```
/**********************************ASSIGNMENT 8**************************************
Beginning with an empty binary search tree, Construct a binary search tree by inserting
the values in the order given. After constructing a binary tree -
i. Insert new node
ii. Find number of nodes in longest path from root
iii. Minimum data value found in the tree
iv. Change a tree so that the roles of the left and right pointers are swapped at every node
v. Search a value.
*****************************************************************************************/
#include <stdio.h>
#include <stdlib.h>


// Node structure
struct Node {
  int data;
  struct Node* left;
  struct Node* right;
};


// Create a new node
struct Node* createNode(int data) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->left = newNode->right = NULL;
  return newNode;
}


// Insert a node into BST
struct Node* insert(struct Node* root, int data) {
  if (root == NULL)
     return createNode(data);

  if (data < root->data)
     root->left = insert(root->left, data);
  else if (data > root->data)
     root->right = insert(root->right, data);

  return root;
}


// Find height of BST (longest path from root to a leaf)
int findHeight(struct Node* root) {
  if (root == NULL)
     return 0;

  int leftHeight = findHeight(root->left);
  int rightHeight = findHeight(root->right);
```

```
  return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;
}


// Find minimum value in BST
int findMin(struct Node* root) {
  struct Node* current = root;
  while (current && current->left != NULL)
      current = current->left;
  return current->data;
}


// Mirror the tree
void mirror(struct Node* root) {
  if (root == NULL)
      return;

  // Swap left and right
  struct Node* temp = root->left;
  root->left = root->right;
  root->right = temp;


  // Recurse for children
  mirror(root->left);
  mirror(root->right);
}


// Search for a value
int search(struct Node* root, int key) {
  if (root == NULL)
      return 0;
  if (key == root->data)
      return 1;
  else if (key < root->data)
      return search(root->left, key);
  else
      return search(root->right, key);
}


// Inorder traversal for debugging
void inorder(struct Node* root) {
  if (root == NULL)
      return;
  inorder(root->left);
  printf("%d ", root->data);
  inorder(root->right);
}
```

```
int main() {
  struct Node* root = NULL;


  // Initial values
  int values[] = {50, 30, 70, 20, 40, 60, 80};
  int n = sizeof(values) / sizeof(values[0]);


  for (int i = 0; i < n; i++)
     root = insert(root, values[i]);


  // i. Insert new node
  root = insert(root, 65);
  printf("Inorder after insertion: ");
  inorder(root);
  printf("\n");


  // ii. Longest path
  int height = findHeight(root);
  printf("Longest path (height): %d\n", height);


  // iii. Minimum value
  printf("Minimum value in BST: %d\n", findMin(root));


  // iv. Mirror the tree
  mirror(root);
  printf("Inorder after mirroring: ");
  inorder(root);
  printf("\n");


  // v. Search
  int searchValue = 40;
  if (search(root, searchValue))
     printf("%d found in the tree.\n", searchValue);
  else
     printf("%d not found in the tree.\n", searchValue);


  searchValue = 90;
  if (search(root, searchValue))
     printf("%d found in the tree.\n", searchValue);
  else
     printf("%d not found in the tree.\n", searchValue);


  return 0;
}
```

/**OUTPUT***
student@student-OptiPlex-3000:~$ ./ass8
Inorder after insertion: 20 30 40 50 60 65 70 80
Longest path (height): 4
Minimum value in BST: 20
Inorder after mirroring: 80 70 65 60 50 40 30 20
40 not found in the tree.
90 not found in the tree.
***/