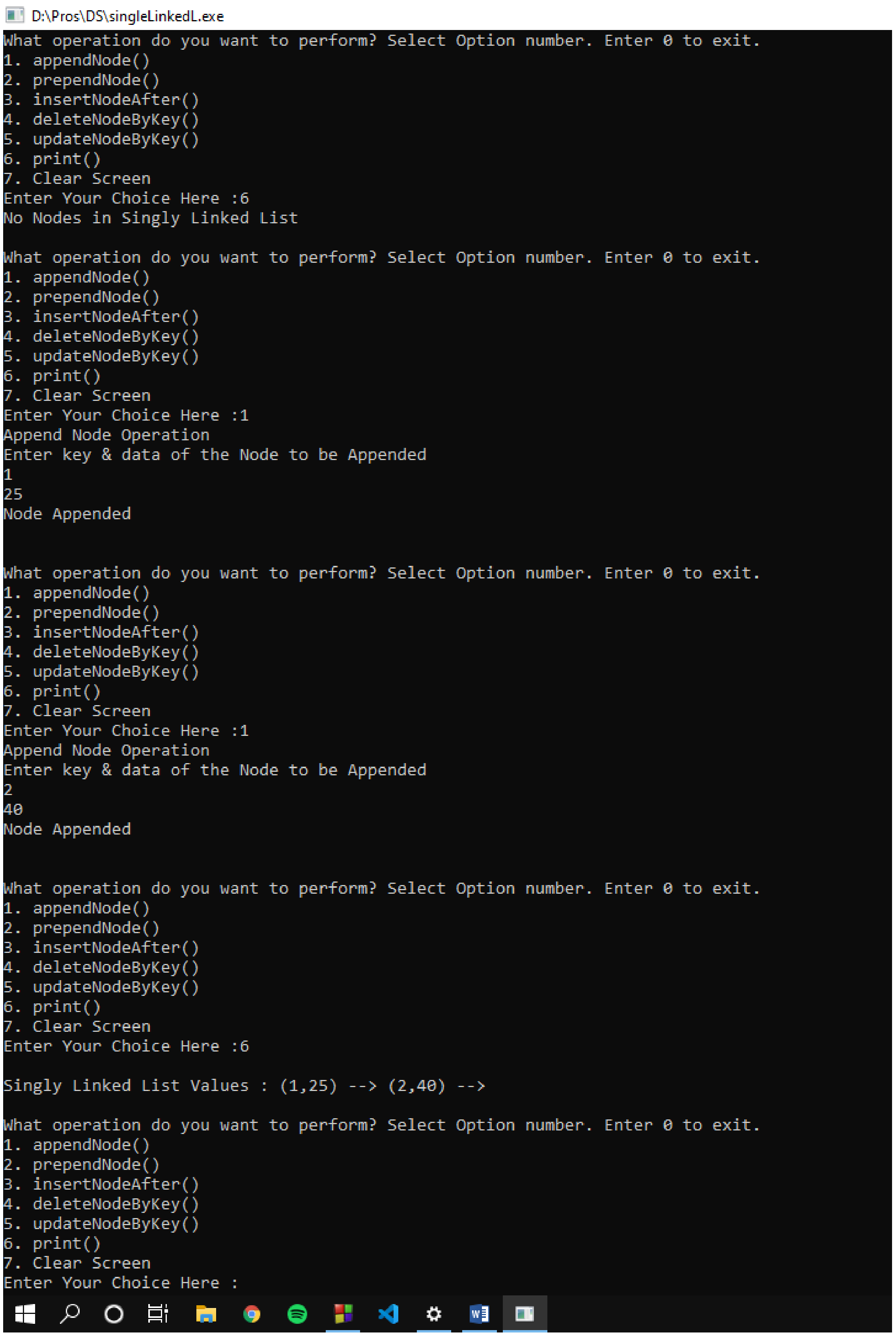
Single Linked List

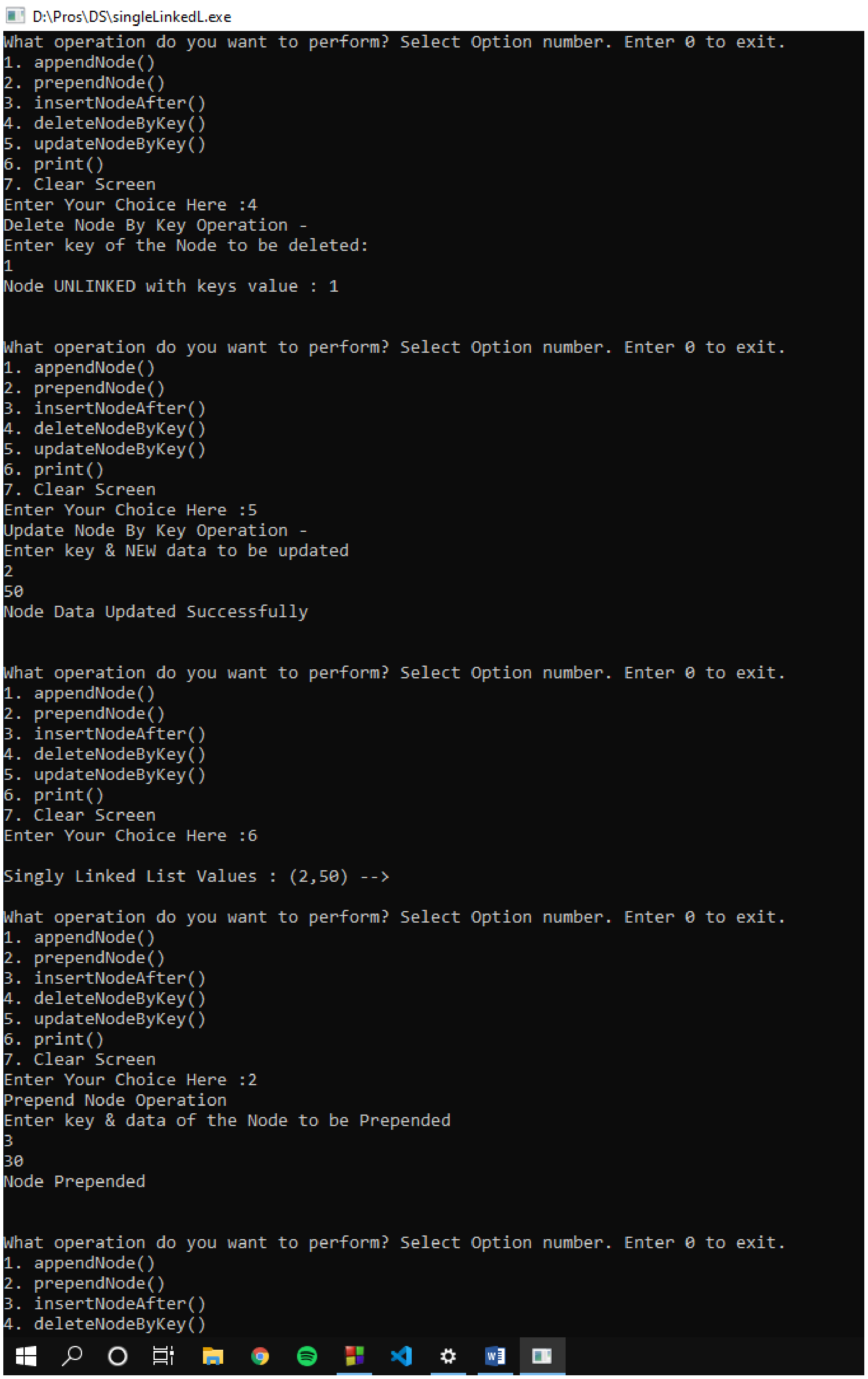
```
#include<iostream>
using namespace std;

class Node {
 public:
  int key;
 int data;
 Node * next;

 Node() {
  key = 0;
  data = 0;
  next = NULL;
 }
 Node(int k, int d) {
  key = k;
  data = d;
 }
};

class SinglyLinkedList {
 public:
  Node * head;

 SinglyLinkedList() {
  head = NULL;
 }
 SinglyLinkedList(Node * n) {
  head = n;
 }

 // 1. CHeck if node exists using key value
 Node * nodeExists(int k) {
  Node * temp = NULL;

  Node * ptr = head;
  while (ptr != NULL) {
   if (ptr - > key == k) {
    temp = ptr;
   }
   ptr = ptr - > next;
  }
  return temp;
 }

 // 2. Append a node to the list
 void appendNode(Node * n) {
  if (nodeExists(n - > key) != NULL) {
   cout << "Node Already exists with key value : " << n - > key << ". Append another node with different Key 
value" << endl;
  } else {
   if (head == NULL) {
    head = n;
    cout << "Node Appended" << endl;
   } else {
    Node * ptr = head;
    while (ptr - > next != NULL) {
     ptr = ptr - > next;
    }
    ptr - > next = n;
    cout << "Node Appended" << endl;
   }
  }

 }
 // 3. Prepend Node - Attach a node at the start
 void prependNode(Node * n) {
  if (nodeExists(n - > key) != NULL) {
   cout << "Node Already exists with key value : " << n - > key << ". Append another node with different Key 
value" << endl;
  } else {
   n - > next = head;
   head = n;
   cout << "Node Prepended" << endl;
  }
 }

 // 4. Insert a Node after a particular node in the list
 void insertNodeAfter(int k, Node * n) {
  Node * ptr = nodeExists(k);
  if (ptr == NULL) {
   cout << "No node exists with key value: " << k << endl;
  } else {
    if (nodeExists(n - > key) != NULL) {
      cout << "Node Already exists with key value : " << n - > key << ". Append another node with different Key
value" << endl;
    } else {
      n - > next = ptr - > next;
      ptr - > next = n;
      cout << "Node Inserted" << endl;
    }
  }
}

// 5. Delete node by unique key
void deleteNodeByKey(int k) {
  if (head == NULL) {
    cout << "Singly Linked List already Empty. Cant delete" << endl;
  } else if (head != NULL) {
    if (head - > key == k) {
      head = head - > next;
      cout << "Node UNLINKED with keys value : " << k << endl;
    } else {
      Node * temp = NULL;
      Node * prevptr = head;
      Node * currentptr = head - > next;
      while (currentptr != NULL) {
        if (currentptr - > key == k) {
          temp = currentptr;
          currentptr = NULL;
        } else {
          prevptr = prevptr - > next;
          currentptr = currentptr - > next;
        }
      }
      if (temp != NULL) {
        prevptr - > next = temp - > next;
