FINAL ASSIGNMENT

DATA STRUCTURE LAB

NAME:AYESHA IMRAN
ROLL NO:2023-BSAI-011
SUBMITTED TO:MAM IRSHA QURESHI

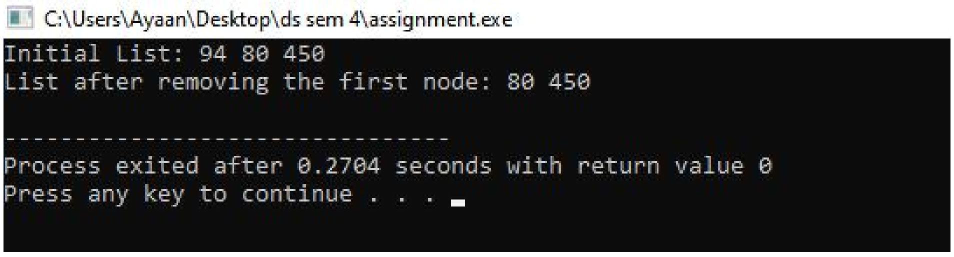
DOUBLE LINK LIST

1. Write a program to delete the first node in a doubly linked list.

Explanation:
To remove the first node in a doubly linked list, we update the start pointer to refer to the second node. If the new head exists, its prev pointer is set to nullptr. Finally, the original first node is deleted.

```
#include <iostream>
using namespace std;
struct DNode {
   int value;
   DNode* next;
   DNode* previous;
};
void removeFirst(DNode*& start) {
   if (start == nullptr) {
      cout << "The list is empty; no nodes to remove.\n";
      return;
   }
   DNode* tempNode = start;
   start = start->next;
   if (start != nullptr)
      start->previous = nullptr;
   delete tempNode;
}
void displayList(DNode* start) {
   DNode* current = start;
   while (current != nullptr) {
      cout << current->value << " ";
      current = current->next;
   }
   cout << endl;
}
```

```
int main() {
    DNode* head = new DNode{94, nullptr, nullptr};
    DNode* middle = new DNode{80, nullptr, head};
    DNode* tail = new DNode{450, nullptr, middle};
    head->next = middle;
    middle->next = tail;

    cout << "Initial List: ";
    displayList(head);

    removeFirst(head);
    cout << "List after removing the first node: ";
    displayList(head);

    return 0;
}
```



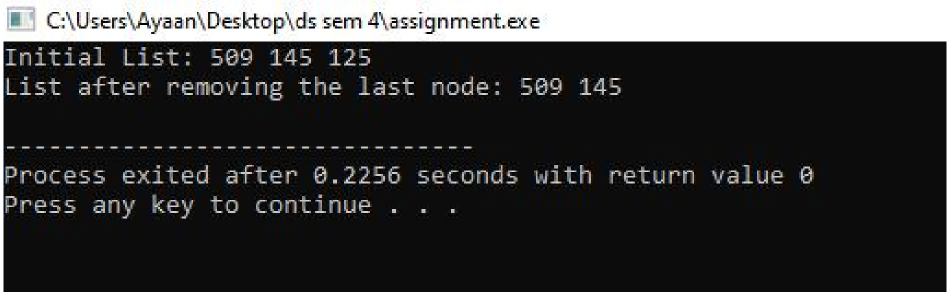
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

```
Initial List: 94 80 450
List after removing the first node: 80 450

--------------------------------
Process exited after 0.2704 seconds with return value 0
Press any key to continue . . .
```

2.How can you delete the last node in a doubly linked list? Write the code.

Explanation:
To delete the last node, traverse the list to the last node, adjust the next pointer of the second-last node to nullptr, and delete the last node.

```
#include <iostream>
using namespace std;
struct DNode {
    int value;
    DNode* next;
    DNode* previous;
};
void removeLast(DNode*& start) {
    if (start == nullptr) {
        cout << "The list is empty; no nodes to remove.\n";
        return;
    }
    if (start->next == nullptr) { // Single node case
        delete start;
        start = nullptr;
        return;
    }
    DNode* temp = start;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->previous->next = nullptr;
    delete temp;
}
void displayList(DNode* start) {
    DNode* current = start;
    while (current != nullptr) {
        cout << current->value << " ";
        current = current->next;
    }
    cout << endl;
}
int main() {
    DNode* head = new DNode{509, nullptr, nullptr};
    DNode* middle = new DNode{145, nullptr, head};
    DNode* tail = new DNode{125, nullptr, middle};
    head->next = middle;
    middle->next = tail;

    cout << "Initial List: ";
    displayList(head);

    removeLast(head);
    cout << "List after removing the last node: ";
    displayList(head);

    return 0;
}
```



