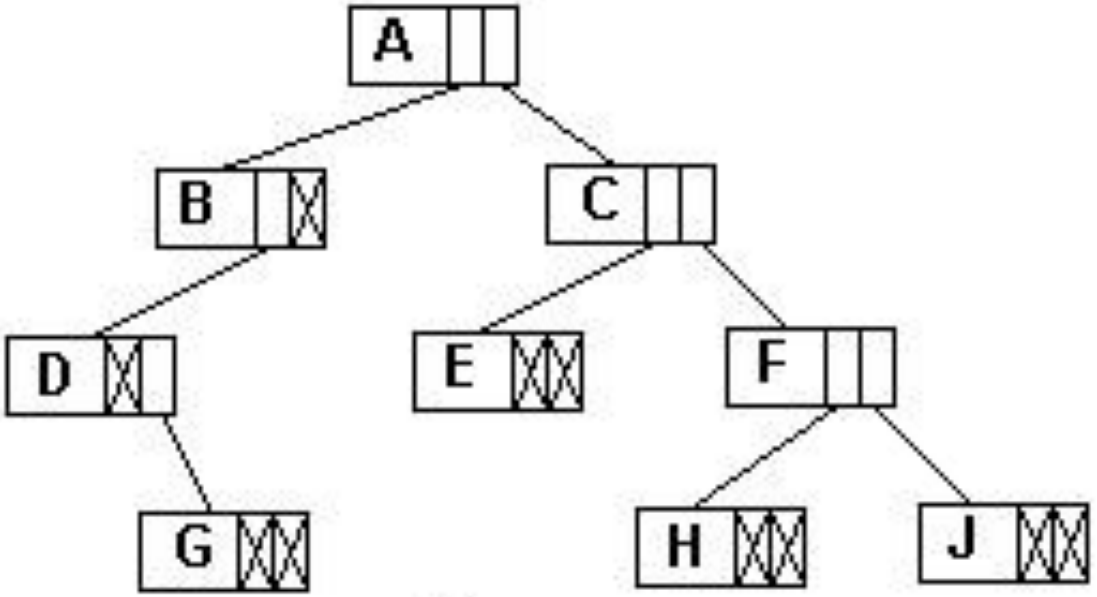
Функция main()

```
1.   int main()
2.  {
3.        struct element *nach;
4.        vvod_2(&nach);
5.        prosmotr_2(nach);
6.     . . .
7.
8.   }
```

Бинарное дерево и структура в динамической памяти



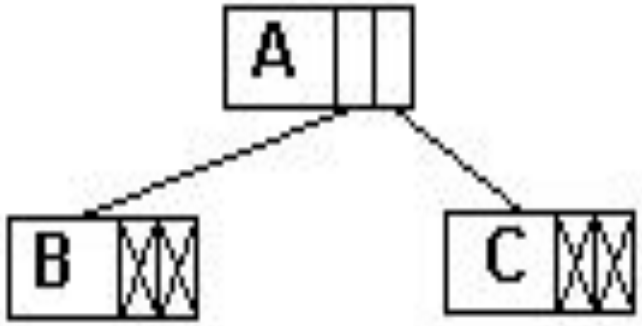
а



б

Способы обхода деревьев

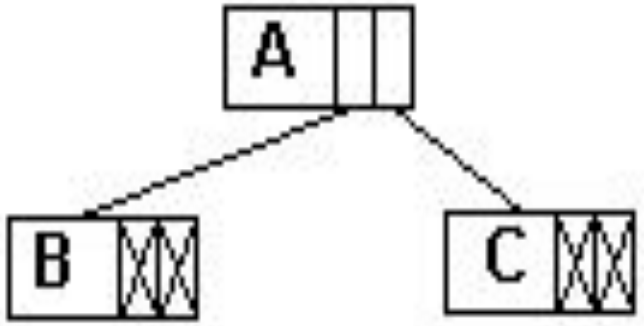
1. Сверху (прямой порядок прохождения дерева);
2. Слева направо (обратный порядок);
3. Снизу (концевой порядок прохождения).



Алгоритм 1. Прямой порядок.

