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

/* Solution to Problem 3, part 1: a type that is capable of representing binary
 search trees of arbitrary types of elements. */
typedef enum {integer, string, person} kind;

struct Person{
 char * first;
 char * last;
}Person;

struct node{
 void *elem;
 struct node *left, *right;
};

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

/* Solution to Problem 3, part 2: a function for determining if an arbitrary
value appears in a given binary search tree*/
int isEqual(void *elem1, void *elem2, kind k) {
 if (k == 0) {
  int val1 = *((int *) elem1);
  int val2 = *((int *) elem2);
  if (val1 == val2) {
   return 1;
  }
 }
 else if (k == 1) {
  char * str1 = (char*) elem1;
  char * str2 = (char*) elem2;
  if (strcmp(str1, str2) == 0) {
   return 1;
  }
 }
 else if (k == 2) {
  struct Person * str1 = elem1;
  struct Person * str2 = elem2;
  if (strcmp(str1->last, str2->last) == 0) {
   return 1;
  }
 }
 return 0;

}
```

```
54
55  int compare(void *elem1, void *elem2, kind k) {
56   if (k == 0) {
57    int val1 = *((int *) elem1);
58    int val2 = *((int *) elem2);
59    return val1 - val2;
60   }
61   else if (k == 1){
62    char * str1 = (char*) elem1;
63    char * str2 = (char*) elem2;
64    return strcmp(str1, str2);
65   }
66   else if (k == 2){
67    struct Person * str1 = elem1;
68    struct Person * str2 = elem2;
69    return strcmp(str1->last, str2->last);
70   }
71  }
72
73  int member(struct node* node, void *val,
74   int (*eq)(void*,void*, kind),
75   int (*ord)(void*, void*, kind), kind k) {
76
77    if (node == NULL ) {
78     return 0;
79    }
80
81    if ((*eq)(node->elem, val, k) == 1) {
82     return 1;
83    }
84
85    if ((*ord)(node->elem, val, k) < 0) {
86     return member(node->right, val, isEqual, compare, k);
87    }
88
89    return member(node->left, val, isEqual, compare, k);
90  }
91
92  /* Solution to Problem 3, part 3: a function for inserting an arbitrary value
93   into a given binary search tree */
94
95   struct node* insert(struct node* node, void* val,
96    int (*ord)(void*, void*, kind), kind k) {
97    if (node == NULL) {
98     return create(val);
99    }
100
101   /* left Subtree is unchanged */
102   if ((*ord)(node->elem, val, k) < 0) {
103    node->right  = insert(node->right, val, compare, k);
104   }
105   /* Right Subtree is unchanged */
106   else if ((*ord)(node->elem, val, k) > 0) {
107    node->left = insert(node->left, val, compare, k);
108   }
109
```

```
110   return node;
111  }
112
113  /* Solution to Problem 3, part 4: a function to print elements in an integer
114   or string binary search tree using an inorder traversal*/
115
116  void printVal(void *val, kind k)  {
117   if (k == 0) {
118    int num = *((int *) val);
119    printf("%d\n", num);
120   }
121   else if (k == 1) {
122    char * str1 = (char*) val;
123    printf("%s\n", str1);
124   }
125   else if (k == 2) {
126    struct Person * person = val;
127    printf("%s %s\n", person->first, person->last);
128   }
129  }
130
131  void printtree(struct node *node, void (*prt)(void*, kind k), kind k) {
132   if (node != NULL) {
133
134    printtree(node->left, printVal, k);
135
136    (*prt)(node->elem, k);
137
138    printtree(node->right, printVal, k);
139   }
140  }
141
142  void memberTest(int res) {
143   if (res == 1) {
144    printf("is a member of the tree\n");
145   } else {
146    printf("is not a member of the tree\n");
147   }
148  }
149
150 int main(){
151  struct node *tree = NULL;
152
153  /* Test 1 For Integer tree */
154  int num1 = 10;
155  printf("Test 1: Integer tree\n");
156  struct node *intTree = insert(tree, &num1, compare, integer);
157  int num2 = 3;
158  insert(intTree, &num2, compare, integer);
159  int num3 = 22;
160  insert(intTree, &num3, compare, integer);
161  int num4 = 15;
162  insert(intTree, &num4, compare, integer);
163  int num5 = 9;
164  insert(intTree, &num5, compare, integer);
165  printf("Tree:\n");
```

```
166  printtree(intTree, printVal, integer);
167
168  int num = 10;
169  int res = member(intTree, &num, isEqual, compare, integer);
170  printf("%d ", num);
171  memberTest(res);
172
173  num = 12;
174  res = member(intTree, &num, isEqual, compare, integer);
175  printf("%d ", num);
176  memberTest(res);
177
178  /* Test 2 For String tree */
179  printf("\nTest 2: String tree\n");
180  char * str1 = "ardvark";
181  struct node *strTree = insert(tree, str1, compare, string);
182  char * str2 = "goose";
183  insert(strTree, str2, compare, string);
184  char * str3 = "beetle";
185  insert(strTree, str3, compare, string);
186  char * str4 = "zebra";
187  insert(strTree, str4, compare, string);
188  char * str5 = "eagle";
189  insert(strTree, str5, compare, string);
190  printf("Tree:\n");
191  printtree(strTree, printVal, string);
192
193  char* str6 = "ardvark";
194  res = member(strTree, str6, isEqual, compare, string);
195  printf("%s ", str6);
196  memberTest(res);
197
198  char* str7 = "butterfly";
199  res = member(strTree, str7, isEqual, compare, string);
200  printf("%s ", str7);
201  memberTest(res);
202
203  /* Test 3 For Person Struct tree */
204  printf("\nTest 3: Person Struct tree\n");
205  struct Person *per1 =  (struct Person *)malloc(sizeof(struct Person));
206  per1->first = "Billy";
207  per1->last = "Joel";
208  struct node *perTree = insert(tree, per1, compare, person);
209
210  struct Person *per2 =  (struct Person *)malloc(sizeof(struct Person));
211  per2->first = "Michael";
212  per2->last = "Jackson";
213  insert(perTree, per2, compare, person);
214
215  struct Person *per3 =  (struct Person *)malloc(sizeof(struct Person));
216  per3->first = "Freddie";
217  per3->last = "Mercury";
218  insert(perTree, per3, compare, person);
219
220  struct Person *per4 =  (struct Person *)malloc(sizeof(struct Person));
221  per4->first = "Elton";
```

```
222   per4->last = "John";
223   insert(perTree, per4, compare, person);
224
225   struct Person *per5 =  (struct Person *)malloc(sizeof(struct Person));
226   per5->first = "Elvis";
227   per5->last = "Presley";
228   insert(perTree, per5, compare, person);
229   printtree(perTree, printVal, person);
230
231   struct Person *per6 =  (struct Person *)malloc(sizeof(struct Person));
232   per6->first = "Elvis";
233   per6->last = "Presley";
234   res = member(perTree, per6, isEqual, compare, person);
235   printf("%s %s ", per6->first, per6->last);
236   memberTest(res);
237
238   struct Person *per7 =  (struct Person *)malloc(sizeof(struct Person));
239   per7->first = "Mick";
240   per7->last = "Jagger";
241   res = member(perTree, per7, isEqual, compare, person);
242   printf("%s %s ", per7->first, per7->last);
243   memberTest(res);
244
245   return 0;
246 }
247
```
