1- (Binary Tree Search) Write function binaryTreeSearch that attempts to locate a specified value in a binary search tree. The function should take as arguments a pointer to the root node of the binary tree and a search key to be located. If the node containing the search key is found, the function should return a pointer to that node; otherwise, the function should return a NULL pointer.

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

/* TreeNode structure definition */
struct TreeNode {
	struct TreeNode *leftPtr; /* pointer to left subtree */
	int data; /* node data */
	struct TreeNode *rightPtr; /* pointer to right subtree */
};/* end struct TreeNode */

typedef struct TreeNode TreeNode;
typedef TreeNode *TreeNodePtr;

/* function prototypes */
void insertNode(TreeNodePtr *treePtr, intvalue);
TreeNodePtr binaryTreeSearch(TreeNodePtr treePtr, const int key);

int main() {
	int i; /* loop counter */
	int item; /* random value to insert in tree */
	int searchKey; /* value to search for */
	TreeNodePtr rootPtr = NULL; /* points to the tree root */
	TreeNodePtr searchResultPtr; /* pointer to search result */

	srand(time( NULL)); /* randomize */
	printf("The numbers being placed in the tree are:\n");

	/* insert random values between 1 and 20 in the tree */
	for (i = 1; i <= 10; i++) {
		item = 1 + rand() % 20;
		printf("%3d", item);
		insertNode(&rootPtr, item);
	} /* end for */

	/* prompt user and read integer search key */
	printf("\n\nEnter an integer to search for: ");
	scanf("%d", &searchKey);

	searchResultPtr = binaryTreeSearch(rootPtr, searchKey);`
```

```
      /* if searchKey not found */
      if (searchResultPtr == NULL) {
            printf("\n%d was not found in the tree.\n\n", searchKey);
      }/* end if */
      else { /* if key found */
            printf("\n%d was found in the tree.\n\n", searchResultPtr-
>data);
      } /* end else */

      return 0; /* indicate successful termination */

} /* end main */

/* insert a node into the tree */
Void insertNode(TreeNodePtr *treePtr, int value) {

      /* if treePtr is NULL */
      if (*treePtr == NULL) {

            /* dynamically allocate memory */
            *treePtr = malloc(sizeof(TreeNode));

            /* if memory was allocated, insert node */
            if (*treePtr != NULL) {
                  (*treePtr)->data = value;
                  (*treePtr)->leftPtr = NULL;
                  (*treePtr)->rightPtr = NULL;
            }/* end if */
            else {
                  printf("%d not inserted. No memory available.\n", value);
            } /* end else */

      }/* end if */
      else { /* recursively call insertNode */

            /* insert node in left subtree */
            if (value < (*treePtr)->data) {
                  insertNode(&((*treePtr)->leftPtr), value);
            } /* end if */
            else {

                  /* insert node in right subtree */
                  if (value > (*treePtr)->data) {
                        insertNode(&((*treePtr)->rightPtr), value);
                  } /* end if */
                  else { /* duplicate value */
                        printf("dup");
                  }/* end else */

            } /* end else */

      } /* end else */
```

```
}/* end function insertNode */

/* search for key in tree */
TreeNodePtr binaryTreeSearch(TreeNodePtr treePtr, const int key) {

      /* traverse the tree inOrder */
      if (treePtr == NULL) {
            return NULL;/* key not found */
      } /* end if */
      else if (treePtr->data == key) {
            return treePtr; /* key found */
      } /* end else if */
      else if (key < treePtr->data) {
            binaryTreeSearch(treePtr->leftPtr, key); /* search left */
      } /* end else if */
      else if (key > treePtr->data) {
            binaryTreeSearch(treePtr->rightPtr, key); /* search right */
      } /* end else if */