1. Попасть в корень.
2. Пройти левое поддерево.
3. Пройти правое поддерево

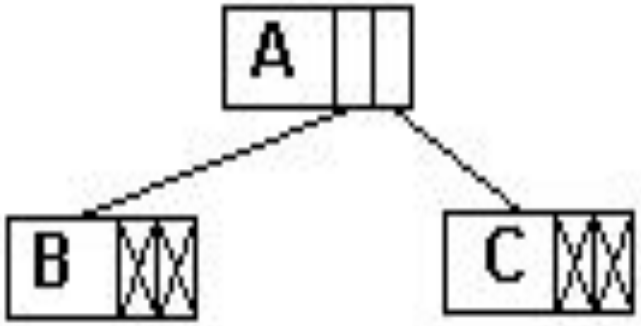
A B C



Алгоритм 2. Обратный порядок.

1. Пройти левое поддерево.
2. Попасть в корень.
3. Пройти правое поддерево.

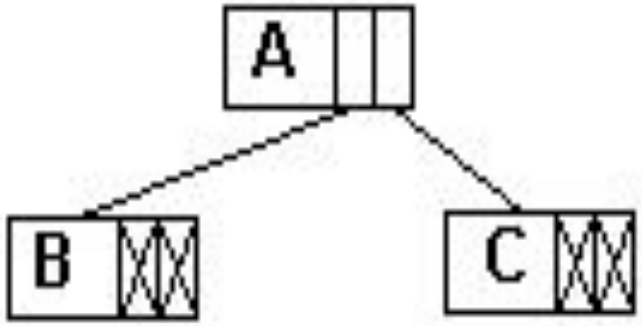
B A C



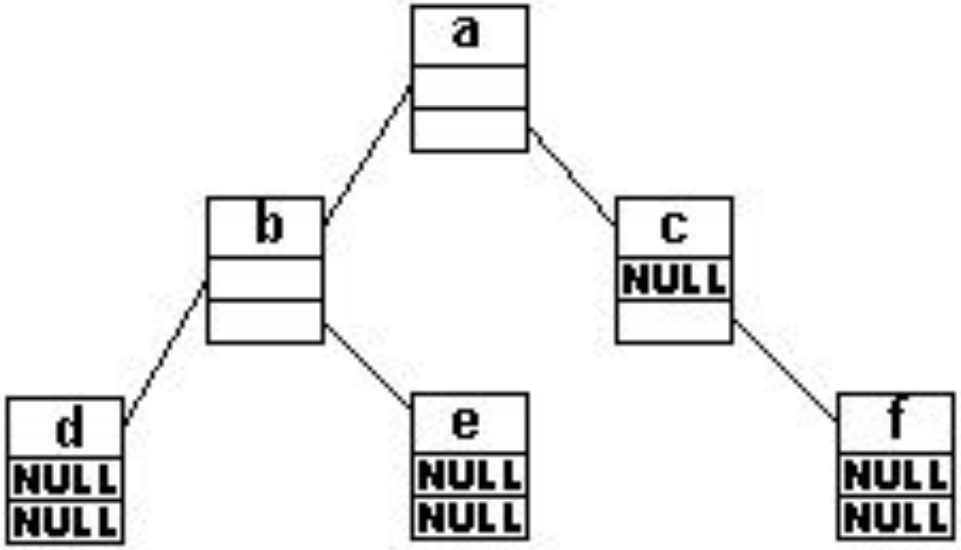
Алгоритм 3. Концевой порядок.

1. Пройти левое поддерево.
2. Пройти правое поддерево.
3. Попасть в корень.

