EENG212/INFE212 Algorithms and Data Structures
Fall 2013-2014, **Date**: 6th Jan. 2014, **Duration**: 30 min
Lecturer: Prof. Dr. Hasan Amca

Student No: _____

Name: _____

Quiz 6

The following program is a binary search tree program that employs inserting a node and searching a binary tree for a desired value. Write an algorithm and draw a flowchart for the function **binaryTreeSearch** that attempts to locate a specified value in a binary search tree. Using the algorithm, write the missing function **binaryTreeSearch** in the program. 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.

```
/* Exercise 12.23 Solution */
#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, int value );
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 )
{

}
```