        cout << "Node UNLINKED with keys value : " << k << endl;
      } else {
        cout << "Node Doesn't exist with key value : " << k << endl;
      }
    }
  }

}
// 6th update node
```

```
 void updateNodeByKey(int k, int d) {

   Node * ptr = nodeExists(k);
   if (ptr != NULL) {
     ptr - > data = d;
     cout << "Node Data Updated Successfully" << endl;
   } else {
     cout << "Node Doesn't exist with key value : " << k << endl;
   }

 }

 // 7th printing
 void printList() {
   if (head == NULL) {
     cout << "No Nodes in Singly Linked List";
   } else {
     cout << endl << "Singly Linked List Values : ";
     Node * temp = head;

     while (temp != NULL) {
       cout << "(" << temp - > key << "," << temp - > data << ") --> ";
       temp = temp - > next;
     }
   }

 }

};

int main() {

 SinglyLinkedList s;
 int option;
 int key1, k1, data1;
 do {
   cout << "\nWhat operation do you want to perform? Select Option number. Enter 0 to exit." << endl;
   cout << "1. appendNode()" << endl;
   cout << "2. prependNode()" << endl;
   cout << "3. insertNodeAfter()" << endl;
   cout << "4. deleteNodeByKey()" << endl;
   cout << "5. updateNodeByKey()" << endl;
   cout << "6. print()" << endl;
```

```
  cout << "7. Clear Screen" << endl << endl;

  cin >> option;
  Node * n1 = new Node();
  //Node n1;

  switch (option) {
  case 0:
    break;
  case 1:
    cout << "Append Node Operation \nEnter key & data of the Node to be Appended" << endl;
    cin >> key1;
    cin >> data1;
    n1 - > key = key1;
    n1 - > data = data1;
    s.appendNode(n1);
    //cout<<n1.key<<" = "<<n1.data<<endl;
    break;

  case 2:
    cout << "Prepend Node Operation \nEnter key & data of the Node to be Prepended" << endl;
    cin >> key1;
    cin >> data1;
    n1 - > key = key1;
    n1 - > data = data1;
    s.prependNode(n1);
    break;

  case 3:
    cout << "Insert Node After Operation \nEnter key of existing Node after which you want to Insert this New
node: " << endl;
    cin >> k1;
    cout << "Enter key & data of the New Node first: " << endl;
    cin >> key1;
    cin >> data1;
    n1 - > key = key1;
    n1 - > data = data1;

    s.insertNodeAfter(k1, n1);
    break;

  case 4:
      cout << "Delete Node By Key Operation - \nEnter key of the Node to be deleted: " << endl;
      cin >> k1;
      s.deleteNodeByKey(k1);

      break;
    case 5:
      cout << "Update Node By Key Operation - \nEnter key & NEW data to be updated" << endl;
      cin >> key1;
      cin >> data1;
      s.updateNodeByKey(key1, data1);

      break;
    case 6:
      s.printList();

      break;
    case 7:
      system("cls");
      break;
    default:
      cout << "Enter Proper Option number " << endl;
    }

  } while (option != 0);

  return 0;
}
```


