COMSATS University Islamabad, Vehari Campus

Department of Computer Science

Class: BCS-SP22 **Submission Deadline: 9 Oct 2023**

Subject: Data Structures and Algorithms-Lab **Instructor: Yasmeen Jana**

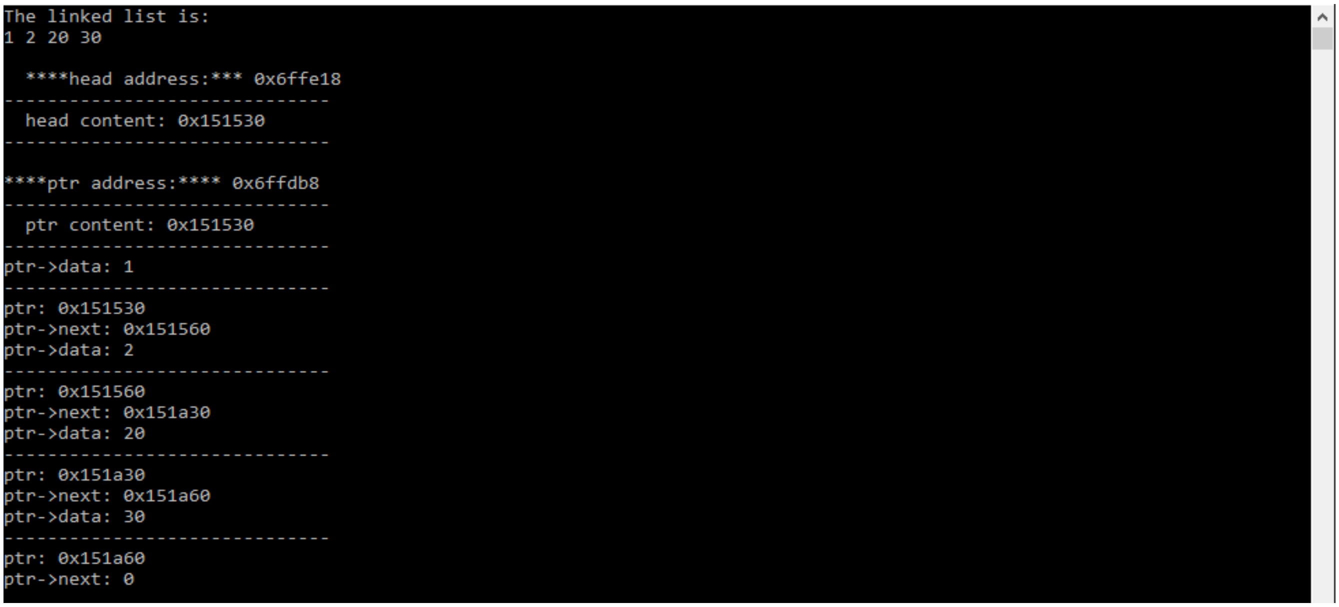
Max Marks: 20 **Reg. No: sp22-bcs-057**

Email: yasmeenjana@cuivehari.edu.pk

You can ask queries related to Lab Activities on the above email.

Activity 1:

Create a function to display linked list output as below:



```
The linked list is:
1 2 20 30

  ****head address:*** 0x6ffe18
------------------------------
  head content: 0x151530
------------------------------

****ptr address:**** 0x6ffdb8
------------------------------
  ptr content: 0x151530
------------------------------
ptr->data: 1
------------------------------
ptr: 0x151530
ptr->next: 0x151560
ptr->data: 2
------------------------------
ptr: 0x151560
ptr->next: 0x151a30
ptr->data: 20
------------------------------
ptr: 0x151a30
ptr->next: 0x151a60
ptr->data: 30
------------------------------
ptr: 0x151a60
ptr->next: 0
```

Answer

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

class List {
   struct Node {
      int data;
      Node* next;
      Node* prev;

      Node(int value) : data(value), next(NULL), prev(NULL) {}
   };

   Node* head_singly;
   Node* head_doubly;
   Node* head_circular;

public:
   List() {
      head_singly = NULL;
      head_doubly = NULL;
      head_circular = NULL;
}