B C A

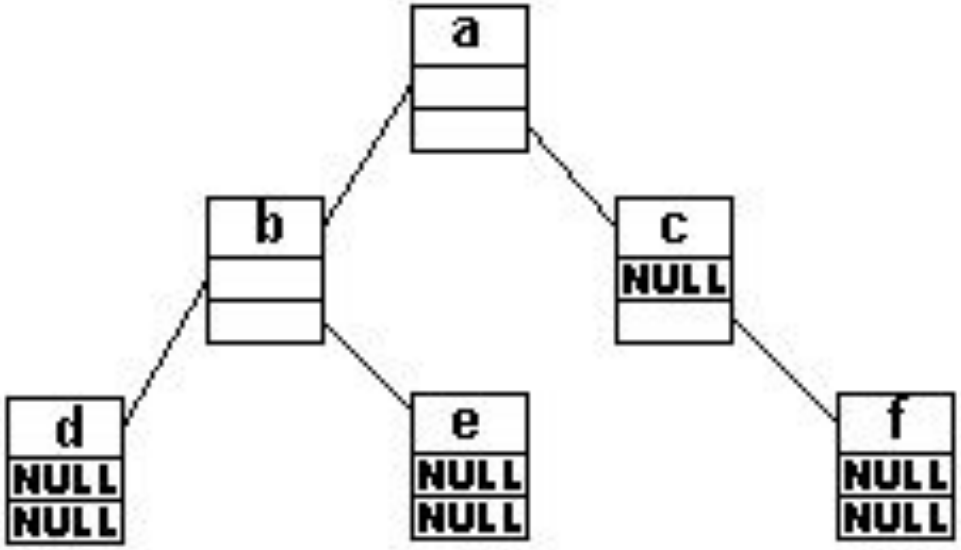


Обход дерева в прямом порядке



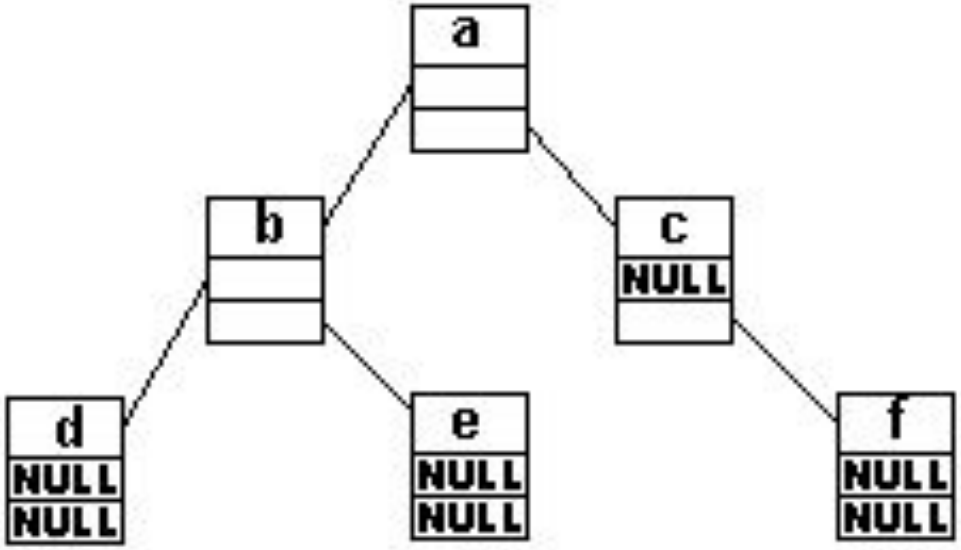
a b d e c f

Обход дерева в обратном порядке



d b e a c f

Обход дерева в концевом порядке



d e b f c a

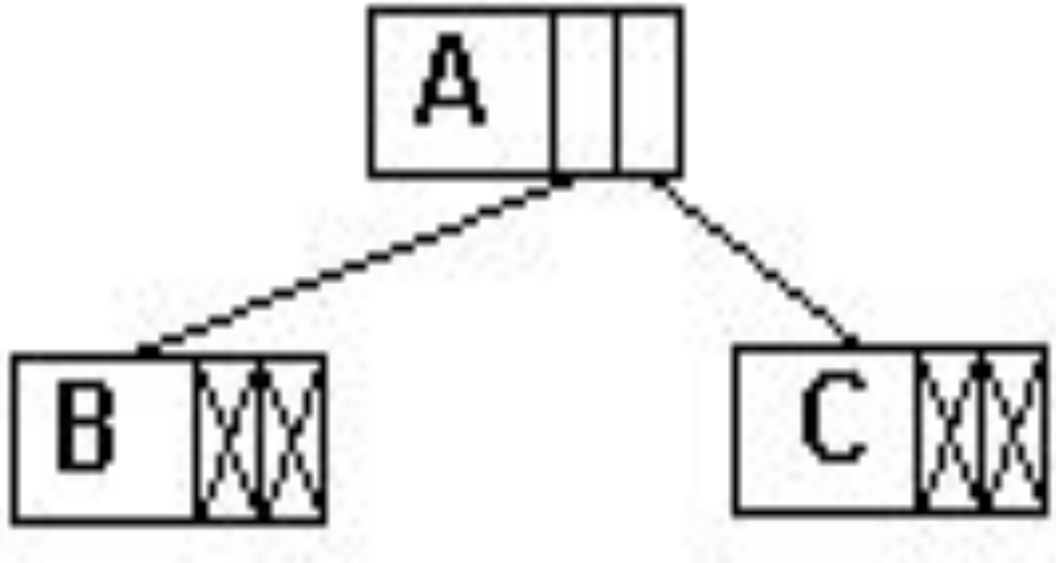
Описание типа

```
struct node
        {
                char info;