D:\Pros\DS\singleLinkedL.exe

```
What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :6
No Nodes in Singly Linked List

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :1
Append Node Operation
Enter key & data of the Node to be Appended
1
25
Node Appended


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :1
Append Node Operation
Enter key & data of the Node to be Appended
2
40
Node Appended


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :6

Singly Linked List Values : (1,25) --> (2,40) -->

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :
```


D:\Pros\DS\singleLinkedL.exe

```
What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :4
Delete Node By Key Operation -
Enter key of the Node to be deleted:
1
Node UNLINKED with keys value : 1


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :5
Update Node By Key Operation -
Enter key & NEW data to be updated
2
50
Node Data Updated Successfully


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :6

Singly Linked List Values : (2,50) -->

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :2
Prepend Node Operation
Enter key & data of the Node to be Prepended
3
30
Node Prepended


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
```

Circular Linked List

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

class Node {
 public:
  int key;
 int data;
 Node * next;

 Node() {
  key = 0;
  data = 0;
  next = NULL;
 }
 Node(int k, int d) {
  key = k;
  data = d;
 }
};

class CircularLinkedList {
 public:
  Node * head;

 CircularLinkedList() {
  head = NULL;
 }

 // 1. CHeck if node exists using key value
 Node * nodeExists(int k) {

  Node * temp = NULL;
  Node * ptr = head;

  if (ptr == NULL) {
   return temp;
  } else {
   do {
    if (ptr - > key == k) {
     temp = ptr;
    }
```

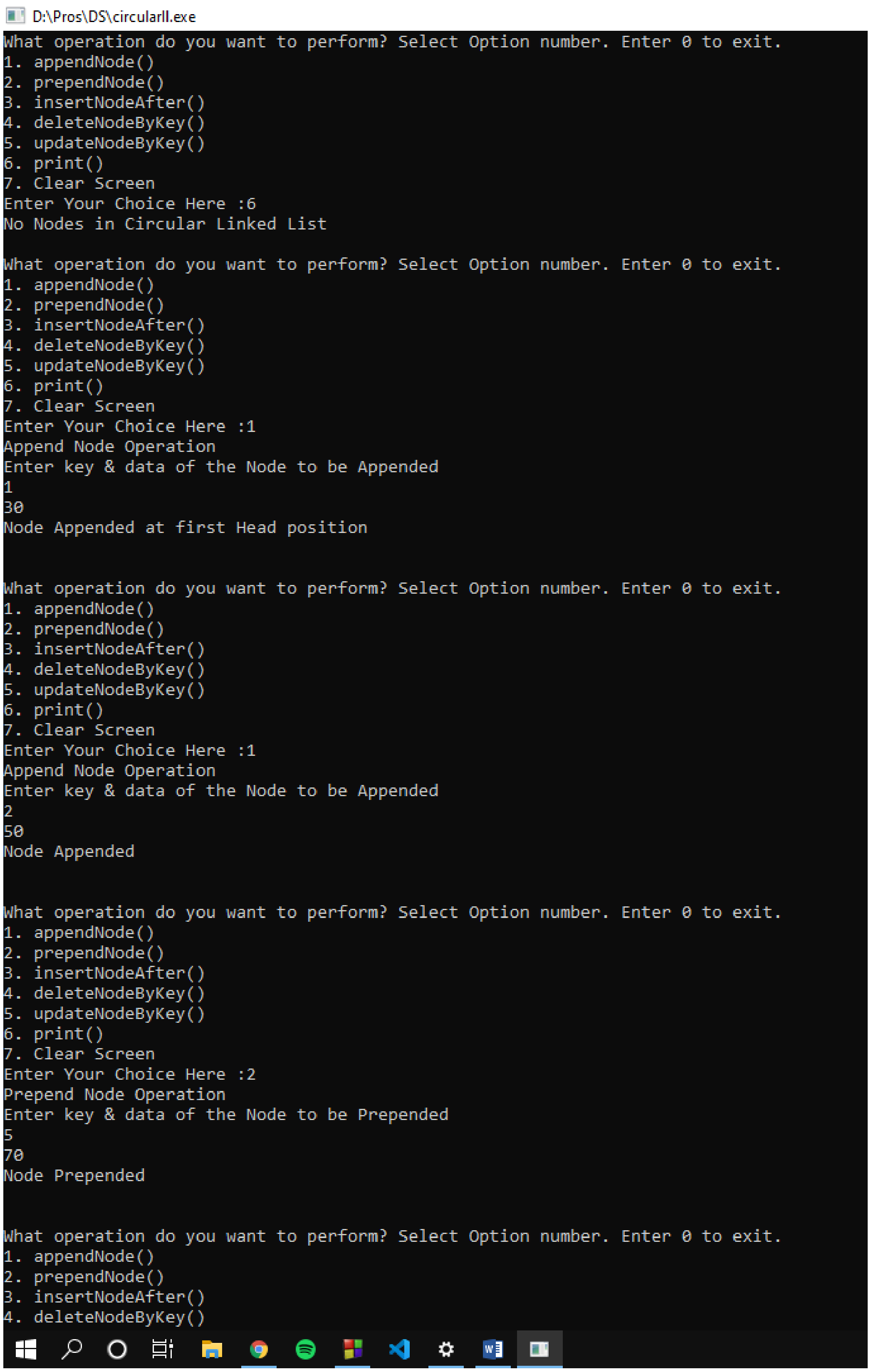
```
      ptr = ptr - > next;

