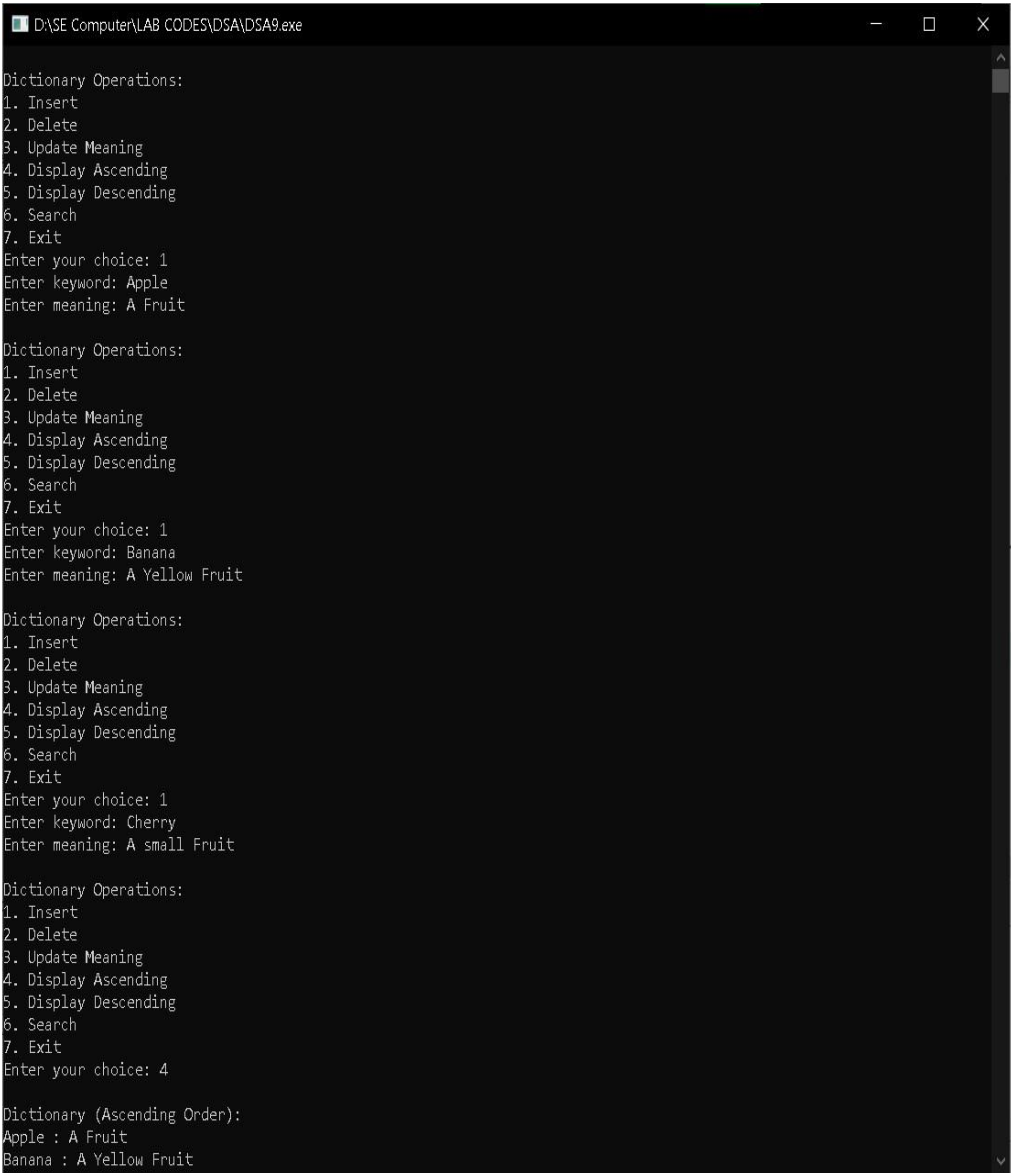
```
#include <iostream>
#include <string>
using namespace std;

struct Node
{
    string keyword, meaning;
    Node* left;
    Node* right;
    int height;
};

// Function to get the height of a node
int getHeight(Node* n)
{
    if (n == NULL)
        return 0;
    else
        return n->height;
}

// Function to get balance factor
int getBalanceFactor(Node* n)
{
    if (n == NULL)
        return 0;
    else
        return getHeight(n->left) - getHeight(n->right);
}

// Function to create a new node
Node* createNode(string keyword, string meaning)
{
    Node* node = new Node();
    node->keyword = keyword;
    node->meaning = meaning;
    node->left = node->right = NULL;
    node->height = 1;
    return node;
}

// Right rotate
Node* rightRotate(Node* y)
{
    Node* x = y->left;
    Node* T2 = x->right;
    x->right = y;
    y->left = T2;

    // Update height of y
    int leftHeight = getHeight(y->left);
    int rightHeight = getHeight(y->right);
    if (leftHeight > rightHeight)
        y->height = leftHeight + 1;
    else
```

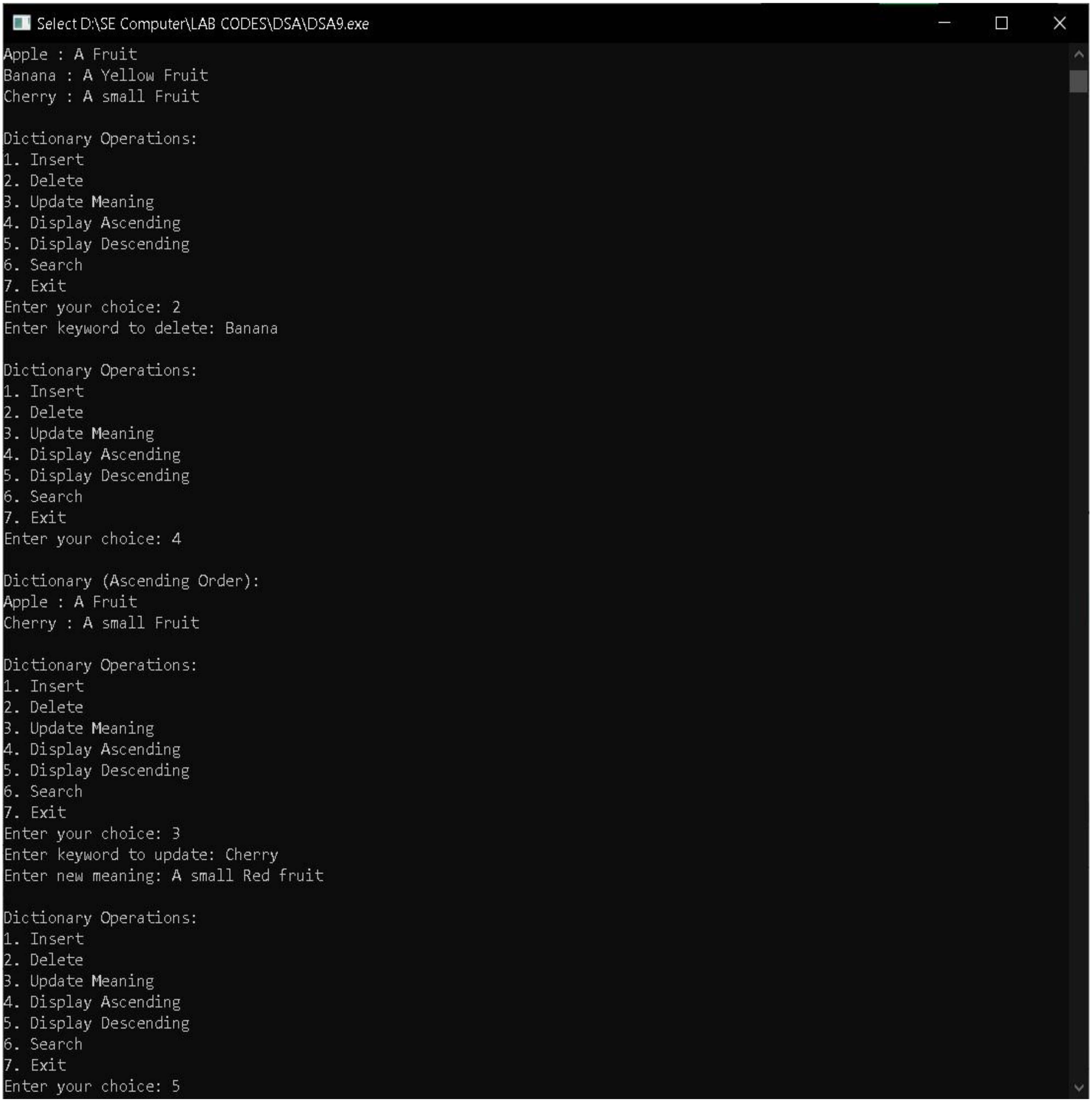
```
        y->height = rightHeight + 1;

    // Update height of x
    leftHeight = getHeight(x->left);
    rightHeight = getHeight(x->right);
    if (leftHeight > rightHeight)
        x->height = leftHeight + 1;
    else
        x->height = rightHeight + 1;

    return x;
}

// Left rotate
Node* leftRotate(Node* x)
{
    Node* y = x->right;
    Node* T2 = y->left;
    y->left = x;
    x->right = T2;

    // Update height of x
    int leftHeight = getHeight(x->left);
    int rightHeight = getHeight(x->right);
    if (leftHeight > rightHeight)
        x->height = leftHeight + 1;
    else
        x->height = rightHeight + 1;

    // Update height of y
    leftHeight = getHeight(y->left);
    rightHeight = getHeight(y->right);
    if (leftHeight > rightHeight)
        y->height = leftHeight + 1;