// Functions for inserting into singly, doubly, and circular linked lists
void insert_beg_singly(int n) {
    Node* newNode = new Node(n);
    newNode->next = head_singly;
    head_singly = newNode;
}


void insert_end_singly(int n) {
    Node* newNode = new Node(n);
    if (head_singly == NULL) {
        head_singly = newNode;
    } else {
        Node* current = head_singly;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
}
```

```
void insert_beg_doubly(int n) {
    Node* newNode = new Node(n);
    newNode->next = head_doubly;
    newNode->prev = NULL;

    if (head_doubly != NULL) {
        head_doubly->prev = newNode;
    }

    head_doubly = newNode;
}

void insert_end_doubly(int n) {
    Node* newNode = new Node(n);
    newNode->next = NULL;

    if (head_doubly == NULL) {
        newNode->prev = NULL;
        head_doubly = newNode;
    } else {
        Node* current = head_doubly;
        while (current->next != NULL) {
            current = current->next;
        }

        newNode->prev = current;
        current->next = newNode;
    }
}

void insert_beg_circular(int n) {
    Node* newNode = new Node(n);
    if (head_circular == NULL) {
        head_circular = newNode;
        head_circular->next = head_circular;
    } else {
        Node* current = head_circular;
        while (current->next != head_circular) {
            current = current->next;
        }
        newNode->next = head_circular;
        head_circular = newNode;
        current->next = head_circular;
    }
```

```
}

void insert_end_circular(int n) {
    Node* newNode = new Node(n);
    if (head_circular == NULL) {
        head_circular = newNode;
        head_circular->next = head_circular;
    } else {
        Node* current = head_circular;
        while (current->next != head_circular) {
            current = current->next;
        }
        newNode->next = head_circular;
        current->next = newNode;
    }
}

// Functions for deleting from singly, doubly, and circular linked lists
void delete_beg_singly() {
    if (head_singly == NULL) {
        return; // List is empty
    }
    Node* temp = head_singly;
    head_singly = head_singly->next;
    delete temp;
}

void delete_end_singly() {
    if (head_singly == NULL) {
        return; // List is empty
    } else if (head_singly->next == NULL) {
        delete head_singly;
        head_singly = NULL;
    } else {
        Node* current = head_singly;
        while (current->next->next != NULL) {
            current = current->next;
        }
        delete current->next;
        current->next = NULL;
    }
}

void delete_beg_doubly() {
    if (head_doubly == NULL) {
        return; // List is empty
    }
    Node* temp = head_doubly;
    head_doubly = head_doubly->next;
    if (head_doubly != NULL) {
        head_doubly->prev = NULL;
    }
    delete temp;
}

void delete_end_doubly() {
    if (head_doubly == NULL) {
        return; // List is empty
    } else if (head_doubly->next == NULL) {
        delete head_doubly;
        head_doubly = NULL;
    } else {
        Node* current = head_doubly;
        while (current->next->next != NULL) {
            current = current->next;
        }
        delete current->next;
        current->next = NULL;
    }
}

void delete_beg_circular() {
    if (head_circular == NULL) {
        return; // List is empty
    }
    Node* current = head_circular;
    while (current->next != head_circular) {
        current = current->next;
    }
    Node* temp = head_circular;
    head_circular = head_circular->next;
    current->next = head_circular;
    delete temp;
}

void delete_end_circular() {
    if (head_circular == NULL) {
        return; // List is empty
    } else if (head_circular->next == head_circular) {
        delete head_circular;
        head_circular = NULL;
    } else {
        Node* current = head_circular;
        Node* previous = NULL;
        while (current->next != head_circular) {
            previous = current;
            current = current->next;
        }
        previous->next = head_circular;
        delete current;
    }
}

// Functions for seeking in singly, doubly, and circular linked lists
int seek_singly(int value) {
    Node* current = head_singly;
    int index = 0;

    while (current != NULL) {
        if (current->data == value) {
            return index; // Value found at this index
        }
        current = current->next;
        index++;
    }

    return -1; // Value not found in the list
}

int seek_doubly(int value) {
    Node* current = head_doubly;
    int forwardIndex = 0;