C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

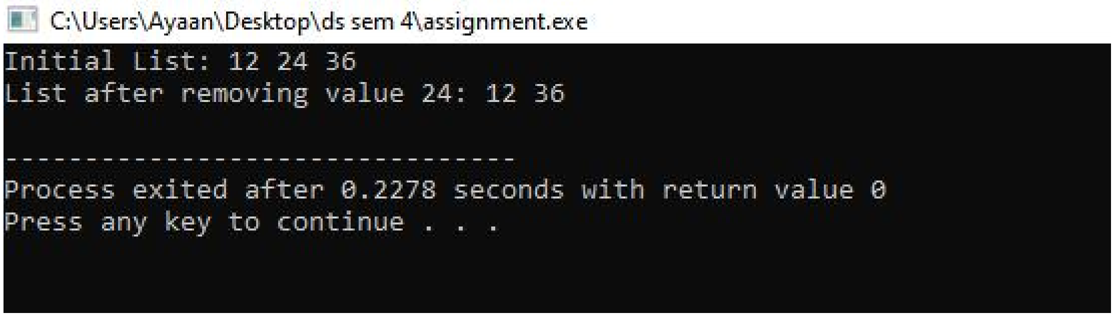
```
Initial List: 509 145 125
List after removing the last node: 509 145

--------------------------------
Process exited after 0.2256 seconds with return value 0
Press any key to continue . . .
```

3. Write code to delete a node by its value in a doubly linked list.

Explanation:
Find the node with the given value by traversing the list. Adjust pointers of its neighboring nodes to exclude it, and then delete the node.

```
#include <iostream>
```

```
using namespace std;
struct DNode {
    int value;
    DNode* next;
    DNode* previous;
};
void removeByValue(DNode*& start, int target) {
    if (start == nullptr) {
        cout << "The list is empty; no nodes to remove.\n";
        return;
    }
    DNode* temp = start;
    while (temp != nullptr && temp->value != target) {
        temp = temp->next;
    }
    if (temp == nullptr) {
        cout << "Value not found in the list.\n";
        return;
    }
    if (temp->previous != nullptr)
        temp->previous->next = temp->next;
    else
        start = temp->next; // Removing the first node
    if (temp->next != nullptr)
        temp->next->previous = temp->previous;
    delete temp;
}
void displayList(DNode* start) {
    DNode* current = start;
    while (current != nullptr) {
        cout << current->value << " ";
        current = current->next;
    }
    cout << endl;
}
int main() {
    DNode* head = new DNode{12, nullptr, nullptr};
    DNode* middle = new DNode{24, nullptr, head};
    DNode* tail = new DNode{36, nullptr, middle};
    head->next = middle;
    middle->next = tail;

    cout << "Initial List: ";
    displayList(head);

    removeByValue(head, 24);
    cout << "List after removing value 24: ";
    displayList(head);

    return 0;
}
```


C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

```
Initial List: 12 24 36
List after removing value 24: 12 36

--------------------------------
Process exited after 0.2278 seconds with return value 0
Press any key to continue . . .
```

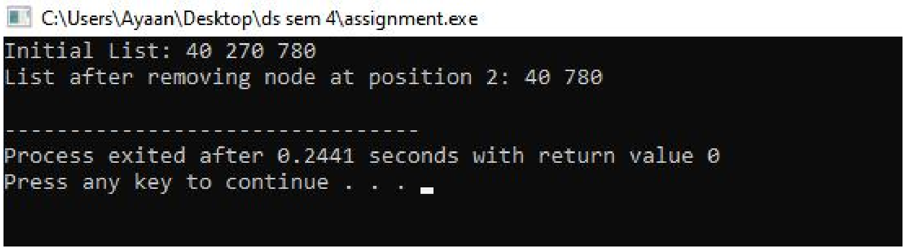
---

4. How would you delete a node at a specific position in a doubly linked list? Show it in code.

Explanation:
To remove a node at a particular position, traverse to that position, adjust the pointers of adjacent nodes, and delete the target node.

```
#include <iostream>
using namespace std;
struct DNode {
   int value;
   DNode* next;
   DNode* previous;
};
void removeAtPosition(DNode*& start, int position) {
   if (start == nullptr) {
      cout << "The list is empty; no nodes to remove.\n";
      return;
   }
   if (position == 1) { // Special case: first node
      DNode* temp = start;
      start = start->next;
      if (start != nullptr)
         start->previous = nullptr;
      delete temp;
      return;
   }
   DNode* temp = start;
   for (int i = 1; temp != nullptr && i < position; i++) {
      temp = temp->next;
   }
   if (temp == nullptr) {
      cout << "Position out of range.\n";
      return;
   }
   if (temp->previous != nullptr)
```

```
        temp->previous->next = temp->next;
    if (temp->next != nullptr)
        temp->next->previous = temp->previous;
    delete temp;
}
void displayList(DNode* start) {
    DNode* current = start;
    while (current != nullptr) {
        cout << current->value << " ";
        current = current->next;
    }
    cout << endl;
}
int main() {
    DNode* head = new DNode{40, nullptr, nullptr};
    DNode* second = new DNode{270, nullptr, head};
    DNode* third = new DNode{780, nullptr, second};
    head->next = second;
    second->next = third;

    cout << "Initial List: ";
    displayList(head);

    removeAtPosition(head, 2);
    cout << "List after removing node at position 2: ";
    displayList(head);

    return 0;
}
```



```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
Initial List: 40 270 780
List after removing node at position 2: 40 780

--------------------------------
Process exited after 0.2441 seconds with return value 0
Press any key to continue . . .
```