    else
        y->height = rightHeight + 1;

    return y;
}

// Insert a keyword
Node* insert(Node* root, string keyword, string meaning)
{
    if (root == NULL)
        return createNode(keyword, meaning);

    if (keyword < root->keyword)
        root->left = insert(root->left, keyword, meaning);
    else if (keyword > root->keyword)
        root->right = insert(root->right, keyword, meaning);
    else {
        root->meaning = meaning; // Update meaning if keyword exists
        return root;
    }

    // Update height of root
    int leftHeight = getHeight(root->left);
    int rightHeight = getHeight(root->right);
    if (leftHeight > rightHeight)
        root->height = leftHeight + 1;
    else
        root->height = rightHeight + 1;

    int balance = getBalanceFactor(root);

    // Perform rotations if needed
    if (balance > 1 && keyword < root->left->keyword)
        return rightRotate(root);
    if (balance < -1 && keyword > root->right->keyword)
        return leftRotate(root);
    if (balance > 1 && keyword > root->left->keyword)
        {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    if (balance < -1 && keyword < root->right->keyword)
        {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }

    return root;
}

// Find the node with minimum value
Node* minValueNode(Node* root)
{
    while (root->left)
        root = root->left;
    return root;
}

// Delete a keyword
Node* deleteNode(Node* root, string keyword)
{
    if (root == NULL)
        return root;

    if (keyword < root->keyword)
        root->left = deleteNode(root->left, keyword);
    else if (keyword > root->keyword)
        root->right = deleteNode(root->right, keyword);
    else
        {
        if (!root->left || !root->right)
                {
            Node* temp = root->left ? root->left : root->right;
            if (!temp)
                temp = root, root = NULL;