    // Forward traversal
    while (current != NULL) {
        if (current->data == value) {
            return forwardIndex; // Value found at this index
        }
        current = current->next;
        forwardIndex++;
    }
    // Value not found in the forward direction, let's try backward
    current = head_doubly;
    int backwardIndex = 0;
    while (current->next != NULL) {
        current = current->next;
    }

    while (current != NULL) {
        if (current->data == value) {
            return backwardIndex; // Value found at this index
        }
        current = current->prev;
        backwardIndex--;
    }

    return -1; // Value not found in the list
}

int seek_circular(int value) {
    if (head_circular == NULL) {
        return -1; // List is empty
    }
    Node* current = head_circular;
    int index = 0;

    do {
        if (current->data == value) {
            return index; // Value found at this index
        }
        current = current->next;
        index++;
    } while (current != head_circular);

    return -1; // Value not found in the list
}

// Functions for reversing singly, doubly, and circular linked lists
void reverse_singly() {
    Node* prev = NULL;
    Node* current = head_singly;
    Node* nextNode = NULL;

    while (current != NULL) {
        nextNode = current->next;

        current->next = prev;

        prev = current;

        current = nextNode;

    }


    head_singly = prev;

}


void reverse_doubly() {

    Node* current = head_doubly;

    Node* temp = NULL;

    while (current != NULL) {

        temp = current->prev;

        current->prev = current->next;

        current->next = temp;

        current = current->prev;

    }

    if (temp != NULL) {

        head_doubly = temp->prev;

    }

}
```

```
void reverse_circular() {
    if (head_circular == NULL) {
        return; // List is empty
    }

    Node* current = head_circular;
    Node* prev = NULL;
    Node* nextNode = NULL;

    do {
        nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    } while (current != head_circular);

    head_circular->next = prev;
    head_circular = prev;
}

// Display functions for singly, doubly, and circular linked lists
```

```
void display_singly() {
    Node* current = head_singly;
    while (current != NULL) {
        cout << current->data << " -> ";
        current = current->next;
    }
    cout << "NULL" << endl;
}

void display_doubly() {
    Node* current = head_doubly;
    while (current != NULL) {
        cout << current->data << " <-> ";
        current = current->next;
    }
    cout << "NULL" << endl;
}

void display_circular() {
    if (head_circular == NULL) {
        cout << "Empty Circular Linked List" << endl;
        return;
        }

        Node* current = head_circular;
        do {
            cout << current->data << " -> ";
            current = current->next;
        } while (current != head_circular);
        cout << "Head" << endl;
    }
};

int main() {
    List l;
    int ch, val;

    do {
        cout << "\nOperations on link list" << endl;
        cout << "1-Insertion \n2-Deletion \n3-Seek \n4-Reverse \n5-Display \n6-Exit" << endl;
        cout << "\nEnter Your Choice: ";
        cin >> ch;
        int ps, ds;
        switch (ch) {
```

```
case 1:
  cout << "\n1-Insertion in Singly \n2-Insertion in Doubly \n3-Insertion in Circular" << endl;
  cout << "\nEnter your choice: ";
  cin >> ps;

  switch (ps) {
    case 1:
      cout << "\n1-Insertion at beginning \n2-Insertion at end\n";
      cout << "\nEnter Your Choice: ";
      cin >> ds;
      cout << "\nEnter Value to insert: ";
      cin >> val;
      switch (ds) {
        case 1:
          l.insert_beg_singly(val);
          cout<<"\nData after insertion"<<endl;
          l.display_singly();
          break;
        case 2:
          l.insert_end_singly(val);
          cout<<"\nData after insertion"<<endl;
          l.display_singly();
```