                struct node *llink;
                struct node *rlink;
        };
```

Функция main()

```
1.   int main()
2.  {
3.        struct node *root;
4.        vtree(&root);
5.        btree(root);
6.        btree2();
7.        return 0;
8.   }
```

Как вводить дерево



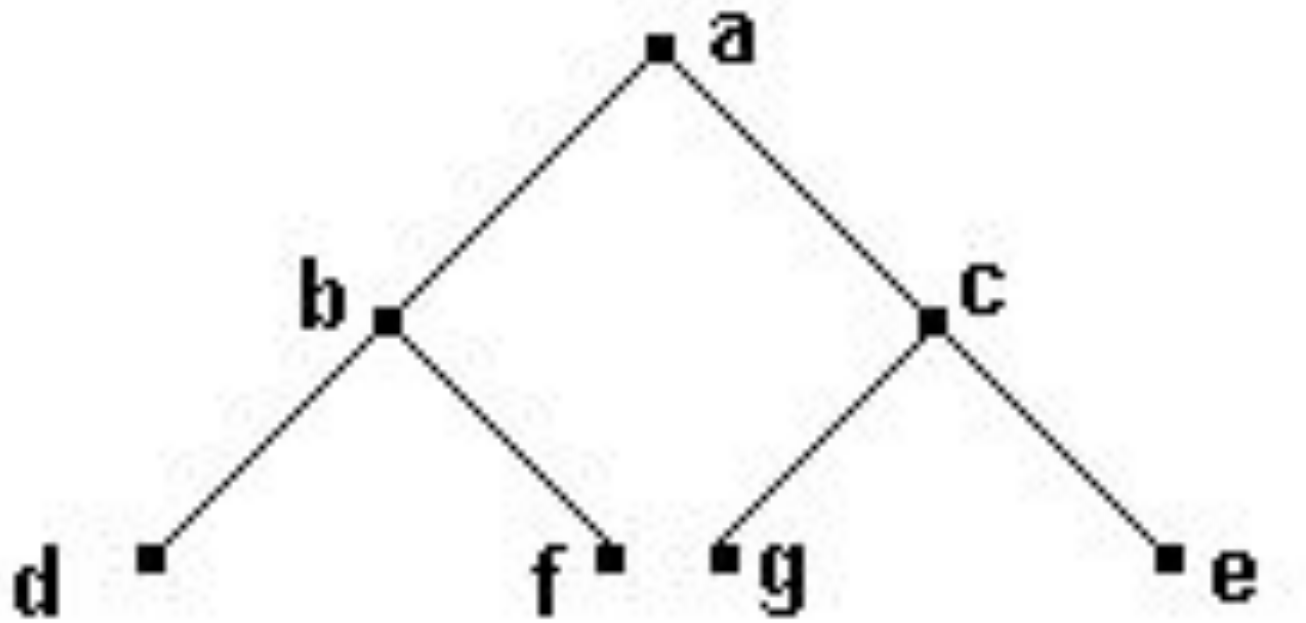
A B . . C . .

