# Building a binary tree in C

The following program shows how to build a binary tree in a C program. It uses dynamic memory allocation, pointers and recursion. A binary tree is a very useful data-structure, since it allows efficient insertion, searching and deletion in a sorted list. As such a tree is essentially a recursively defined structure, recursive programming is the natural and efficient way to handle it.

*tree*

*empty*

*node left-branch right-branch*

*left-branch*

*tree*

*right-branch*

*tree*

#include<stdlib.h>

#include<stdio.h>

struct tree\_el {

int val;

struct tree\_el \* right, \* left;

};

typedef struct tree\_el node;

void insert(node \*\* tree, node \* item) {

if(!(\*tree)) {

\*tree = item;

return;

}

if(item->val<(\*tree)->val)

insert(&(\*tree)->left, item);

else if(item->val>(\*tree)->val)

insert(&(\*tree)->right, item);

}

void printout(node \* tree) {

if(tree->left) printout(tree->left);

printf("%d\n",tree->val);

if(tree->right) printout(tree->right);

}

void main() {

node \* curr, \* root;

int i;

root = NULL;

for(i=1;i<=10;i++) {

curr = (node \*)malloc(sizeof(node));

curr->left = curr->right = NULL;

curr->val = rand();

insert(&root, curr);

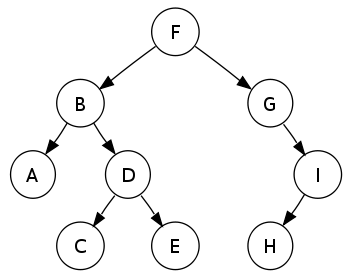
}

printout(root);

}

**C Program to Implement Binary Search Tree Traversal**

C Program to implement Binary Search Tree Traversal

[](http://img.c4learn.com/2012/01/350px-Sorted_binary_tree.svg_1.png)



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | Preorder traversal sequence : F, B, A, D, C, E, G, I, H     (root, left, right)  Inorder traversal sequence  : A, B, C, D, E, F, G, H, I     (left, root, right)  Postorder traversal sequence: A, C, E, D, B, H, I, G, F     (left, right, root) |

**Reference :** <http://en.wikipedia.org/wiki/Tree_traversal>

**Program :**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153 | # include <stdio.h>  # include <conio.h>  # include <stdlib.h>    typedef struct BST {     int data;     struct BST \*lchild, \*rchild;  } node;    void insert(node \*, node \*);  void inorder(node \*);  void preorder(node \*);  void postorder(node \*);  node \*search(node \*, int, node \*\*);    void main() {     int choice;     char ans = 'N';     int key;     node \*new\_node, \*root, \*tmp, \*parent;     node \*get\_node();     root = NULL;     clrscr();       printf("\nProgram For Binary Search Tree ");     do {        printf("\n1.Create");        printf("\n2.Search");        printf("\n3.Recursive Traversals");        printf("\n4.Exit");        printf("\nEnter your choice :");        scanf("%d", &choice);          switch (choice) {        case 1:           do {              new\_node = get\_node();              printf("\nEnter The Element ");              scanf("%d", &new\_node->data);                if (root == NULL) /\* Tree is not Created \*/                 root = new\_node;              else                 insert(root, new\_node);                printf("\nWant To enter More Elements?(y/n)");              ans = getch();           } while (ans == 'y');           break;          case 2:           printf("\nEnter Element to be searched :");           scanf("%d", &key);             tmp = search(root, key, &parent);           printf("\nParent of node %d is %d", tmp->data, parent->data);           break;          case 3:           if (root == NULL)              printf("Tree Is Not Created");           else {              printf("\nThe Inorder display : ");              inorder(root);              printf("\nThe Preorder display : ");              preorder(root);              printf("\nThe Postorder display : ");              postorder(root);           }           break;        }     } while (choice != 4);  }  /\*  Get new Node  \*/  node \*get\_node() {     node \*temp;     temp = (node \*) malloc(sizeof(node));     temp->lchild = NULL;     temp->rchild = NULL;     return temp;  }  /\*  This function is for creating a binary search tree  \*/  void insert(node \*root, node \*new\_node) {     if (new\_node->data < root->data) {        if (root->lchild == NULL)           root->lchild = new\_node;        else           insert(root->lchild, new\_node);     }       if (new\_node->data > root->data) {        if (root->rchild == NULL)           root->rchild = new\_node;        else           insert(root->rchild, new\_node);     }  }  /\*  This function is for searching the node from  binary Search Tree  \*/  node \*search(node \*root, int key, node \*\*parent) {     node \*temp;     temp = root;     while (temp != NULL) {        if (temp->data == key) {           printf("\nThe %d Element is Present", temp->data);           return temp;        }        \*parent = temp;          if (temp->data > key)           temp = temp->lchild;        else           temp = temp->rchild;     }     return NULL;  }  /\*  This function displays the tree in inorder fashion  \*/  void inorder(node \*temp) {     if (temp != NULL) {        inorder(temp->lchild);        printf("%d", temp->data);        inorder(temp->rchild);     }  }  /\*  This function displays the tree in preorder fashion  \*/  void preorder(node \*temp) {     if (temp != NULL) {        printf("%d", temp->data);        preorder(temp->lchild);        preorder(temp->rchild);     }  }    /\*  This function displays the tree in postorder fashion  \*/  void postorder(node \*temp) {     if (temp != NULL) {        postorder(temp->lchild);        postorder(temp->rchild);        printf("%d", temp->data);     }  } |

**Explanation :**

* **get\_node()** function will **allocate memory dynamically** and allocate one node.
* if below condition is satisfied then we can say that we are going to create first node of the tree. **(i.e Tree is empty and this created node is very first node**)



|  |  |
| --- | --- |
| 1 | if(root == NULL) |

* If condition does not satisfied then we can say that we have already node in a tree. (**i.e this node which we have created is not a first node**)

Display Tree

To display tree we have 3 traversal Techniques –

1. In-Order Traversal
2. Pre-Order Traversal
3. Post-Order Traversal

**Algorithm for Preorder Traversal of Binary Search Tree :**



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | preorder(node)    {    if node = null     then       return    else       print Node Data       Go to Left Child of node       Go to Right Child of node    } |

Similarly Post order and inorder traversal works.