            else
                *root = *temp;
            delete temp;
        }
                else
                {
            Node* temp = minValueNode(root->right);
            root->keyword = temp->keyword;
            root->meaning = temp->meaning;
            root->right = deleteNode(root->right, temp->keyword);
        }
    }

    if (!root)
        return root;

    // Update height of root
    int leftHeight = getHeight(root->left);
    int rightHeight = getHeight(root->right);
    if (leftHeight > rightHeight)
        root->height = leftHeight + 1;
    else
        root->height = rightHeight + 1;

    int balance = getBalanceFactor(root);

    if (balance > 1 && getBalanceFactor(root->left) >= 0)
        return rightRotate(root);
    if (balance > 1 && getBalanceFactor(root->left) < 0)
        {
        root->left = leftRotate(root->left);
        return rightRotate(root);
    }
    if (balance < -1 && getBalanceFactor(root->right) <= 0)
        return leftRotate(root);
    if (balance < -1 && getBalanceFactor(root->right) > 0)
        {
        root->right = rightRotate(root->right);
        return leftRotate(root);
    }

    return root;
}

// Display dictionary in ascending order
void inOrder(Node* root)
{
    if (root)
        {
        inOrder(root->left);
        cout << root->keyword << " : " << root->meaning << endl;
        inOrder(root->right);
    }
}

// Display dictionary in descending order
void reverseInOrder(Node* root)
{
    if (root)
        {
        reverseInOrder(root->right);
        cout << root->keyword << " : " << root->meaning << endl;
        reverseInOrder(root->left);
    }
}

// Search for a keyword and count comparisons
int search(Node* root, string keyword, int comparisons = 0)
{
    if (root == NULL)
        return comparisons;
    comparisons++;
    if (keyword == root->keyword)
        return comparisons;
    else if (keyword < root->keyword)
