1. **Activity: Given breadth first traversal realise a complete binary tree with an array. Implement the following**
   1. **Given index of a node, identify node’s children and parents**
   2. **Identify and print the available parent-leftchild-rightchild node combinations available in the CBT**
   3. **Identify and print the leaf nodes in the CBT**

**NOTE: In array implementation, the breadth first traversal is how we save the tree in an array. So we can simply input the breadth first traversal in an array and use it as the tree**

**Algorithm:**

1. Function IdentifyRelatives:
   1. If ((2\*index + 1) < size) Lchild = (2\*index + 1) else Lchild = -1;
   2. If ((2\*index + 2) < size) Lchild = (2\*index + 1) else Lchild = -1;
   3. parent = ceil( index / 2 ) - 1;
   4. if ( parent >=0 ) print “Parent: “, tree[parent] else print “Parent not found”
   5. if ( lchild >=0 ) print “Left Child: “, tree[lchild] else print “Left Child not found”
   6. if ( rchild >=0 ) print “Right Child: “, tree[rchild] else print “Right Child not found”
2. Function PrintCombos:
   1. If ( (2\*index + 1) >= size) return;
   2. if ( (2\*index + 2) >= size) return;
   3. print “Parent: “ + tree[index] + “Left: “ + tree[(2\*index + 1)] + “Right: “ + tree[(2\*index + 2)]
   4. printCombos(tree, (2\*index + 1));
   5. printCombos(tree, (2\*index + 2));
3. function PrintLeaves:
   1. if (index >= size) return;
   2. if ( (2\*index + 1) >= size && (2\*index + 2) >= size) printf("%d ", tree[index]);
   3. if ( (2\*index + 1) < size) printLeaves(tree, (2\*index + 1));
   4. if ( (2\*index + 2) < size) printLeaves(tree, (2\*index + 2));

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <ctype.h>

int size, space\_diff = 10;

void printTree(int tree[], int rootIndex, int space){

    if(rootIndex >= size) return;

    space += space\_diff;

    printTree(tree, (2\*rootIndex + 2), space);

    printf("\n");

    for (int i = 0; i < space; i++) printf(" ");

    printf("%d\n", tree[rootIndex]);

    printTree(tree, (2\*rootIndex + 1), space);

}

void identifyRelatives(int tree[], int index){

    int lchild = ((2\*index + 1) < size)?(2\*index + 1):-1;

    int rchild = ((2\*index + 2) < size)?(2\*index + 2):-1;

    int parent = ((int)ceil(((float)index)/2)) - 1;

    printf("\nCurrent: %d\n", tree[index]);

    (parent == -1)?printf("Parent: Does Not Exist\n"):printf("Parent: %d\n", tree[parent]);

    (lchild == -1)?printf("Left Child: Does Not Exist\n"):printf("Left Child: %d\n", tree[lchild]);

    (rchild == -1)?printf("Right Child: Does Not Exist\n"):printf("Right Child: %d\n", tree[rchild]);

}

void printCombos(int tree[], int index){

    if((2\*index + 1) >= size) return;

    if((2\*index + 2) >= size) return;

    printf("P: %d -> L: %d -> R: %d\n", tree[index], tree[(2\*index + 1)], tree[(2\*index + 2)]);

    //Go to left subtree

    printCombos(tree, (2\*index + 1));

    //Go to right subtree

    printCombos(tree, (2\*index + 2));

}

void printLeaves(int tree[], int index){

    if(index >= size)return;

    if((2\*index + 1) >= size && (2\*index + 2) >= size){

        printf("%d ", tree[index]);

    }

    if((2\*index + 1) < size) printLeaves(tree, (2\*index + 1));

    if((2\*index + 2) < size) printLeaves(tree, (2\*index + 2));

}

int main(){

    printf("Enter Size of tree: ");

    scanf("%d", &size);

    int tree[size];

    printf("Enter traversal: ");

    for (int i = 0; i < size; i++)

    {

        scanf("%d", &tree[i]);

    }

    printf("\n");

    printTree(tree, 0, -10);

    int inf = 1;

    char choice;

    while(inf){

        while((getchar()) != '\n');

        printf("Do you want to find relatives of a node: ");

        scanf("%c", &choice);

        switch (tolower(choice))

        {

        case 'y': ;

            int indx;

            printf("Enter node index: ");

            scanf("%d", &indx);

            identifyRelatives(tree, indx);

            break;

        case 'n':

            inf = 0;

            break;

        default:

            printf("Invalid selection\n");

            getchar();

            break;

        }

    }

    printf("Possible Parent-Left-Right combinations -> \n");

    printCombos(tree, 0);