} /* end function binaryTreeSearch */
```

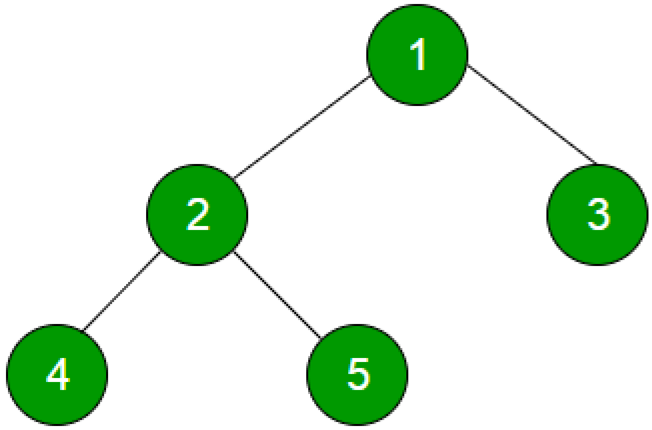
```
The numbers being placed in the tree are:

18 9 7 2 13 2dup 10 1 19 2dup

Enter an integer to search for: 8

8 was not found in the tree.
```

2- Level order traversal of a tree is breadth first traversal for the tree.



Level order traversal of the above tree is 1 2 3 4 5

```c
// Recursive C program for level order traversal of Binary Tree
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node* left, *right;
};

/* Function protoypes */
void printGivenLevel(struct node* root, int level);
int height(struct node* node);
struct node* newNode(int data);

/* Function to print level order traversal a tree*/
void printLevelOrder(struct node* root)
{
    int h = height(root);
    int i;
    for (i=1; i<=h; i++)
        printGivenLevel(root, i);
}
  /* Print nodes at a given level */
void printGivenLevel(struct node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        printf("%d ", root->data);
    else if (level > 1)
    {
        printGivenLevel(root->left, level-1);
        printGivenLevel(root->right, level-1);
    }
}
```

```c
/* Compute the "height" of a tree -- the number of
    nodes along the longest path from the root node
    down to the farthest leaf node.*/
int height(struct node* node)
{
    if (node==NULL)
        return 0;
    else
    {
        /* compute the height of each subtree */
        int lheight = height(node->left);
        int rheight = height(node->right);

        /* use the larger one */
        if (lheight > rheight)
            return(lheight+1);
        else return(rheight+1);
    }
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
    struct node* node = (struct node*)
                        malloc(sizeof(struct node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;

    return(node);
}

/* Driver program to test above functions*/
int main()
{
    struct node *root = newNode(1);
    root->left        = newNode(2);
    root->right       = newNode(3);
    root->left->left  = newNode(4);
    root->left->right = newNode(5);

    printf("Level Order traversal of binary tree is \n");
    printLevelOrder(root);

    return 0;
}
```

3- Given a root of a tree, and an integer k. Print all the nodes which are at k distance from root.

For example, in the below tree, 4, 5 & 8 are at distance 2 from root.

```
      1
     / \
    2   3
   / \  /
  4   5 8
```

```c
#include <stdio.h>
#include <stdlib.h>

/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
   int data;
   struct node* left;
   struct node* right;
};

void printKDistant(struct node *root , int k)
{
   if(root == NULL)
      return;
   if( k == 0 )
   {
      printf( "%d ", root->data );
      return ;
   }
   else
   {
      printKDistant( root->left, k-1 ) ;
      printKDistant( root->right, k-1 ) ;
   }
}

/* Helper function that allocates a new node with the
   given data and NULL left and right pointers. */
struct node* newNode(int data)
{
  struct node* node = (struct node*)
                       malloc(sizeof(struct node));
  node->data = data;
  node->left = NULL;
  node->right = NULL;

  return(node);
}
```

```c
/* Driver program to test above functions*/
int main()
{

  /* Constructed binary tree is
            1
          /   \
        2       3
      /   \    /
    4      5   8
  */
  struct node *root = newNode(1);
  root->left        = newNode(2);
  root->right       = newNode(3);
  root->left->left  = newNode(4);
  root->left->right = newNode(5);
  root->right->left = newNode(8);

  printKDistant(root, 2);

  getchar();
  return 0;
}
```