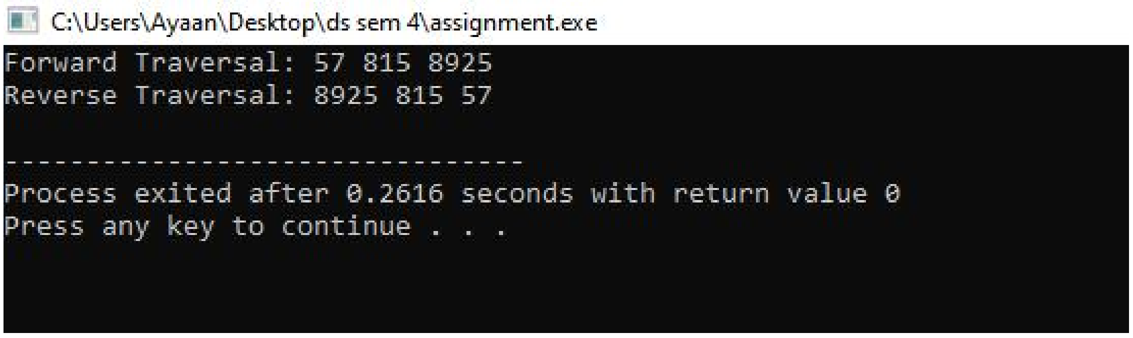
---

5. After deleting a node, how will you write the forward and reverse traversal functions?

Explanation:
Forward traversal starts from the head and uses the next pointers to visit nodes until the end. Reverse traversal begins from the last node by first traversing the list to the tail and then using the previous pointers to go back to the start.

```
#include <iostream>
```

```
using namespace std;
struct DNode {
    int value;
    DNode* next;
    DNode* previous;
};
void displayForward(DNode* start) {
    if (start == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    DNode* current = start;
    while (current != nullptr) {
        cout << current->value << " ";
        current = current->next;
    }
    cout << endl;
}
void displayReverse(DNode* start) {
    if (start == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    DNode* current = start;
    while (current->next != nullptr) {
        current = current->next; // Move to the last node
    }
    while (current != nullptr) {
        cout << current->value << " ";
        current = current->previous;
    }
    cout << endl;
}
int main() {
    DNode* head = new DNode{57, nullptr, nullptr};
    DNode* middle = new DNode{815, nullptr, head};
    DNode* tail = new DNode{8925, nullptr, middle};
    head->next = middle;
    middle->next = tail;

    cout << "Forward Traversal: ";
    displayForward(head);

    cout << "Reverse Traversal: ";
    displayReverse(head);

    return 0;
}
```



C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

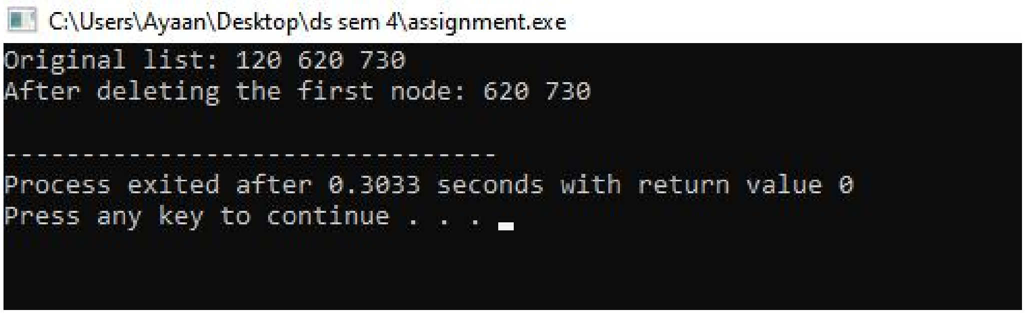
```
Forward Traversal: 57 815 8925
Reverse Traversal: 8925 815 57

--------------------------------
Process exited after 0.2616 seconds with return value 0
Press any key to continue . . .
```

CIRCULAR LINK LIST

1. Write a program to delete the first node in a circular linked list.

Explanation:
To remove the first node, locate the last node to adjust its next pointer to the new head. If the list has only one node, set the head to nullptr.

```
#include <iostream>
using namespace std;
struct CNode {
    int value;
    CNode* next;
};
void deleteFirstNode(CNode*& head) {
    if (head == nullptr) {
        cout << "The circular linked list is empty.\n";
        return;
    }
    if (head->next == head) { // Single node case
        delete head;
        head = nullptr;
        return;
    }
    CNode* temp = head;
    CNode* tail = head;
    while (tail->next != head) {
        tail = tail->next;
    }
    head = head->next; // Move head to the next node
    tail->next = head;
    delete temp;
}
void displayCircularList(CNode* head) {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    CNode* current = head;
    do {
        cout << current->value << " ";
        current = current->next;
    } while (current != head);
    cout << endl;
}
int main() {
    CNode* head = new CNode{120, nullptr};
    CNode* second = new CNode{620, nullptr};
    CNode* third = new CNode{730, nullptr};
    head->next = second;
    second->next = third;
    third->next = head;

    cout << "Original list: ";
    displayCircularList(head);

    deleteFirstNode(head);
    cout << "After deleting the first node: ";
    displayCircularList(head);

    return 0;
}
```