    } while (ptr != head);
    return temp;
  }

  //return temp;
}

// 2. Append a node to the list
void appendNode(Node * new_node) {
  if (nodeExists(new_node - > key) != NULL) {
    cout << "Node Already exists with key value : " <<
      new_node - > key <<
      ". Append another node with different Key value" <<
      endl;
  } else {
    if (head == NULL) {
      head = new_node;
      new_node - > next = head;
      cout << "Node Appended at first Head position" << endl;
    } else {
      Node * ptr = head;
      while (ptr - > next != head) {
        ptr = ptr - > next;
      }
      ptr - > next = new_node;
      new_node - > next = head;
      cout << "Node Appended" << endl;
    }
  }

}
// 3. Prepend Node - Attach a node at the start
void prependNode(Node * new_node) {
  if (nodeExists(new_node - > key) != NULL) {
    cout << "Node Already exists with key value : " <<
      new_node - > key <<
      ". Append another node with different Key value" <<
      endl;
  } else {
    if (head == NULL) {
      head = new_node;
      new_node - > next = head;
      cout << "Node Prepended at first Head position" << endl;
    } else {
      Node * ptr = head;
      while (ptr - > next != head) {
        ptr = ptr - > next;
      }

      ptr - > next = new_node;
      new_node - > next = head;
      head = new_node;
      cout << "Node Prepended" << endl;
    }

  }
}

// 4. Insert a Node after a particular node in the list
void insertNodeAfter(int k, Node * new_node) {
  Node * ptr = nodeExists(k);
  if (ptr == NULL) {
    cout << "No node exists with key value OF: " << k << endl;
  } else {
    if (nodeExists(new_node - > key) != NULL) {
      cout << "Node Already exists with key value : " <<
        new_node - > key <<
        ". Append another node with different Key value" <<
        endl;
    } else {
      if (ptr - > next == head) {
        new_node - > next = head;
        ptr - > next = new_node;
        cout << "Node Inserted at the End" << endl;
      } else {
        new_node - > next = ptr - > next;
        ptr - > next = new_node;
        cout << "Node Inserted in between" << endl;
      }

    }
  }
}
```

```
// 5. Delete node by unique key
void deleteNodeByKey(int k) {
  Node * ptr = nodeExists(k);
  if (ptr == NULL) {
    cout << "No node exists with key value OF : " << k <<
      endl;
  } else {

    if (ptr == head) {
      if (head - > next == NULL) {
        head = NULL;
        cout << "Head node Unlinked... List Empty";
      } else {
        Node * ptr1 = head;
        while (ptr1 - > next != head) {
          ptr1 = ptr1 - > next;
        }
        ptr1 - > next = head - > next;
        head = head - > next;
        cout << "Node UNLINKED with keys value : " << k << endl;
      }
    } else {
      Node * temp = NULL;
      Node * prevptr = head;
      Node * currentptr = head - > next;
      while (currentptr != NULL) {
        if (currentptr - > key == k) {
          temp = currentptr;
          currentptr = NULL;
        } else {
          prevptr = prevptr - > next;
          currentptr = currentptr - > next;
        }
      }

      prevptr - > next = temp - > next;
      cout << "Node UNLINKED with keys value : " << k << endl;