        return search(root->left, keyword, comparisons);
    else
        return search(root->right, keyword, comparisons);
}

// Update the meaning of a keyword
void updateMeaning(Node* root, string keyword, string newMeaning)
{
    if (root == NULL)
        {
        cout << "Keyword not found!\n";
        return;
    }
    if (keyword < root->keyword)
        updateMeaning(root->left, keyword, newMeaning);
    else if (keyword > root->keyword)
        updateMeaning(root->right, keyword, newMeaning);
    else
        root->meaning = newMeaning;
}

int main()
{
    Node* root = NULL;
    int choice;
    string keyword, meaning;

    do {
        cout << "\nDictionary Operations:\n";
        cout << "1. Insert\n2. Delete\n3. Update Meaning\n4. Display Ascending\n5. Display Descending\n6. Search\n7. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice)
                {
            case 1:
                cout << "Enter keyword: ";
                cin >> keyword;
                cout << "Enter meaning: ";
                cin.ignore();
                getline(cin, meaning);
                root = insert(root, keyword, meaning);
                break;
```

```cpp

            case 2:
                cout << "Enter keyword to delete: ";
                cin >> keyword;
                root = deleteNode(root, keyword);
                break;

            case 3:
                cout << "Enter keyword to update: ";
                cin >> keyword;
                cout << "Enter new meaning: ";
                cin.ignore();
                getline(cin, meaning);
                updateMeaning(root, keyword, meaning);
                break;

            case 4:
                cout << "\nDictionary (Ascending Order):\n";
                inOrder(root);
                break;

            case 5:
                cout << "\nDictionary (Descending Order):\n";
                reverseInOrder(root);