C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

```
Original list: 120 620 730
After deleting the first node: 620 730

--------------------------------
Process exited after 0.3033 seconds with return value 0
Press any key to continue . . .
```

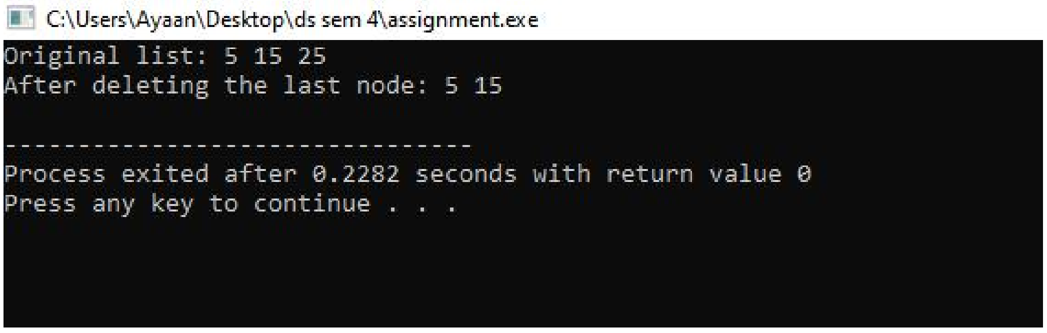
---

2. <u>How can you delete the last node in a circular linked list? Write the code.</u>

Explanation:
To delete the last node, traverse the list to find the second last node and set its next pointer to point to the head. Remove the last node.

```
#include <iostream>
```

```
using namespace std;
struct CNode {
    int value;
    CNode* next;
};
void deleteLastNode(CNode*& head) {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    if (head->next == head) { // Single node case
        delete head;
        head = nullptr;
        return;
    }
    CNode* current = head;
    CNode* prev = nullptr;
    while (current->next != head) {
        prev = current;
        current = current->next;
    }
    prev->next = head; // Update second last node's pointer
    delete current;
}
void displayCircularList(CNode* head) {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    CNode* current = head;
    do {
        cout << current->value << " ";
        current = current->next;
    } while (current != head);
    cout << endl;
}
int main() {
    CNode* head = new CNode{5, nullptr};
    CNode* node2 = new CNode{15, nullptr};
    CNode* node3 = new CNode{25, nullptr};
    head->next = node2;
    node2->next = node3;
    node3->next = head;

    cout << "Original list: ";
    displayCircularList(head);

    deleteLastNode(head);
    cout << "After deleting the last node: ";
    displayCircularList(head);
    return 0;
}
```



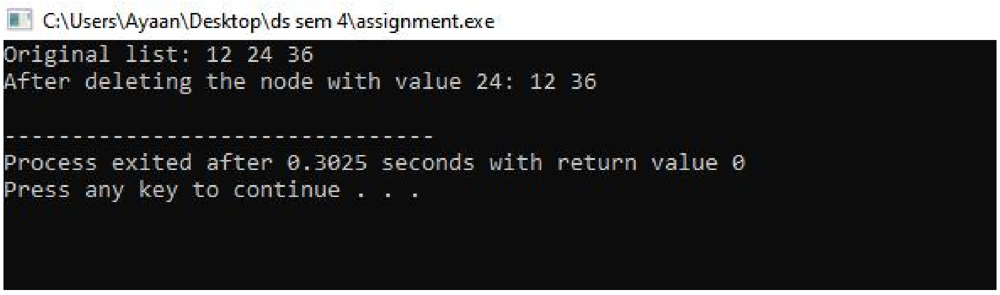
```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
Original list: 5 15 25
After deleting the last node: 5 15

--------------------------------
Process exited after 0.2282 seconds with return value 0
Press any key to continue . . .
```

