15

Case Study of Data Structure (Stack, Queue, Tree)

Example : Binary Tree Search

- **Problem definition (Exercise 12.19)**
  - **Write a function binaryTreeSearch**
  - **Attempt to locate a specified value in a binary search tree.**
  - **Input: a pointer to the root node of the binary tree and a search key to be located**
  - **Output: a pointer to that node (if found) or NULL (not found)**

Source Code (1/5)

```
1  /* Exercise 12.19 Solution */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* TreeNode structure definition */
7  struct TreeNode {
8     struct TreeNode *leftPtr;  /* pointer to left subtree */
9     int data;                  /* node data */
10    struct TreeNode *rightPtr; /* pointer to right subtree */
11 }; /* end struct TreeNode */
12
13 typedef struct TreeNode TreeNode;
14 typedef TreeNode *TreeNodePtr;
15
16 /* function prototypes */
17 void insertNode( TreeNodePtr *treePtr, int value );
18 TreeNodePtr binaryTreeSearch( TreeNodePtr treePtr, const int key );
19
20 int main( void )
21 {
22    int i;                     /* loop counter */
23    int item;                  /* random value to insert in tree */
24    int searchKey;             /* value to search for */
25    TreeNodePtr rootPtr = NULL;  /* points to the tree root */
26    TreeNodePtr searchResultPtr; /* pointer to search result */
```

Source Code (2/5)

```
27
28      srand( time( NULL ) ); /* randomize */
29      printf( "The numbers being placed in the tree are:\n" );
30
31      /* insert random values between 1 and 20 in the tree */
32      for ( i = 1; i <= 10; i++ ) {
33         item = 1 + rand() % 20;
34         printf( "%3d", item );
35         insertNode( &rootPtr, item );
36      } /* end for */
37
38      /* prompt user and read integer search key */
39      printf( "\n\nEnter an integer to search for: " );
40      scanf( "%d", &searchKey );
41
42      searchResultPtr = binaryTreeSearch( rootPtr, searchKey );
43
44      /* if searchKey not found */
45      if ( searchResultPtr == NULL ) {
46         printf( "\n%d was not found in the tree.\n\n", searchKey );
47      } /* end if */
48      else { /* if key found */
```

Source Code (3/5)

```
49          printf( "\n%d was found in the tree.\n\n",
50             searchResultPtr->data );
51       } /* end else */
52
53       return 0; /* indicate successful termination */
54
55    } /* end main */
56
57    /* insert a node into the tree */
58    void insertNode( TreeNodePtr *treePtr, int value )
59    {
60
61       /* if treePtr is NULL */
62       if ( *treePtr == NULL ) {
63
64          /* dynamically allocate memory */
65          *treePtr = malloc( sizeof( TreeNode ) );
66
67          /* if memory was allocated, insert node */
68          if ( *treePtr != NULL ) {
69             ( *treePtr )->data = value;
70             ( *treePtr )->leftPtr = NULL;
71             ( *treePtr )->rightPtr = NULL;
72          } /* end if */
73          else {
74             printf( "%d not inserted. No memory available.\n", value );
75          } /* end else */
```

Source Code (4/5)

```
76
77       } /* end if */
78       else { /* recursively call insertNode */
79
80          /* insert node in left subtree */
81          if ( value < ( *treePtr )->data ) {
82             insertNode( &( ( *treePtr )->leftPtr ), value );
83          } /* end if */
84          else {
85
86             /* insert node in right subtree */
87             if ( value > ( *treePtr )->data ) {
88                insertNode( &( ( *treePtr )->rightPtr ), value );
89             } /* end if */
90             else { /* duplicate value */
91                printf( "dup" );
92             } /* end else */
93
94          } /* end else */
95
96       } /* end else */
97
98    } /* end function insertNode */
99
100   /* search for key in tree */
101   TreeNodePtr binaryTreeSearch( TreeNodePtr treePtr, const int key )
102   {
```

Source Code (5/5)

```
103
104     /* traverse the tree inOrder */
105     if ( treePtr == NULL ) {
106        return NULL; /* key not found */
107     } /* end if */
108     else if ( treePtr->data == key ) {
109        return treePtr; /* key found */
110     } /* end else if */
111     else if ( key < treePtr->data ) {
112        return binaryTreeSearch( treePtr->leftPtr, key ); /* search left */
113     } /* end else if */
114     else if ( key > treePtr->data ) {
115        return binaryTreeSearch( treePtr->rightPtr, key ); /*search right */
116     } /* end else if */
117
118  } /* 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.
```