Рекурсивная функция построения бинарного дерев

- Параметр функции – указатель на указатель на корень дерева
- Вершины дерева – символьные данные

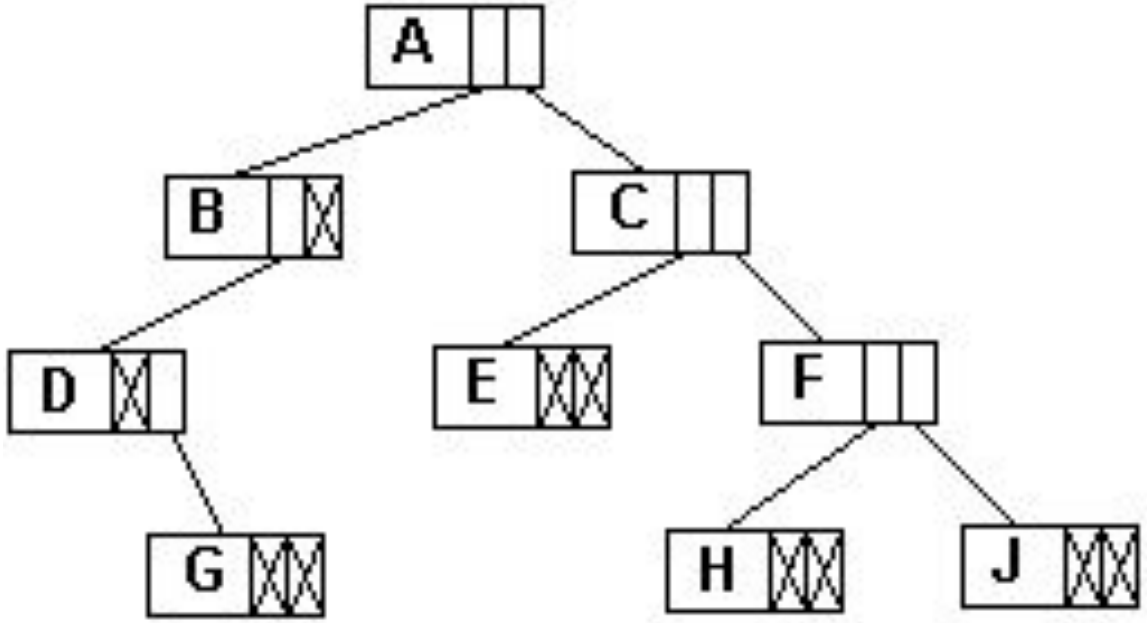
```
1.  void vtree(struct node **t)
2.  {
3.      char c;
4.      scanf("%c", &c);
5.      if (c != '.')
6.      {
7.          *t=(struct node *)malloc(sizeof(struct node ));
8.          (*t)->info=c;
9.          vtree(&((*t)->llink));
10.         vtree(&((*t)->rlink));
11.     }
12.     else
13.         *t=NULL;
14.  }
```

Как вводить дерево



a b d .. f . . c g . . e . .

Как вводить дерево



A B D . G . . . C E . . F H . . J . .

Функция рекурсивного обхода дерева

```
1.  void btree(struct node *t)
2.   {
3.      if (t != NULL)
4.      {
5.         btree(t->llink); // обойти левое поддерево
6.         printf("%c ", t->info); // попасть в корень
7.         btree(t->rlink); // обойти правое поддерево
8.       }
9.   }
```

Алгоритм стекового обхода дерева

ROOT – указатель на бинарное дерево;

A – стек, в который заносятся адреса еще не пройденных вершин;