    }

  }

}
```

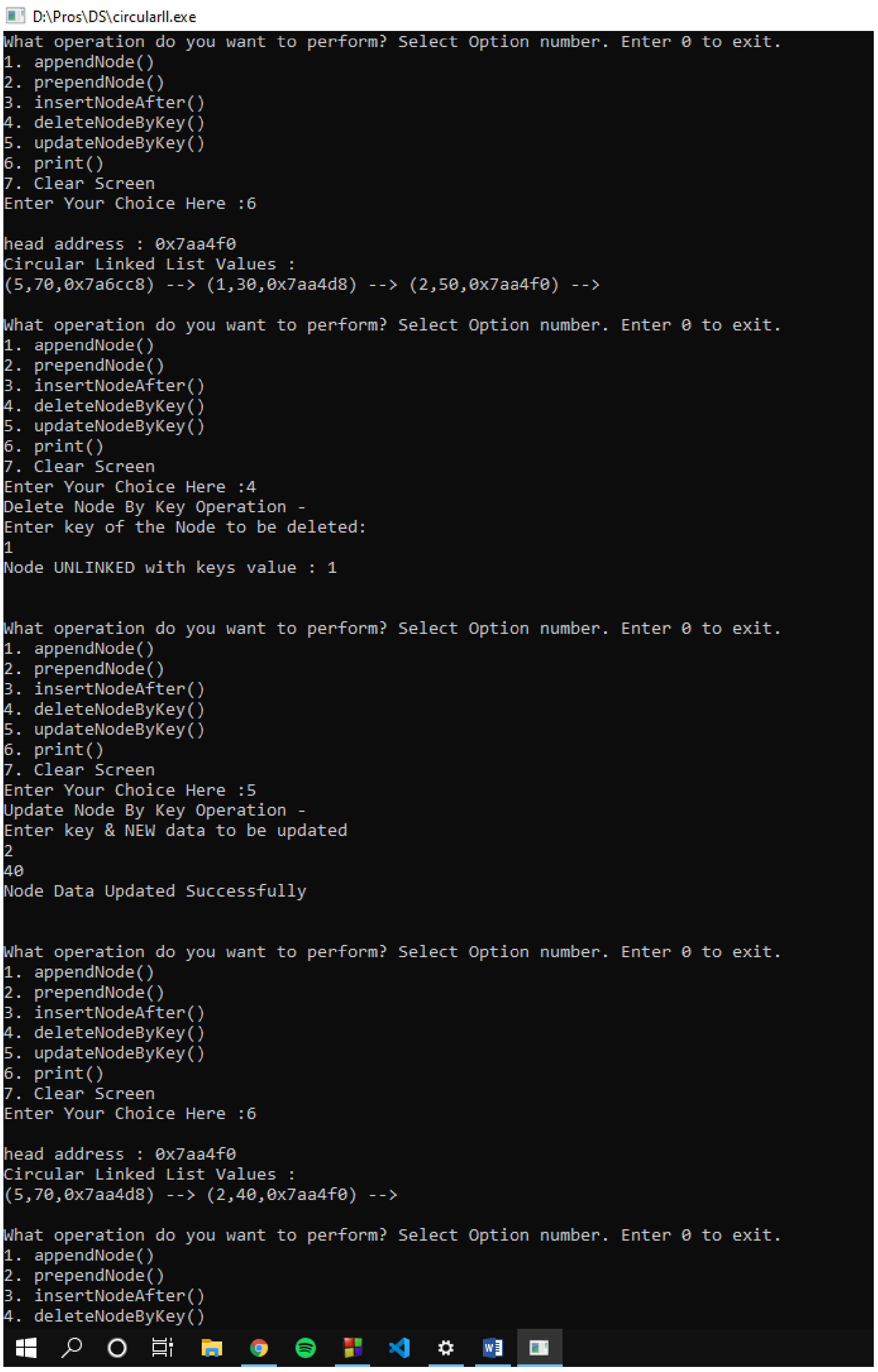
```
 // 6th update node
 void updateNodeByKey(int k, int new_data) {

   Node * ptr = nodeExists(k);
   if (ptr != NULL) {
     ptr - > data = new_data;
     cout << "Node Data Updated Successfully" << endl;
   } else {
     cout << "Node Doesn't exist with key value : " << k << endl;
   }

 }

 // 7th printing
 void printList() {
   if (head == NULL) {
     cout << "No Nodes in Circular Linked List";
   } else {
     cout << endl << "head address : " << head << endl;
     cout << "Circular Linked List Values : " << endl;

     Node * temp = head;

     do {
       cout << "(" << temp - > key << "," << temp - > data << "," << temp - > next << ") --> ";
       temp = temp - > next;
     } while (temp != head);
   }

 }

};

int main() {

 CircularLinkedList obj;
 int option;
 int key1, k1, data1;
 do {
   cout << "\nWhat operation do you want to perform? Select Option number. Enter 0 to exit." << endl;
   cout << "1. appendNode()" << endl;
   cout << "2. prependNode()" << endl;
   cout << "3. insertNodeAfter()" << endl;
```

```
  cout << "4. deleteNodeByKey()" << endl;
  cout << "5. updateNodeByKey()" << endl;
  cout << "6. print()" << endl;
  cout << "7. Clear Screen" << endl << endl;

  cin >> option;
  Node * n1 = new Node();
  //Node n1;

  switch (option) {
  case 0:
    break;
  case 1:
    cout << "Append Node Operation \nEnter key & data of the Node to be Appended" << endl;
    cin >> key1;
    cin >> data1;
    n1 - > key = key1;
    n1 - > data = data1;
    obj.appendNode(n1);
    //cout<<n1.key<<" = "<<n1.data<<endl;
    break;

  case 2:
    cout << "Prepend Node Operation \nEnter key & data of the Node to be Prepended" << endl;
    cin >> key1;
    cin >> data1;
    n1 - > key = key1;
    n1 - > data = data1;
    obj.prependNode(n1);
    break;

  case 3:
    cout << "Insert Node After Operation \nEnter key of existing Node after which you want to Insert this New
node: " << endl;
    cin >> k1;
    cout << "Enter key & data of the New Node first: " << endl;
    cin >> key1;
    cin >> data1;
    n1 - > key = key1;
    n1 - > data = data1;

    obj.insertNodeAfter(k1, n1);
    break;
```

```
   case 4:

     cout << "Delete Node By Key Operation - \nEnter key of the Node to be deleted: " << endl;
     cin >> k1;
     obj.deleteNodeByKey(k1);

     break;
   case 5:
     cout << "Update Node By Key Operation - \nEnter key & NEW data to be updated" << endl;
     cin >> key1;
     cin >> data1;
     obj.updateNodeByKey(key1, data1);

     break;
   case 6:
     obj.printList();

     break;
   case 7:
     system("cls");
     break;
   default:
     cout << "Enter Proper Option number " << endl;
   }