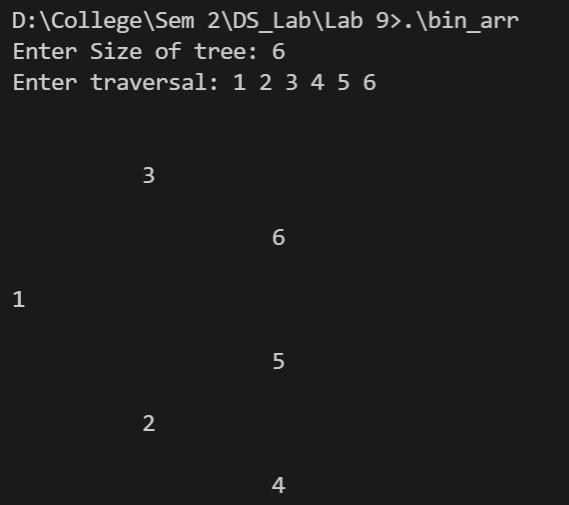
    printf("Leaf Nodes: ");

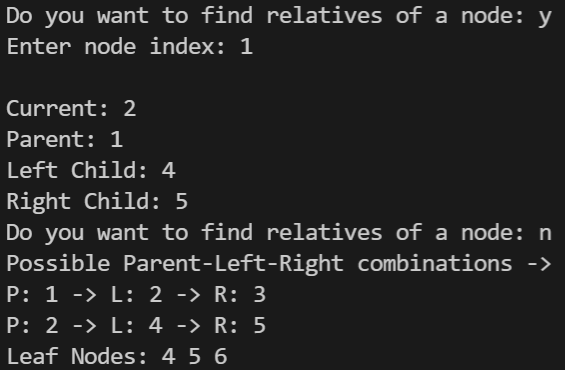
    printLeaves(tree, 0);

    return 0;

}

**Screenshot of compilation and execution:**

****

****

1. **Activity: Construct a binary search tree using linked list**

**Algorithm:**

1. Insert function:
   1. if (node == NULL) return newNode(key);
   2. if (key < node->data) node->left = insert(node->left, key);
   3. else if (key > node->data) node->right = insert(node->right, key);
   4. return node;
2. newNode function
   1. struct node\* temp = (struct node\*)malloc(sizeof(struct node));
   2. temp->data = item;
   3. temp->left = temp->right = NULL;
   4. return temp;

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <ctype.h>

int space\_diff = 10;

struct node{

    int data;

    struct node \*left, \*right;

};

struct node\* newNode(int item)

{

    struct node\* temp

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

    temp->data = item;

    // Since every inserted node will be a leaf node

    temp->left = temp->right = NULL;

    return temp;

}

struct node\* insert(struct node\* node, int key)

{

    if (node == NULL)

        return newNode(key);

    if (key < node->data)

        node->left = insert(node->left, key);

    else if (key > node->data)

        node->right = insert(node->right, key);

    /\* return the (unchanged) node pointer \*/

    return node;

}

void printTree(struct node \*root, int space)

{

    if (root == NULL)

        return;

    space += space\_diff;

    printTree(root->right, space);

    // Print current node after space

    // count

    printf("\n");

    for (int i = space\_diff; i < space; i++)

        printf(" ");

    printf("%d\n", root->data);

    // Process left child

    printTree(root->left, space);

}

int main(){

    srand(time(0));

    printf("Binary Search tree using linked lists\n\n");

    int root\_num = rand() % 100;

    struct node\* root = NULL;

    root = insert(root, root\_num);

    printf("Root: %d\n", root\_num);

    int inf = 1;

    char choice;

    while(inf){

        printf("Do you want to insert node: ");

        scanf("%c", &choice);

        switch (tolower(choice))

        {

        case 'y': ;

            int data;

            printf("Enter node data: ");

            scanf("%d", &data);

            insert(root, data);

            break;

        case 'n':

            inf = 0;

            break;

        default:

            printf("Invalid selection\n");

            getchar();

            break;

        }

        while((getchar()) != '\n');

    }

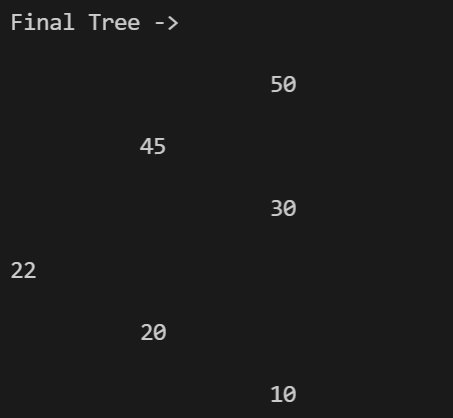
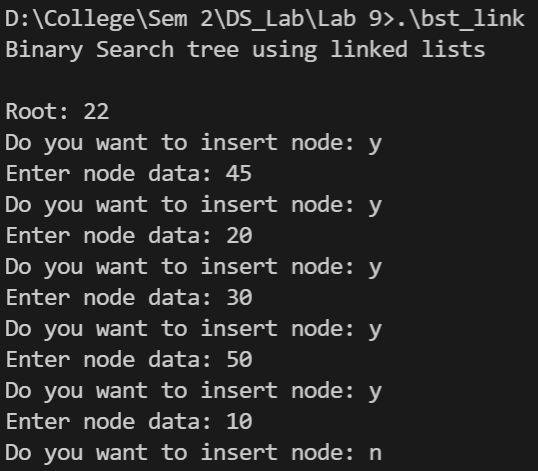
    printf("\nFinal Tree -> \n");

    printTree(root, 0);

    return 0;

