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

/* Solution to Problem 1, part 1: a type for representing binary search trees
of integer values */

struct node{
 int val;
 struct node *left, *right;
};

struct node *create(int val) {
 struct node *new =  (struct node *)malloc(sizeof(struct node));
 new->val = val;
 new->left = NULL;
 new->right = NULL;
 return new;
}

/* Solution to Problem 1, part 2: a function for determining if a given integer
appears in a given integer binary search tree*/

int isEqual(int num1, int num2) {
 if (num1 == num2) {
  return 1;
 }
 return 0;
}

int compare(int num1, int num2) {
 return num1 - num2;
}

int member(struct node* node, int val, int (*eq)(int, int),
 int (*ord)(int, int)) {
  if (node == NULL ) {
   return 0;
  }
  if ((*eq)(node->val, val) == 1) {
   return 1;
  }

  if ((*ord)(node->val, val) < 0) {
   return member(node->right, val, isEqual, compare);
  }

  return member(node->left, val, isEqual, compare);
}

/* Solution to Problem 1, part 3: a function for inserting a given integer into
a given integer binary search tree */

struct node* insert(struct node* node, int val, int (*ord)(int, int)) {
```

```
 if (node == NULL) {
  return create(val);
 }

 /* left Subtree is unchanged */
 if ((*ord)(node->val, val) < 0) {
  node->right  = insert(node->right, val, compare);
 }
 /* Right Subtree is unchanged */
 else if ((*ord)(node->val, val) > 0) {
  node->left = insert(node->left, val, compare);
 }

 return node;
}

/* Solution to Problem 1, part 4: a function to print elements in an integer
binary search tree using an inorder traversal*/

void printVal(int val)  {
 printf("%d\n", val);
}

void printtree(struct node *node, void (*prt)(int)) {
 if (node != NULL) {

  printtree(node->left, printVal);

  (*prt)(node->val);

  printtree(node->right, printVal);
 }
}

void memberTest(int res, int val) {
 if (res == 1) {
  printf("%d is a member of the tree\n", val);
 } else {
  printf("%d is not a member of the tree\n", val);
 }
}

int main() {
 /* Test 1 for empty tree */
 printf("Test 1: Empty tree\n");
 struct node *tree = NULL;
 int val = 10;
 int res = member(tree, val, isEqual, compare);
 memberTest(res, val);

 /* Test 2 for 1 element tree */
 printf("\nTest 2: Inserting 1 element\n");
 struct node *newTree = insert(tree, 10, compare);
 printf("Tree:\n");
 printtree(newTree, printVal);
 res = member(newTree, val, isEqual, compare);
```

```
110   memberTest(res, val);
111
112   /* Test 3 for a right heavy tree */
113   printf("\nTest 3: right heavy tree\n");
114   insert(newTree, 11, compare);
115   insert(newTree, 12, compare);
116   insert(newTree, 13, compare);
117   insert(newTree, 14, compare);
118
119   printf("Tree:\n");
120   printtree(newTree, printVal);
121   val = 14;
122   res = member(newTree, val, isEqual, compare);
123   memberTest(res, val);
124   val = 9;
125   res = member(newTree, val, isEqual, compare);
126   memberTest(res, val);
127
128   /* Test 4 for a left heavy tree */
129   printf("\nTest 4: Left heavy tree\n");
130   struct node *newTree2 = insert(tree, 10, compare);
131   insert(newTree2, 9, compare);
132   insert(newTree2, 8, compare);
133   insert(newTree2, 7, compare);
134   insert(newTree2, 6, compare);
135
136   printf("Tree:\n");
137   printtree(newTree2, printVal);
138   val = 14;
139   res = member(newTree2, val, isEqual, compare);
140   memberTest(res, val);
141   val = 9;
142   res = member(newTree2, val, isEqual, compare);
143   memberTest(res, val);
144
145   /* Test 5 for a right heavy tree */
146   printf("\nTest 5: Average case\n");
147   insert(newTree, 9, compare);
148   insert(newTree, 8, compare);
149   insert(newTree, 7, compare);
150   insert(newTree, 6, compare);
151
152   printf("Tree:\n");
153   printtree(newTree, printVal);
154   val = 14;
155   res = member(newTree, val, isEqual, compare);
156   memberTest(res, val);
157   val = 9;
158   res = member(newTree, val, isEqual, compare);
159   memberTest(res, val);
160
161   return 0;
162 }
```