```
            break;
        default:
            cout << "\nInvalid choice." << endl;
    }
    break;
case 2:
    cout << "\n1-Insertion at beginning \n2-Insertion at end" << endl;
    cout << "\nEnter Your Choice: ";
    cin >> ds;
    cout << "\nEnter Value to insert: ";
    cin >> val;
    switch (ds) {
        case 1:
            l.insert_beg_doubly(val);
            cout<<"\nData after insertion"<<endl;
            l.display_doubly();
            break;
        case 2:
            l.insert_end_doubly(val);
            cout<<"\nData after insertion"<<endl;
            l.display_doubly();
            break;
        default:
            cout << "Invalid choice." << endl;
    }
    break;
case 3:
    cout << "\n1-Insertion at beginning \n2-Insertion at end" << endl;
    cout << "\nEnter Your Choice: ";
    cin >> ds;
    cout << "\nEnter Value to insert: ";
    cin >> val;
    switch (ds) {
        case 1:
            l.insert_beg_circular(val);
            cout<<"\nData after insertion"<<endl;
            l.display_circular();
            break;
        case 2:
            l.insert_end_circular(val);
            cout<<"\nData after insertion"<<endl;
            l.display_circular();
            break;
        default:
```

```
          cout << "Invalid choice." << endl;

      }

      break;

    default:

      cout << "Invalid choice." << endl;

  }

  break;


case 2:

  cout << "\n1-Deletion in Singly \n2-Deletion in Doubly \n3-Deletion in Circular" << endl;

  cout << "\nEnter your choice: ";

  cin >> ps;

  switch (ps) {

    case 1:

      cout << "\n1-Deletion at beginning \n2-Deletion at end ";

      cout << "\nEnter Your Choice: ";

      cin >> ds;


      switch (ds) {

        case 1:

          l.delete_beg_singly();

          cout<<"\nData after deletion"<<endl;
```

```
            l.display_singly();

            break;

        case 2:

            l.delete_end_singly();

            cout<<"\nData after deletion"<<endl;

            l.display_singly();

            break;

        default:

            cout << "Invalid choice." << endl;

    }

    break;

case 2:

    cout << "\n1-Deletion at beginning \n2-Deletion at end" << endl;

    cout << "\nEnter Your Choice: ";

    cin >> ds;

    switch (ds) {

        case 1:

            l.delete_beg_doubly();

            cout<<"\nData after deletion"<<endl;

            l.display_doubly();

            break;

        case 2:
```

```
      l.delete_end_doubly();
      cout<<"\nData after deletion"<<endl;
      l.display_doubly();
      break;
    default:
      cout << "Invalid choice." << endl;
      break;
  }
  break;
case 3:
  cout << "\n1-Deletion at beginning \n2-Deletion at end" << endl;
  cout << "\nEnter Your Choice: ";
  cin >> ds;
  switch (ds) {
    case 1:
      l.delete_beg_circular();
      cout<<"\nData after deletion"<<endl;
      l.display_circular();
      break;
    case 2:
      l.delete_end_circular();
      cout<<"\nData after deletion"<<endl;
```

```
               l.display_circular();
               break;
            default:
               cout << "Invalid choice." << endl;
               break;
         }
         break;
      default:
         cout << "Invalid choice." << endl;
         break;
   }
   break;