  } while (option != 0);

  return 0;
}
```



```
D:\Pros\DS\circularll.exe
What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :6
No Nodes in Circular Linked List

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :1
Append Node Operation
Enter key & data of the Node to be Appended
1
30
Node Appended at first Head position


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :1
Append Node Operation
Enter key & data of the Node to be Appended
2
50
Node Appended


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :2
Prepend Node Operation
Enter key & data of the Node to be Prepended
5
70
Node Prepended


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
```



D:\Pros\DS\circularll.exe

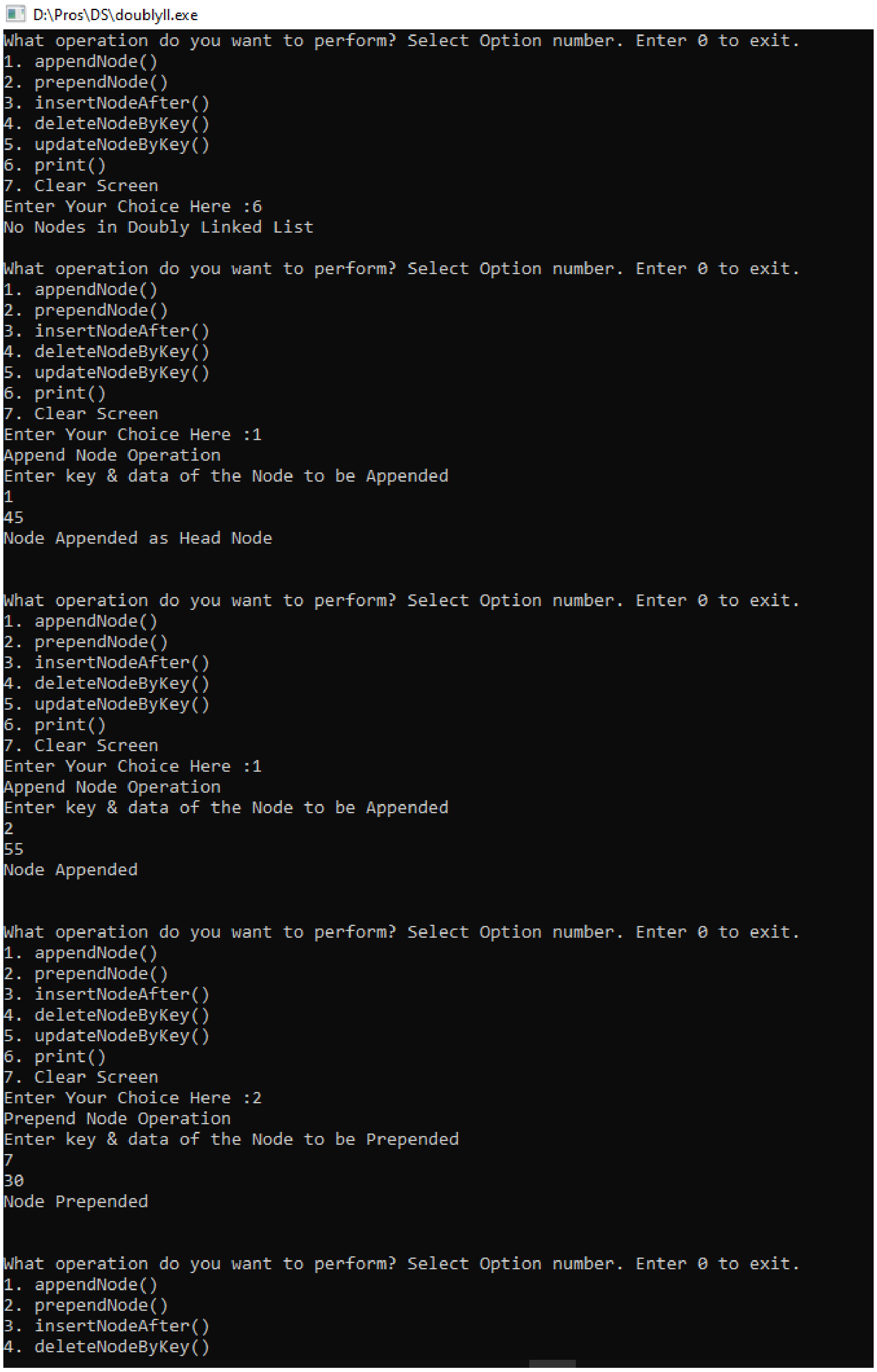
```
What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :6

head address : 0x7aa4f0
Circular Linked List Values :
(5,70,0x7a6cc8) --> (1,30,0x7aa4d8) --> (2,50,0x7aa4f0) -->

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :4
Delete Node By Key Operation -
Enter key of the Node to be deleted:
1
Node UNLINKED with keys value : 1


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :5
Update Node By Key Operation -
Enter key & NEW data to be updated
2
40
Node Data Updated Successfully


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :6

head address : 0x7aa4f0
Circular Linked List Values :
(5,70,0x7aa4d8) --> (2,40,0x7aa4f0) -->