---

3. Write a function to delete a node by its value in a circular linked list

Explanation:
Search the list for the target value. If found, adjust the previous node’s next pointer to bypass the node to be deleted.

```
#include <iostream>
using namespace std;
struct CNode {
    int value;
    CNode* next;
};
void deleteByValue(CNode*& head, int key) {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    if (head->value == key && head->next == head) { // Single node
        delete head;
        head = nullptr;
        return;
    }
    CNode* current = head;
    CNode* prev = nullptr;
    do {
        if (current->value == key) {
            if (prev == nullptr) { // Deleting the head node
                CNode* last = head;
                while (last->next != head) {
                    last = last->next;
                }
                head = head->next;
                last->next = head;
                delete current;
            } else {
                prev->next = current->next;
                delete current;
            }
            return;
        }
        prev = current;
        current = current->next;
    } while (current != head);
    cout << "Value not found in the list.\n";
}
void displayCircularList(CNode* head) {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    CNode* current = head;
    do {
        cout << current->value << " ";
        current = current->next;
    } while (current != head);
    cout << endl;
}
int main() {
    CNode* head = new CNode{12, nullptr};
    CNode* node2 = new CNode{24, nullptr};
    CNode* node3 = new CNode{36, nullptr};
    head->next = node2;
    node2->next = node3;
    node3->next = head;

    cout << "Original list: ";
    displayCircularList(head);

    deleteByValue(head, 24);
    cout << "After deleting the node with value 24: ";
    displayCircularList(head);

    return 0;
}
```



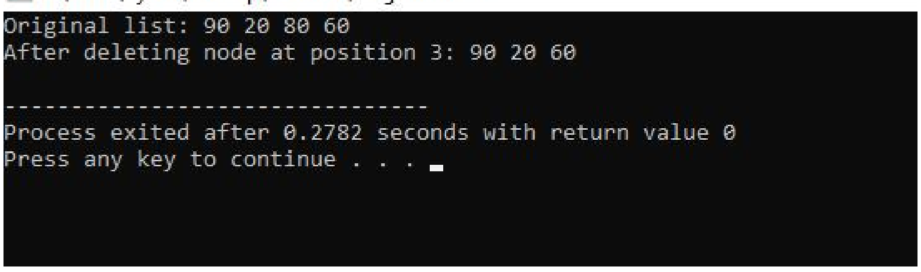
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

```
Original list: 12 24 36
After deleting the node with value 24: 12 36

--------------------------------
Process exited after 0.3025 seconds with return value 0
Press any key to continue . . .
```

4. How will you delete a node at a specific position in a circular linked list? Write code for it.

Explanation:
To delete a node at a specific position, traverse the list up to the node just before the position. Adjust pointers to bypass the node to be deleted.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};
void deleteAtPosition(Node*& head, int position) {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }

    if (position == 1) { // Delete the first node
        if (head->next == head) { // Single node case
            delete head;
            head = nullptr;
        } else {
            Node* temp = head;
            Node* last = head;
            while (last->next != head) {
                last = last->next;
            }
            head = head->next;
            last->next = head;
            delete temp;
        }
        return;
    }

    Node* current = head;
    Node* prev = nullptr;
    int count = 1;

    do {
        if (count == position) {
            prev->next = current->next;
            delete current;
            return;
        }
        prev = current;
        current = current->next;
        count++;
    } while (current != head);

    cout << "Position out of bounds.\n";
}
void display(Node* head) {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}
int main() {
    Node* head = new Node{90, nullptr};
    Node* second = new Node{20, nullptr};
    Node* third = new Node{80, nullptr};
    Node* fourth = new Node{60, nullptr};
    head->next = second;
    second->next = third;
    third->next = fourth;
    fourth->next = head;

    cout << "Original list: ";
    display(head);

    deleteAtPosition(head, 3);
    cout << "After deleting node at position 3: ";
    display(head);

    return 0;
}
```



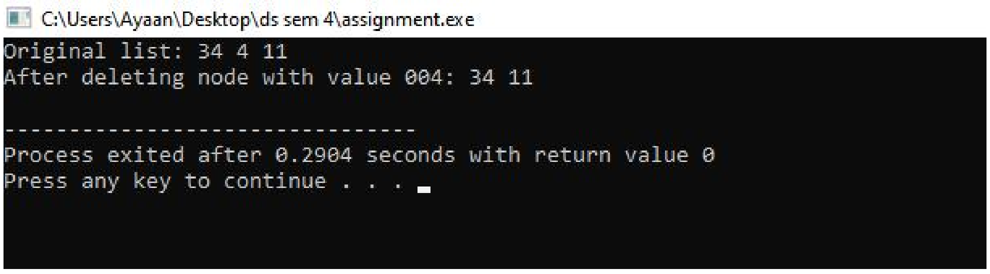
```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
Original list: 90 20 80 60
After deleting node at position 3: 90 20 60

--------------------------------
Process exited after 0.2782 seconds with return value 0
Press any key to continue . . .
```

---

5. Write a program to show forward traversal after deleting a node in a circular linked list.

Explanation:
The task combines node deletion with forward traversal. Implement deletion logic first, then use a loop to print the elements starting from the head.