case 3:
   cout << "\n1-Seek in Singly \n2-Seek in Doubly \n3-Seek in Circular" << endl;
   cout << "\nEnter your choice: ";
   cin >> ps;
   switch (ps) {
      case 1:
         cout << "Enter the value to seek: ";
         cin >> val;
         int position_singly;
```

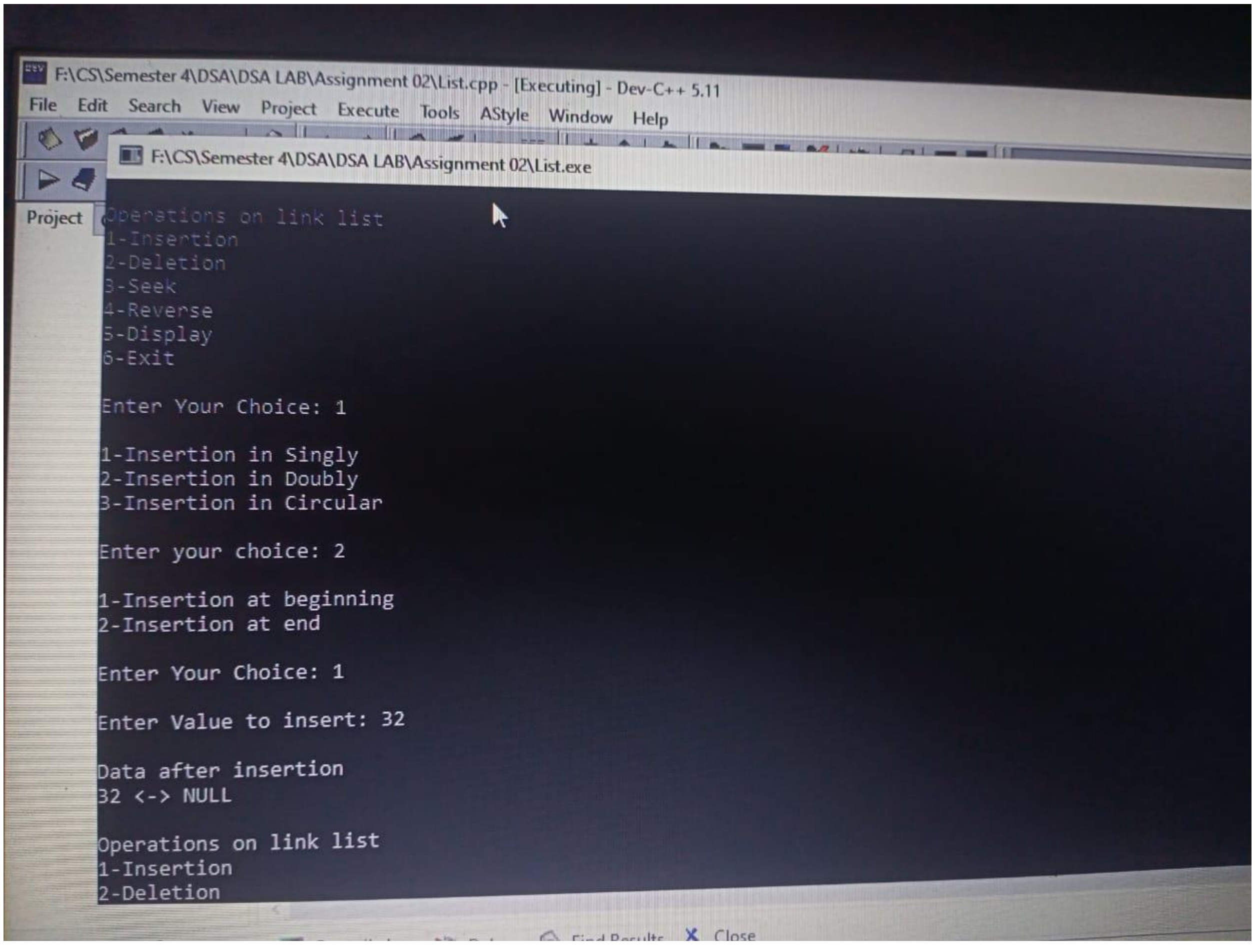
```
    position_singly = l.seek_singly(val);
    if (position_singly != -1) {
        cout << "Value found at position " << position_singly << endl;
    } else {
        cout << "Value not found in the singly linked list." << endl;
    }
    break;
case 2:
    cout << "Enter the value to seek: ";
    cin >> val;
    int position_doubly;
    position_doubly = l.seek_doubly(val);
    if (position_doubly != -1) {
        cout << "Value found at position " << position_doubly << endl;
    } else {
        cout << "Value not found in the doubly linked list." << endl;
    }
    break;
case 3:
    cout << "Enter the value to seek: ";
    cin >> val;
    int position_circular;
```

