Binary search tree

construct, traverse, display.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <process.h>
struct node
{
    int info;
    struct node *rlink;
    struct node *llink;
};
typedef struct node *NODE;
NODE getnode()
{
    NODE x;
    x=(NODE) malloc (sizeof( struct node ));
    if (x==NULL)
    {
        printf("mem full\n");
        exit(0);
    }
    return x;
}
NODE insert(NODE root, int item)
{
    NODE temp, cur, prev;
    temp = getnode();
    temp->rlink= NULL;
    temp->llink = NULL;
    temp->info = item;
    if (root==NULL)
```

```
return temp;
prev = NULL;
cur = root;
while(cur != NULL)
{
    prev = cur;
    cur = (item < cur->info) ? cur->llink : cur->rlink;
}
if (item < prev->info)
    prev->llink = temp;
else
    prev->rlink = temp;
    return root;
}
void display (NODE root, int l)
{
    int j;
    if (root != NULL)
    {
        display (root->rlink, l+1);
        for (j=0; j<l; j++)
            printf("  ");
        printf("%d\n", root->info);
        display (root->llink, l+1);
    }
}
void preorder (NODE root)
{
    if (root != NULL)
    {
        pf ("%d\n", root->info);
        preorder (root->llink);
        preorder (root->rlink);
    }
}
```

```
void postorder(NODE root)
{
    if (root != NULL)
    {
        postorder(root->llink);
        postorder(root->rlink);
        printf("%d\n", root->info);
    }
}
void inorder(NODE root)
{
    if (root != NULL)
    {
        inorder(root->llink);
        printf("%d\n", root->info);
        inorder(root->rlink);
    }
}
int main()
{
    int item, choice;
    NODE root = NULL;
    for(;;)
    {
        pf("\n1. Insert \n2. display \n3. preord \n4. post
            \n5. in \n6. exit \n");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: printf("enter item\n");
                    scanf("%d", &item);
                    root = insert(root, item);
                    break;
```

```
case 2: display (root, 0);
        break;
case 3: preorder (root);
        break;
case 4: postorder (root);
        break;

case 5: inorder (root)
        break

case 6: exit (0);
        break
default: continue;
    }
  }
}
```