```
#include <iostream>
using namespace std;
struct Node {
    int value;
    Node* next;
};
void deleteNode(Node*& head, int key) {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }

    if (head->value == key && head->next == head) { // Single node case
        delete head;
        head = nullptr;
        return;
    }

    Node* current = head;
    Node* prev = nullptr;
    do {
        if (current->value == key) {
            if (current == head) { // Deleting the head node
                Node* last = head;
                while (last->next != head) {
                    last = last->next;
                }
                head = head->next;
                last->next = head;
            } else {
                prev->next = current->next;
            }
            delete current;
            return;
        }
        prev = current;
        current = current->next;
    } while (current != head);

    cout << "Value not found in the list.\n";
}
void traverse(Node* head) {
    if (head == nullptr) {
        cout << "The list is empty.\n";
        return;
    }
    Node* current = head;
    do {
        cout << current->value << " ";
        current = current->next;
    } while (current != head);
    cout << endl;
}
int main() {
    Node* head = new Node{34, nullptr};
    Node* second = new Node{004, nullptr};
    Node* third = new Node{11, nullptr};
    head->next = second;
    second->next = third;
    third->next = head;

    cout << "Original list: ";
    traverse(head);

    deleteNode(head, 004);
    cout << "After deleting node with value 004: ";
    traverse(head);

    return 0;
}
```


```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
Original list: 34 4 11
After deleting node with value 004: 34 11

--------------------------------
Process exited after 0.2904 seconds with return value 0
Press any key to continue . . .
```

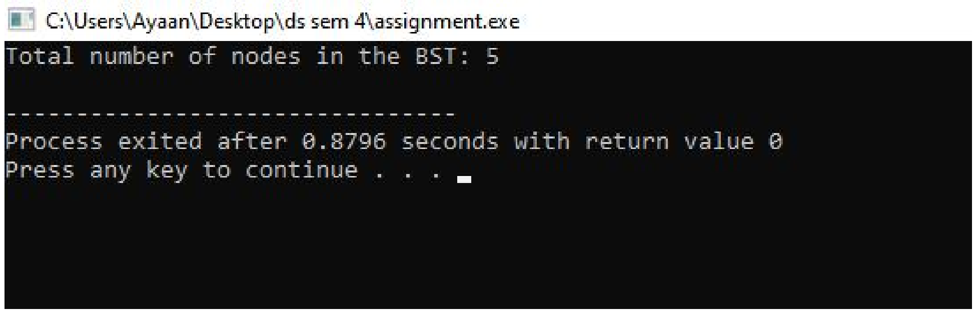
BINARY SEARCH TREE

1. Write a program to count all the nodes in a binary search tree.

Explanation:
To count the nodes in a Binary Search Tree (BST), we use a recursive function. The function traverses the entire tree, returning 1 for each visited node. The base case is when the node is nullptr, returning 0.

```
#include <iostream>
```

```
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
};
Node* createNode(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->left = newNode->right = nullptr;
    return newNode;
}
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
int countNodes(Node* root) {
    if (root == nullptr) {
        return 0;
    }
    return 1 + countNodes(root->left) + countNodes(root->right);
}
int main() {
    Node* root = nullptr;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);

    cout << "Total number of nodes in the BST: " << countNodes(root) << endl;
    return 0;
}
```

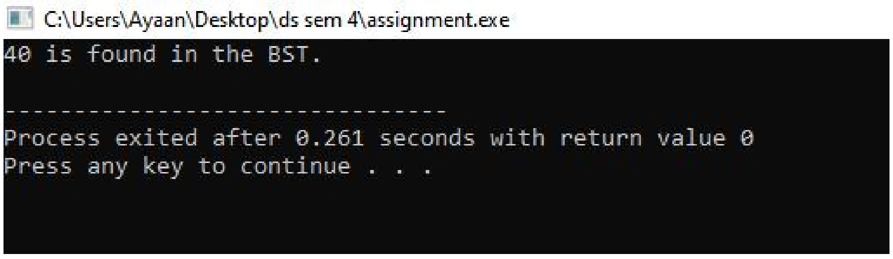

C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

```
Total number of nodes in the BST: 5

--------------------------------
Process exited after 0.8796 seconds with return value 0
Press any key to continue . . .
```

2. How can you search for a specific value in a binary search tree? Write the code.

Explanation:
To search for a specific value in a BST, recursively compare the key with the current node's value. If it matches, return true. If it's smaller, search the left subtree; if larger, search the right subtree.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
};
Node* createNode(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->left = newNode->right = nullptr;
    return newNode;
}
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
bool search(Node* root, int key) {
    if (root == nullptr) {
        return false;
    }
    if (root->data == key) {
        return true;
    } else if (key < root->data) {
        return search(root->left, key);
    } else {
        return search(root->right, key);
    }
}
int main() {
    Node* root = nullptr;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);

    int key = 40;
    if (search(root, key)) {
        cout << key << " is found in the BST." << endl;
    } else {
        cout << key << " is not found in the BST." << endl;
    }
    return 0;
}
```



```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
40 is found in the BST.

--------------------------------
Process exited after 0.261 seconds with return value 0
Press any key to continue . . .
```

---

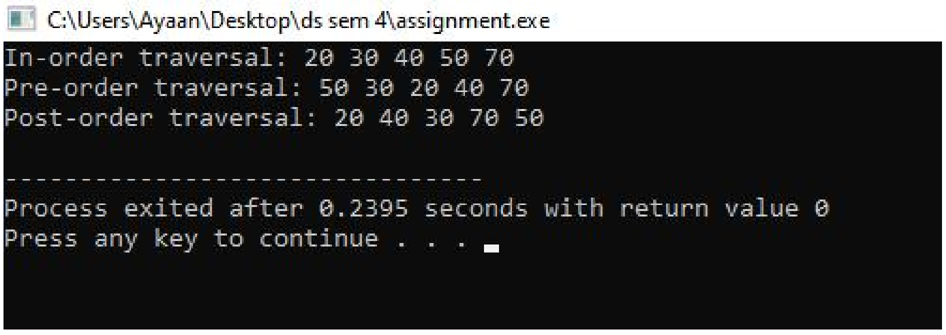
3. Write code to traverse a binary search tree in in-order, pre-order, and post order.