                break;

            case 6:
                cout << "Enter keyword to search: ";
                cin >> keyword;
                cout << "Comparisons required: " << search(root, keyword) << endl;
                break;

            case 7:
                cout << "Exiting...\n";
                break;

            default:
                cout << "Invalid choice!\n";
        }
    }
        while (choice != 7);

    return 0;
}

```

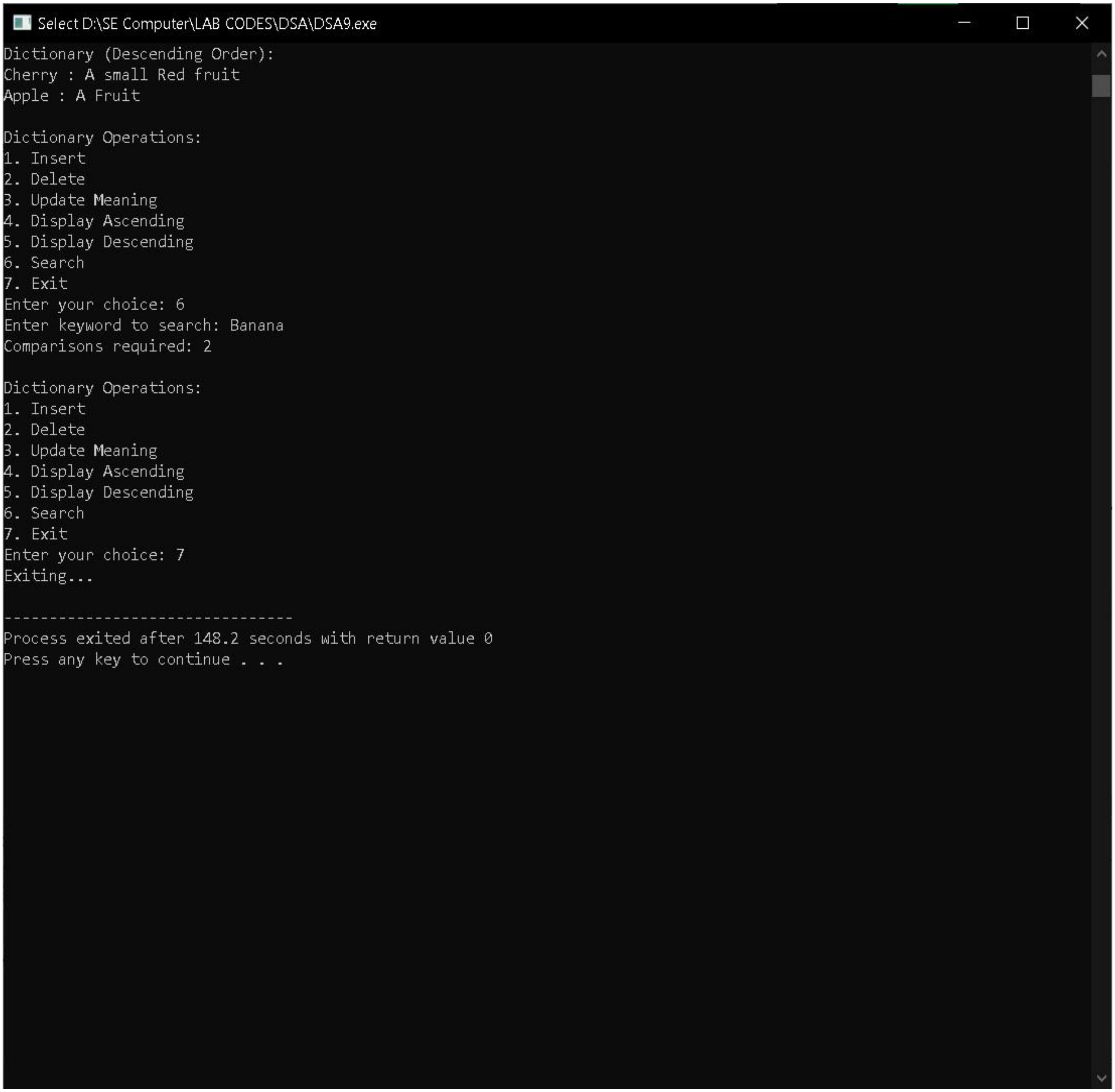
OutPut :



```
Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 1
Enter keyword: Apple
Enter meaning: A Fruit

Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 1
Enter keyword: Banana
Enter meaning: A Yellow Fruit

Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 1
Enter keyword: Cherry
Enter meaning: A small Fruit

Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 4

Dictionary (Ascending Order):
Apple : A Fruit
Banana : A Yellow Fruit
```



```
Apple : A Fruit
Banana : A Yellow Fruit
Cherry : A small Fruit

Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 2
Enter keyword to delete: Banana

Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 4

Dictionary (Ascending Order):
Apple : A Fruit
Cherry : A small Fruit

Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 3
Enter keyword to update: Cherry
Enter new meaning: A small Red fruit

Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 5
```



```
Dictionary (Descending Order):
Cherry : A small Red fruit
Apple : A Fruit

Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 6
Enter keyword to search: Banana
Comparisons required: 2

Dictionary Operations:
1. Insert
2. Delete
3. Update Meaning
4. Display Ascending
5. Display Descending
6. Search
7. Exit
Enter your choice: 7
Exiting...

--------------------------------
Process exited after 148.2 seconds with return value 0
Press any key to continue . . .
```