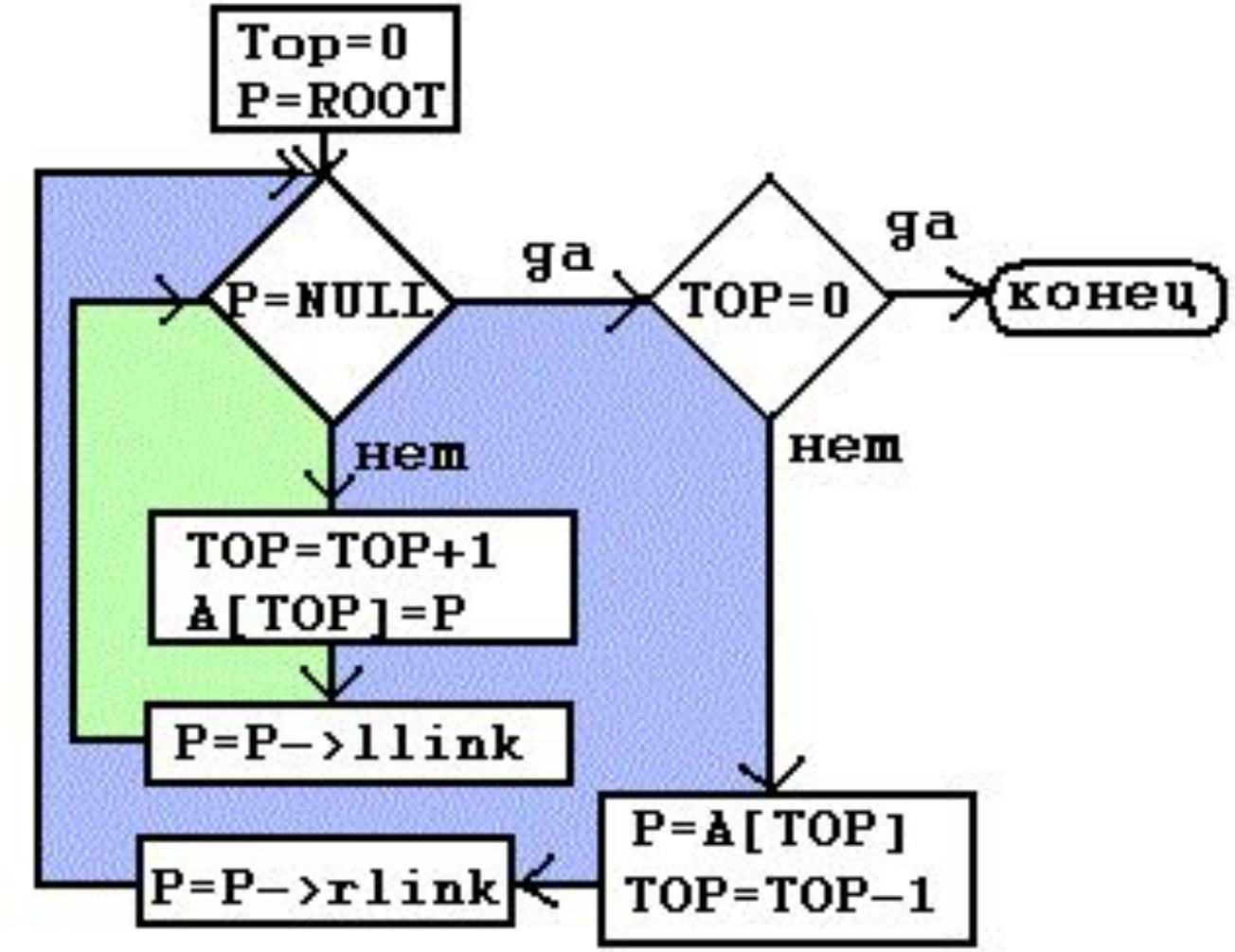
TOP – вершина стека;

P – рабочая переменная-указатель.

Алгоритм стекового обхода дерева

1. Начальная установка:
   TOP = 0; P = ROOT;
2. Если P = NULL, то перейти на 4 (конец ветви).
3. Вершину заносим в стек:
   TOP = TOP+1; A[TOP] = P;
   шаг по левой ветви: P = P->llink;
   перейти на 2 (идем по левой ветви).
4. Если TOP = 0, то КОНЕЦ.
5. Достаем вершину из стека:
   P = A[TOP]; TOP = TOP-1;
   Шаг по правой связи: P = P->rlink;
   перейти на 2.

Блок-схема алгоритма обхода бинарного дерева



Обход бинарного дерева

A: [a] [a,b] [a,b,c] [a,b,f]
[a,b,f,g] [a,b,f,g,m]
[a,b,f,g,n] [a,b,f,g]
[a,b,f,n] [a,b,f] [a,b]
[a,d] [a] [r] [s] [ ]