Explanation:

- **In-order:** Traverse the left subtree, visit the root, and then traverse the right subtree.
- **Pre-order:** Visit the root, then traverse the left subtree and right subtree.
- **Post-order:** Traverse the left subtree, then the right subtree, and finally visit the root.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
};
Node* createNode(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->left = newNode->right = nullptr;
    return newNode;
}
```

```
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
void inOrder(Node* root) {
    if (root != nullptr) {
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
void preOrder(Node* root) {
    if (root != nullptr) {
        cout << root->data << " ";
        preOrder(root->left);
        preOrder(root->right);
    }
}
void postOrder(Node* root) {
    if (root != nullptr) {
        postOrder(root->left);
        postOrder(root->right);
        cout << root->data << " ";
    }
}
int main() {
    Node* root = nullptr;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);

    cout << "In-order traversal: ";
    inOrder(root);
    cout << endl;

    cout << "Pre-order traversal: ";
    preOrder(root);
    cout << endl;

    cout << "Post-order traversal: ";
    postOrder(root);
    cout << endl;

    return 0;
}
```

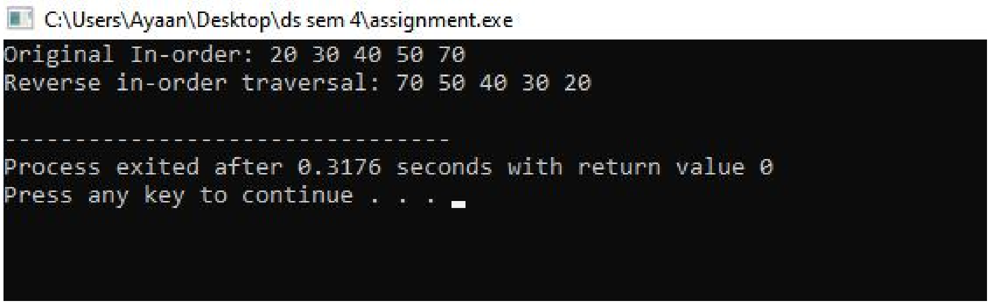


```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
In-order traversal: 20 30 40 50 70
Pre-order traversal: 50 30 20 40 70
Post-order traversal: 20 40 30 70 50

--------------------------------
Process exited after 0.2395 seconds with return value 0
Press any key to continue . . .
```

4.How will you write reverse in-order traversal for a binary search tree? Show it in code

Explanation:
Reverse in-order traversal is similar to in-order traversal, but the order of visiting nodes is reversed. We first traverse the right subtree, visit the root, and then traverse the left subtree.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
};
Node* createNode(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->left = newNode->right = nullptr;
    return newNode;
}
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
void inOrder(Node* root) {
    if (root != nullptr) {
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
void reverseInOrder(Node* root) {
    if (root != nullptr) {
        reverseInOrder(root->right);
        cout << root->data << " ";
        reverseInOrder(root->left);
    }
}
int main() {
    Node* root = nullptr;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);

    cout << "Original In-order: ";
    inOrder(root);
    cout << endl;

    cout << "Reverse in-order traversal: ";
    reverseInOrder(root);
    cout << endl;

    return 0;
}
```



```
C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe
Original In-order: 20 30 40 50 70
Reverse in-order traversal: 70 50 40 30 20

--------------------------------
Process exited after 0.3176 seconds with return value 0
Press any key to continue . . .
```

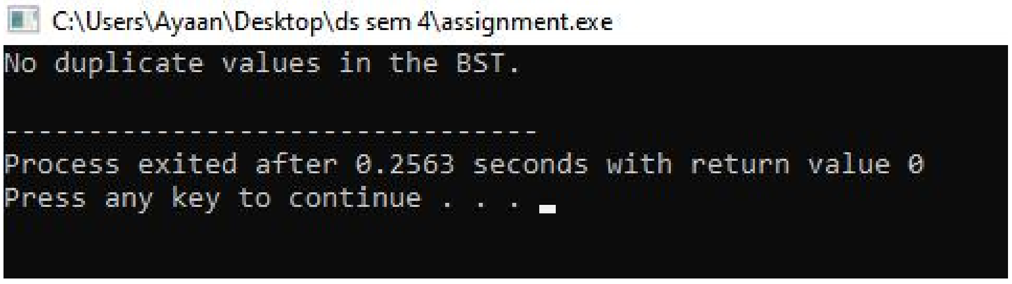
---

5. Write a program to check if there are duplicate values in a binary search tree.

Explanation:
While inserting nodes into the BST, we check if a duplicate value is being inserted. If a duplicate is found, we return true. If the value is less than the

current node’s value, we recursively insert it into the left subtree; otherwise, it goes into the right subtree.