```
      position_circular = l.seek_circular(val);
      if (position_circular != -1) {
        cout << "Value found at position " << position_circular << endl;
      } else {
        cout << "Value not found in the circular linked list." << endl;
      }
      break;
    default:
      cout << "Invalid choice." << endl;
      break;
  }
  break;

case 4:
  cout<<"\n1-Reverse in singly \n2-Reverse in Doubly \n3-Reverse in Circular\n";
  cout << "\nEnter your choice: ";
  cin >> ps;
  switch(ps){
    case 1:
      l.reverse_singly();
      cout<<"\nData after reversal"<<endl;
      l.display_singly();
```

```
            break;
        case 2:
            l.reverse_doubly();
            cout<<"\nData after reversal"<<endl;
            l.display_doubly();
            break;
        case 3:
            l.reverse_circular();
            cout<<"\nData after reversal"<<endl;
            l.display_circular();
            break;
        default:
            cout<<"Invalid Command"<<endl;
    }
    break;

case 5:
    cout << "\n1-Display Singly \n2-Display Doubly \n3-Display Circular" << endl;
    cout << "\nEnter your choice: ";
    cin >> ps;
    switch (ps) {
        case 1:
```

```
      cout << "Data in Singly Linked List:" << endl;
      l.display_singly();
      break;
    case 2:
      cout << "Data in Doubly Linked List:" << endl;
      l.display_doubly();
      break;
    case 3:
      cout << "Data in Circular Linked List:" << endl;
      l.display_circular();
      break;
    default:
      cout << "Invalid choice." << endl;
      break;
  }
  break;

case 6:
  return 0; // Exit the program

default:
  cout << "Invalid choice." << endl;
```

```
            break;

        }

    } while (ch != 6);



    return 0;

}
```


F:\CS\Semester 4\DSA\DSA LAB\Assignment 02\List.cpp - [Executing] - Dev-C++ 5.11
File
Edit
Search
View
Project
Execute
Tools
AStyle
Window
Help
F:\CS\Semester 4\DSA\DSA LAB\Assignment 02\List.exe
Project
Find Results
Close

```
Operations on link list
1-Insertion
2-Deletion
3-Seek
4-Reverse
5-Display
6-Exit