}

**Screenshot of compilation and execution:**

****

1. **Activity: Perform inorder, preorder and postorder traversal on the BST contructed in 9.2**

**Algorithm:**

1. printInOrder function:
   1. if (root == NULL) return;
   2. printInOrder(root->left);
   3. printf("%d ", root->data);
   4. printInOrder(root->right);
2. printPreOrder function:
3. if (root == NULL) return;
4. printf("%d ", root->data);
5. printPreOrder(root->left);
6. printPreOrder(root->right);
7. printPostOrder function:
8. if (root == NULL) return;
9. printPostOrder(root->left);
10. printPostOrder(root->right);
11. printf("%d ", root->data);

**Program:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <ctype.h>

int space\_diff = 10;

struct node{

    int data;

    struct node \*left, \*right;

};

struct node\* newNode(int item)

{

    struct node\* temp

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

    temp->data = item;

    // Since every inserted node will be a leaf node

    temp->left = temp->right = NULL;

    return temp;

}

struct node\* insert(struct node\* node, int key)

{

    if (node == NULL)

        return newNode(key);

    if (key < node->data)

        node->left = insert(node->left, key);

    else if (key > node->data)

        node->right = insert(node->right, key);

    /\* return the (unchanged) node pointer \*/

    return node;

}

void printTree(struct node \*root, int space)

{

    if (root == NULL)

        return;

    space += space\_diff;

    printTree(root->right, space);

    // Print current node after space

    // count

    printf("\n");

    for (int i = space\_diff; i < space; i++)

        printf(" ");

    printf("%d\n", root->data);

    // Process left child

    printTree(root->left, space);

}

void printInOrder(struct node \*root){

    if (root == NULL) return;

    printInOrder(root->left);

    printf("%d ", root->data);

    printInOrder(root->right);

}

void printPreOrder(struct node \*root){

    if (root == NULL) return;

    printf("%d ", root->data);

    printPreOrder(root->left);

    printPreOrder(root->right);

}

void printPostOrder(struct node \*root){

    if (root == NULL) return;

    printPostOrder(root->left);

    printPostOrder(root->right);

    printf("%d ", root->data);

}

int main(){

    srand(time(0));

    printf("Binary Search tree using linked lists\n\n");

    int root\_num = rand() % 100;

    struct node\* root = NULL;

    root = insert(root, root\_num);

    printf("Root: %d\n", root\_num);

    int inf = 1;

    char choice;

    while(inf){

        printf("Do you want to insert node: ");

        scanf("%c", &choice);

        switch (tolower(choice))

        {

        case 'y': ;

            int data;

            printf("Enter node data: ");

            scanf("%d", &data);

            insert(root, data);

            break;

        case 'n':

            inf = 0;

            break;

        default:

            printf("Invalid selection\n");

            getchar();

            break;

        }

        while((getchar()) != '\n');

    }

    printf("\nFinal Tree -> \n");

    printTree(root, 0);

    printf("\nIn-order traversal: ");

    printInOrder(root);

    printf("\nPre-order traversal: ");

    printPreOrder(root);

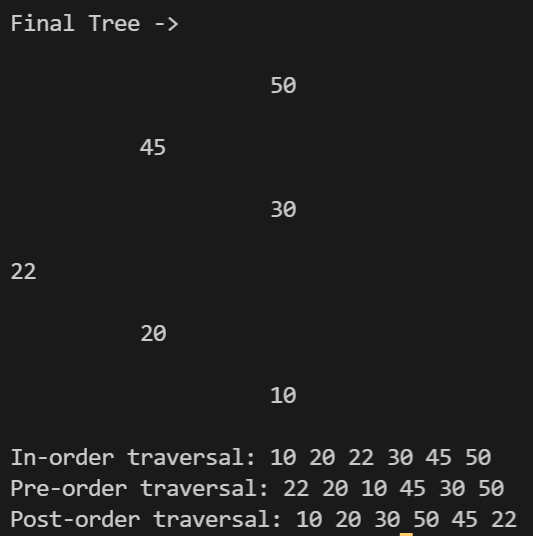
    printf("\nPost-order traversal: ");

    printPostOrder(root);

    return 0;

}

**Screenshot of compilation and execution:**

****