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* left;
   Node* right;
};
Node* createNode(int value) {
   Node* newNode = new Node;
   newNode->data = value;
   newNode->left = newNode->right = nullptr;
   return newNode;
}
bool insertAndCheckDuplicate(Node*& root, int value) {
   if (root == nullptr) {
      root = createNode(value);
      return false;
   }
   if (value == root->data) {
      return true; // Duplicate found
   } else if (value < root->data) {
      return insertAndCheckDuplicate(root->left, value);
   } else {
      return insertAndCheckDuplicate(root->right, value);
   }
}
int main() {
   Node* root = nullptr;
   int values[] = {55, 30, 80, 20, 45, 60, 90, 39};
   bool duplicate = false;

   for (int value : values) {
      if (insertAndCheckDuplicate(root, value)) {
         duplicate = true;
         break;
      }
   }

   if (duplicate) {
      cout << "Duplicate values found in the BST." << endl;
   } else {
      cout << "No duplicate values in the BST." << endl;
   }

   return 0;
}
```


C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

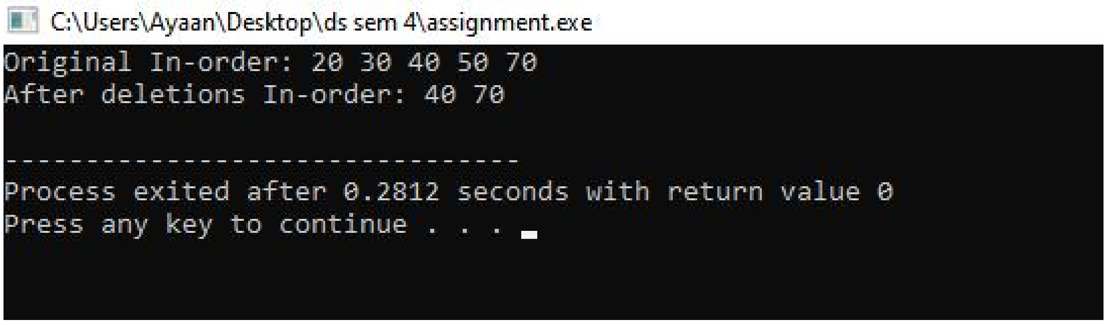
```
No duplicate values in the BST.

--------------------------------
Process exited after 0.2563 seconds with return value 0
Press any key to continue . . .
```

6. How can you delete a node from a binary search tree? Write code for deleting a leaf, a node with one child, and a node with two children.

Explanation:
There are three cases to handle when deleting a node in a BST:

- **Case 1:** Deleting a leaf node (no children).
- **Case 2:** Deleting a node with one child.
- **Case 3:** Deleting a node with two children, which requires finding the in-order successor and replacing the node with that value.

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* left;
    Node* right;
};
Node* createNode(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->left = newNode->right = nullptr;
    return newNode;
}
Node* insert(Node* root, int value) {
    if (root == nullptr) {
        return createNode(value);
    }
    if (value < root->data) {
        root->left = insert(root->left, value);
    } else if (value > root->data) {
        root->right = insert(root->right, value);
    }
    return root;
}
Node* findMin(Node* root) {
    while (root->left != nullptr) {
        root = root->left;
    }
    return root;
}
Node* deleteNode(Node* root, int value) {
    if (root == nullptr) {
        return root;
    }

    if (value < root->data) {
        root->left = deleteNode(root->left, value);
    } else if (value > root->data) {
        root->right = deleteNode(root->right, value);
    } else {
        // Node with one child or no child
        if (root->left == nullptr) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (root->right == nullptr) {
            Node* temp = root->left;
            delete root;
            return temp;
        }

        // Node with two children: Get the inorder successor
        Node* temp = findMin(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }

    return root;
}
void inOrder(Node* root) {
    if (root != nullptr) {
        inOrder(root->left);
        cout << root->data << " ";
        inOrder(root->right);
    }
}
int main() {
    Node* root = nullptr;
    root = insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);

    cout << "Original In-order: ";
    inOrder(root);
    cout << endl;
    root = deleteNode(root, 20); // Leaf node
    root = deleteNode(root, 30); // Node with one child
    root = deleteNode(root, 50); // Node with two children

    cout << "After deletions In-order: ";
    inOrder(root);
    cout << endl;

    return 0;
}
```



C:\Users\Ayaan\Desktop\ds sem 4\assignment.exe

```
Original In-order: 20 30 40 50 70
After deletions In-order: 40 70

--------------------------------
Process exited after 0.2812 seconds with return value 0
Press any key to continue . . .
```