Enter Your Choice: 1

1-Insertion in Singly
2-Insertion in Doubly
3-Insertion in Circular

Enter your choice: 2

1-Insertion at beginning
2-Insertion at end

Enter Your Choice: 1

Enter Value to insert: 32

Data after insertion
32 <-> NULL

Operations on link list
1-Insertion
2-Deletion
```

Activity 2:

Write a program that will implement single, doubly, and circular linked link list operations by showing a menu to the user.

The menu should be:

Which linked list you want:

1: Single

2: Double

3: Circular

After the option is chosen by the user:

Which operation you want to perform:

1: Insertion

2: Deletion

3: Display

4: Reverse

4: Seek

5: Exit

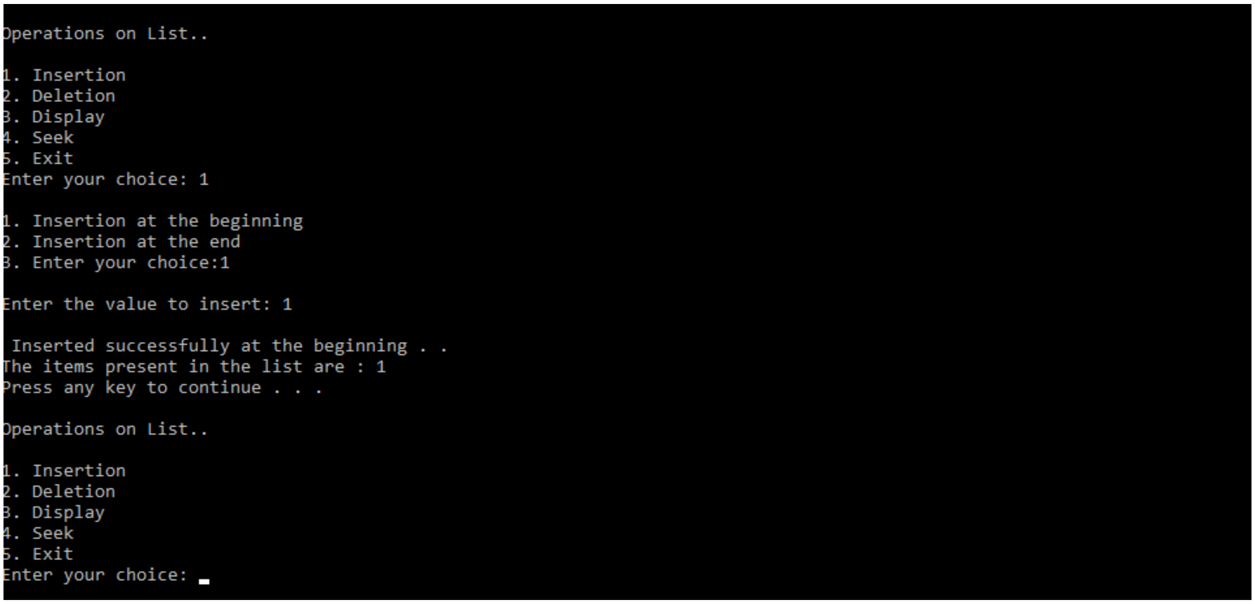
Let's suppose, the user has chosen the insertion option then the following menu should be displayed:

1: insertion at beginning

2: insertion at end

3: insertion at the specific data node

A sample output screenshot is below:



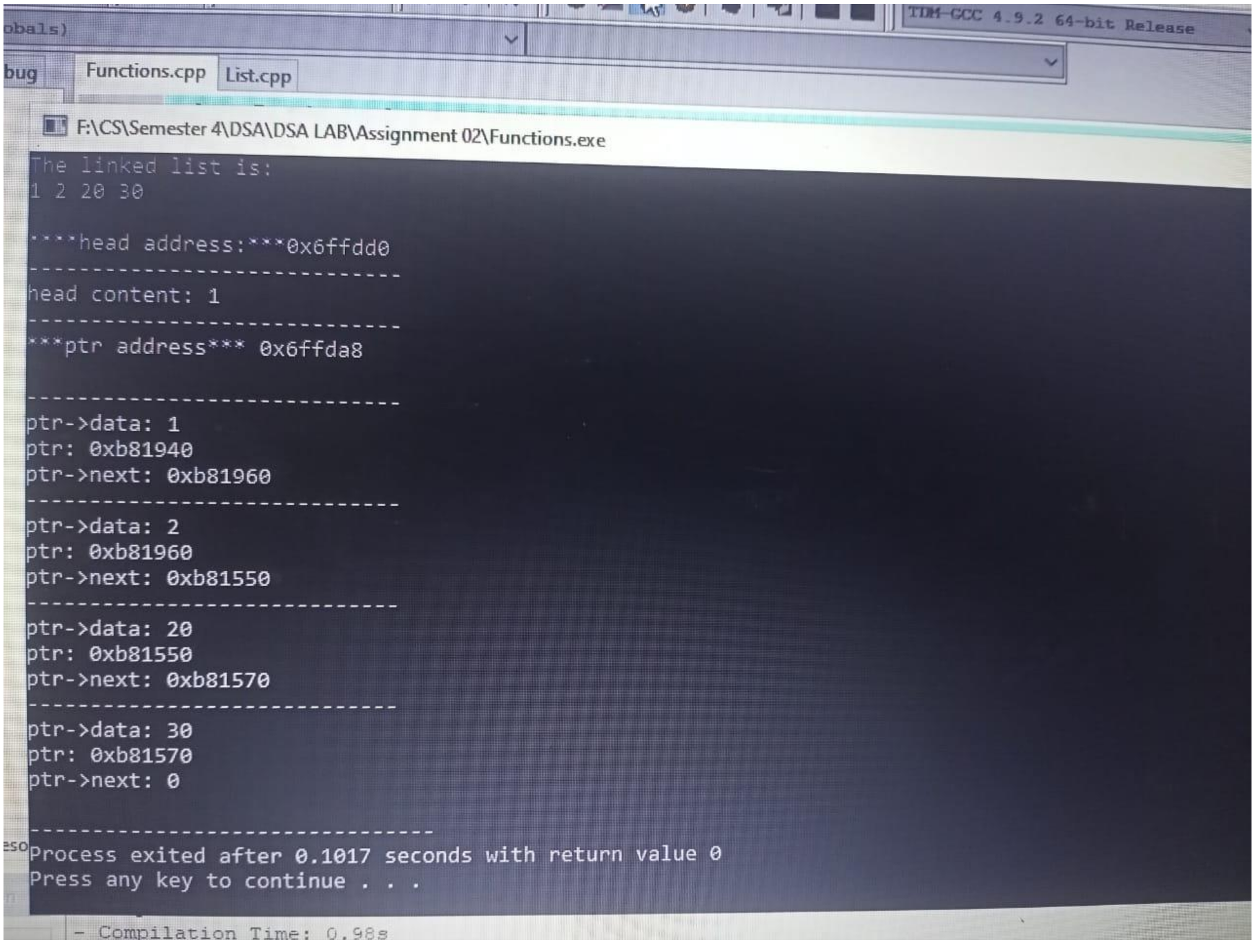
```
Operations on List..