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
```

Doubly Linked List

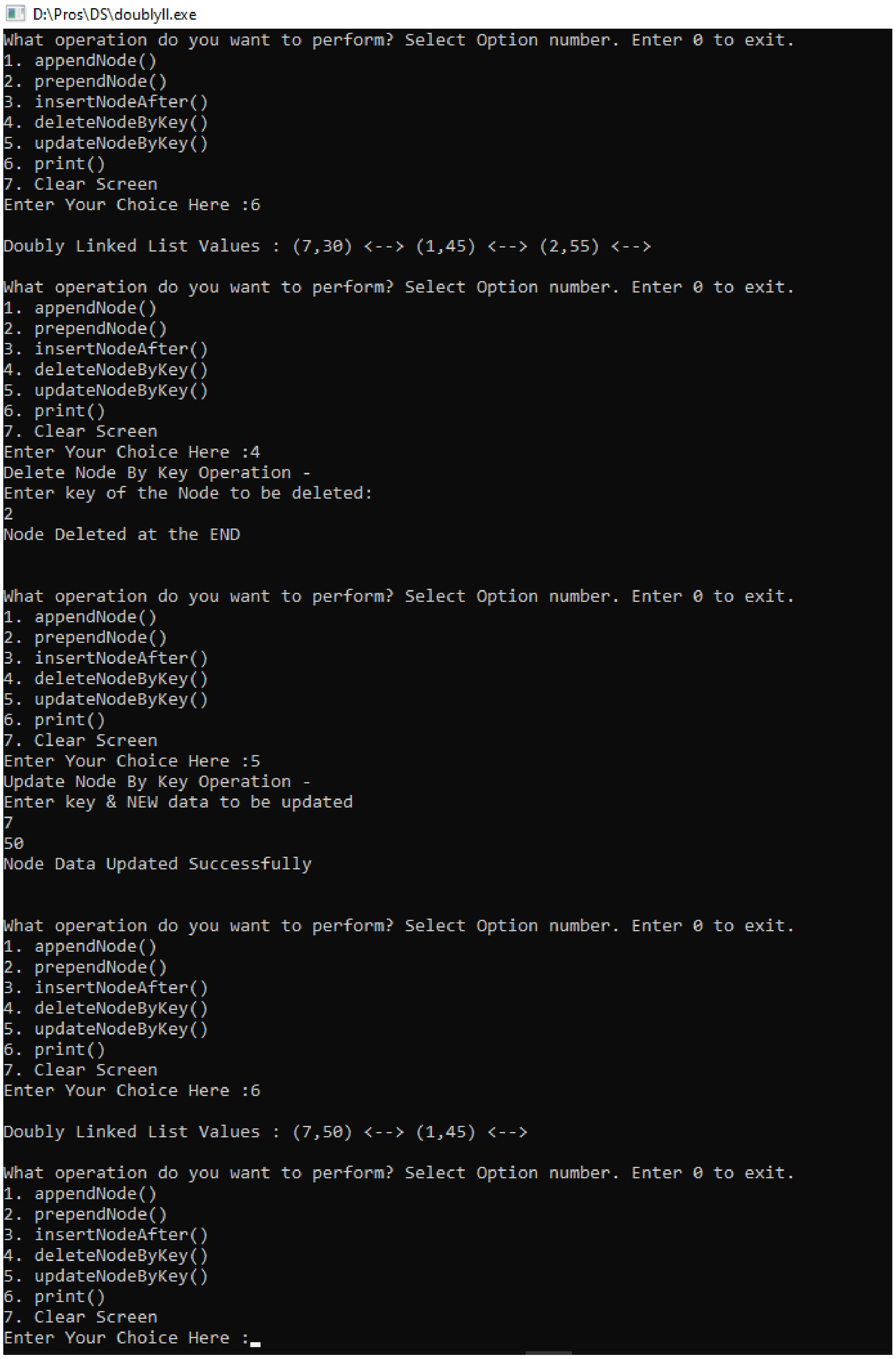
```
#include<iostream>
using namespace std;

class Node {
 public:
  int key;
 int data;
 Node * next;
 Node * previous;

 Node() {
  key = 0;
  data = 0;
  next = NULL;
  previous = NULL;
 }
 Node(int k, int d) {
  key = k;
  data = d;
 }
};

class DoublyLinkedList {

 public:
  Node * head;

 DoublyLinkedList() {
  head = NULL;
 }
 DoublyLinkedList(Node * n) {
  head = n;
 }

 // 1. CHeck if node exists using key value

 Node * nodeExists(int k) {
  Node * temp = NULL;
  Node * ptr = head;

  while (ptr != NULL) {
      if (ptr - > key == k) {
        temp = ptr;
      }
      ptr = ptr - > next;
    }

    return temp;
  }

  // 2. Append a node to the list

  void appendNode(Node * n) {
    if (nodeExists(n - > key) != NULL) {
      cout << "Node Already exists with key value : " << n - > key << ". Append another node with different Key
value" << endl;
    } else {
      if (head == NULL) {
        head = n;
        cout << "Node Appended as Head Node" << endl;
      } else {
        Node * ptr = head;
        while (ptr - > next != NULL) {
          ptr = ptr - > next;
        }
        ptr - > next = n;
        n - > previous = ptr;
        cout << "Node Appended" << endl;
      }
    }
  }

  // 3. Prepend Node - Attach a node at the start
  void prependNode(Node * n) {
    if (nodeExists(n - > key) != NULL) {
      cout << "Node Already exists with key value : " << n - > key << ". Append another node with different Key
value" << endl;
    } else {
      if (head == NULL) {
        head = n;
        cout << "Node Prepended as Head Node" << endl;
      } else {
        head - > previous = n;
        n - > next = head;
        head = n;
        cout << "Node Prepended" << endl;
      }