1. Insertion
2. Deletion
3. Display
4. Seek
5. Exit
Enter your choice: 1

1. Insertion at the beginning
2. Insertion at the end
3. Enter your choice:1

Enter the value to insert: 1

 Inserted successfully at the beginning . .
The items present in the list are : 1
Press any key to continue . . .

Operations on List..

1. Insertion
2. Deletion
3. Display
4. Seek
5. Exit
Enter your choice: _
```

<u>ANSWER:</u>

```
#include <iostream>



using namespace std;



struct Node {

    int data;

    Node* next;

    Node(int val) : data(val), next(NULL) {}

};



void displayLinkedList(Node* head) {

    cout << "The linked list is:" << endl;
```

```
Node* ptr = head;


while (ptr != NULL) {
    cout << ptr->data << " ";
    ptr = ptr->next;
}
cout << endl<<endl;


cout << "****head address:***";
cout << &head << endl;
cout<<"-----------------------------\n";


cout << "head content: " << head->data << endl;
cout <<"-----------------------------\n";


ptr = head;
cout << "***ptr address*** ";
cout << &ptr << endl<<endl;



while (ptr != NULL) {
    cout <<"-----------------------------\n";
```

```
        cout << "ptr->data: " << ptr->data << endl;
        cout << "ptr: " << ptr << endl;
        cout << "ptr->next: " << ptr->next << endl;
        ptr = ptr->next;

    }
}

int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(20);
    head->next->next->next = new Node(30);

    displayLinkedList(head);

    // Free the allocated memory
    Node* current = head;
    while (current != NULL) {
        Node* temp = current;
        current = current->next;
        delete temp;
    }
```



obals)
bug
Functions.cpp
List.cpp
TDM-GCC 4.9.2 64-bit Release
F:\CS\Semester 4\DSA\DSA LAB\Assignment 02\Functions.exe

```
The linked list is:
1 2 20 30

****head address:***0x6ffdd0
----------------------------
head content: 1
----------------------------
***ptr address*** 0x6ffda8

----------------------------
ptr->data: 1
ptr: 0xb81940
ptr->next: 0xb81960
----------------------------
ptr->data: 2
ptr: 0xb81960
ptr->next: 0xb81550
----------------------------
ptr->data: 20
ptr: 0xb81550
ptr->next: 0xb81570
----------------------------
ptr->data: 30
ptr: 0xb81570
ptr->next: 0

--------------------------------
Process exited after 0.1017 seconds with return value 0
Press any key to continue . . .
```

- Compilation Time: 0.98s

```
    return 0;

}
```

<u>**You can get help from the below link:**</u>

https://github.com/programming-debug/Data-Structure-Lab/blob/main/Lab3/single-link%20list.cpp

In this Word file, you should place the code and its output screenshot.

After completing the activities, Upload the final pdf and cpp code files to the "**DSA_Lab"** repository.