    }
  }

  // 4. Insert a Node after a particular node in the list
  void insertNodeAfter(int k, Node * n) {
    Node * ptr = nodeExists(k);
    if (ptr == NULL) {
      cout << "No node exists with key value: " << k << endl;
    } else {
      if (nodeExists(n - > key) != NULL) {
        cout << "Node Already exists with key value : " << n - > key << ". Append another node with different Key
  value" << endl;
      } else {
        Node * nextNode = ptr - > next;
        // inserting at the end
        if (nextNode == NULL) {
          ptr - > next = n;
          n - > previous = ptr;
          cout << "Node Inserted at the END" << endl;
        }

        //inserting in between
        else {
          n - > next = nextNode;
          nextNode - > previous = n;
          n - > previous = ptr;
          ptr - > next = n;

          cout << "Node Inserted in Between" << endl;

        }

      }
    }
  }

  // 5. Delete node by unique key. Basically De-Link not delete
  void deleteNodeByKey(int k) {
    Node * ptr = nodeExists(k);
```

```
 if (ptr == NULL) {
   cout << "No node exists with key value: " << k << endl;
 } else {

   if (head - > key == k) {
     head = head - > next;
     cout << "Node UNLINKED with keys value : " << k << endl;
   } else {
     Node * nextNode = ptr - > next;
     Node * prevNode = ptr - > previous;
     // deleting at the end
     if (nextNode == NULL) {

       prevNode - > next = NULL;
       cout << "Node Deleted at the END" << endl;
     }

     //deleting in between
     else {
       prevNode - > next = nextNode;
       nextNode - > previous = prevNode;
       cout << "Node Deleted in Between" << endl;

     }
   }
 }
}

// 6th update node
void updateNodeByKey(int k, int d) {

 Node * ptr = nodeExists(k);
 if (ptr != NULL) {
   ptr - > data = d;
   cout << "Node Data Updated Successfully" << endl;
 } else {
   cout << "Node Doesn't exist with key value : " << k << endl;
 }

}

// 7th printing
void printList() {
    if (head == NULL) {
      cout << "No Nodes in Doubly Linked List";
    } else {
      cout << endl << "Doubly Linked List Values : ";
      Node * temp = head;

      while (temp != NULL) {
        cout << "(" << temp - > key << "," << temp - > data << ") <--> ";
        temp = temp - > next;
      }
    }

  }

};

int main() {

  DoublyLinkedList obj;
  int option;
  int key1, k1, data1;
  do {
    cout << "\nWhat operation do you want to perform? Select Option number. Enter 0 to exit." << endl;
    cout << "1. appendNode()" << endl;
    cout << "2. prependNode()" << endl;
    cout << "3. insertNodeAfter()" << endl;
    cout << "4. deleteNodeByKey()" << endl;
    cout << "5. updateNodeByKey()" << endl;
    cout << "6. print()" << endl;
    cout << "7. Clear Screen" << endl << endl;

    cin >> option;
    Node * n1 = new Node();
    //Node n1;

    switch (option) {
    case 0:
      break;
    case 1:
      cout << "Append Node Operation \nEnter key & data of the Node to be Appended" << endl;
      cin >> key1;
      cin >> data1;
      n1 - > key = key1;
    n1 - > data = data1;
    obj.appendNode(n1);
    //cout<<n1.key<<" = "<<n1.data<<endl;
    break;

  case 2:
    cout << "Prepend Node Operation \nEnter key & data of the Node to be Prepended" << endl;
    cin >> key1;
    cin >> data1;
    n1 - > key = key1;
    n1 - > data = data1;
    obj.prependNode(n1);
    break;

  case 3:
    cout << "Insert Node After Operation \nEnter key of existing Node after which you want to Insert this New
node: " << endl;
    cin >> k1;
    cout << "Enter key & data of the New Node first: " << endl;
    cin >> key1;
    cin >> data1;
    n1 - > key = key1;
    n1 - > data = data1;

    obj.insertNodeAfter(k1, n1);
    break;

  case 4:

    cout << "Delete Node By Key Operation - \nEnter key of the Node to be deleted: " << endl;
    cin >> k1;
    obj.deleteNodeByKey(k1);

    break;
  case 5:
    cout << "Update Node By Key Operation - \nEnter key & NEW data to be updated" << endl;
    cin >> key1;
    cin >> data1;
    obj.updateNodeByKey(key1, data1);

    break;
  case 6:
    obj.printList();
```

```
      break;
    case 7:
      system("cls");
      break;
    default:
      cout << "Enter Proper Option number " << endl;
    }

  } while (option != 0);

  return 0;
}
```


D:\Pros\DS\doublyll.exe

```
What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :6
No Nodes in Doubly Linked List

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :1
Append Node Operation
Enter key & data of the Node to be Appended
1
45
Node Appended as Head Node


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :1
Append Node Operation
Enter key & data of the Node to be Appended
2
55
Node Appended


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :2
Prepend Node Operation
Enter key & data of the Node to be Prepended
7
30
Node Prepended


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
```



D:\Pros\DS\doublyll.exe

```
What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :6

Doubly Linked List Values : (7,30) <--> (1,45) <--> (2,55) <-->

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :4
Delete Node By Key Operation -
Enter key of the Node to be deleted:
2
Node Deleted at the END


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :5
Update Node By Key Operation -
Enter key & NEW data to be updated
7
50
Node Data Updated Successfully


What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :6

Doubly Linked List Values : (7,50) <--> (1,45) <-->

What operation do you want to perform? Select Option number. Enter 0 to exit.
1. appendNode()
2. prependNode()
3. insertNodeAfter()
4. deleteNodeByKey()
5. updateNodeByKey()
6. print()
7. Clear Screen
Enter Your Choice Here :_
```