Name

Zeeshan Ali

Roll no

SU92-BSITM-F22-019

Assignment

Data Structure & Algorithm

Submitted to

Zeeshan Mubeen

Data structure & Algorithm

Assignment # 02.

Questions :-

Singly Linked List

Tasks

(01)

Palindrome :-

```
class Node{
public:
    int data;
   Node *next;
 Node(int value){
      data = value;
      next = NULL;
      }
 class LinkedList{
 public:
  Node * Head;
 LinkedList( ) : Head(NULL){ }
 void Insert(int data){
  Node *newnode = new Node(data);
  if(!Head){
     Head = newnode;
  }
 else{
     Node * current = Head;
```

```
while (current -> next){
    current = current -> next;
    }
    current -> next = newnode;
    }
}
void Display(){
Node * current = head;
  while (current){
    cout << current -> data;
  if (current -> next){
     cout << " -> ";
     }
     current = current -> next;
     }
     cout << endl;
  bool IsPalindrome ( ){
if ( !Head || !Head -> next){
     return true;
  }
Node * slow, * fast = head;
while( fast && fast -> next){
    slow = slow -> next;
    fast = fast -> next -> next;
    }
  Node * prev = NULL;
  Node * current = slow;
 while (current){
    Node * temp = current -> next;
    current -> next = prev;
     prev = current;
     current = temp;
```

```
Node *firsthalf = head;
Node * secondhalf = prev;
 while (secondhalf){
    if (firsthalf -> data != secondhalf -> data){
      return false;
      }
      firsthalf = firsthalf -> next;
      secondhalf = secondhalf -> next;
      }return true;
      }};
 int main (){
 LinkedList list;
list.insert(1), (insert(2)), insert(3), insert(2),
insert(1);
   cout << " Linked List :" << endl;
    list. display();
    if (list.palindrome()) {
    cout << "It is palindrome" << endl;
    }
    else{
    cout << " It is not palindrome" << endl;
    }
```

—(02)—

Even case : odd case

```
class Node{
    int data;
    Node* next;
Node(int value){ data = value; next = NULL; }
```

{permanent}

```
class LinkedList{
Node * head;
LinkedList(); head(NULL){ }
  void Insert(int data){
  Node * newnode = new Node(data);
    if(! head){
     head = newnode;
    }
    else {
       Node * current = head;
     while(current->next){
       current = current->next;
      }
     current->next = newnode;
      }
    void Display(){
    Node * current = head;
     while (current){
     cout << current->data;
     if (current->data){
       cout << "->";
      }current = current->next;
      }
      cout << endl;
    Node * middleNode(Node * Head){
      if(! Head){
         return NULL;
      }
     Node *slow = head;
     Node *fast = head;
    while(fast && fast->next){
     slow = slow->next;
```

{permanent}

{permanent}

```
        fast = fast -> next -> next;
    }
    return slow;
  }
};
int main(){
  LinkedList list1;
list1.insert(1), insert(2), insert(3), insert(2),
  insert(1);
  cout << "Linked list" <<endl;
  list1.display();
 Node * middle1 = list1.middleNode(list1.head);
  if(middle1){
    cout << "Output : Odd case" <<endl;
  cout << "middle1 -> data << "is the middle
  node" <<endl; }
   else{
  cout << "Output : Empty list" <<endl;
   }
```

Question no (03)

Display Middle Node with Reverse

```
class Node { }; — permanent —
class LinkedList { } — permanent —

  void Display(){ } permanent —
  void * MiddleNode(Node *Head){
    if(!Head){
```

```
return NULL;
}
Node *slow = head;
Node * fast = head;
while (fast && fast -> next){
slow = slow -> next;
fast = fast -> next -> next;
}
return slow;}
void makeMiddleHead(){
Node * middle = middleNode(head);
if (middle){
Node * newhead = middle -> next;
middle -> next = NULL;
newhead = reverseList(newhead);
middle -> next = head;
head = newHead;
}}
Node *reverseList(Node *head){
Node * prev = NULL;
Node * current = head;
while(current){
Node * next_node = current -> next;
current -> next = prev;
prev = current;
current = next_node;
}
return prev;
int main(){
LinkedList List1;
List1.insert(1), insert(2), insert(3), insert(2),
insert(1);
```

```
cout << " original linked list 1 : " ;
   list1.display();
   list1.makeMiddleHead();
   cout << "Modified linked list" << endl;
```

(04)

```
class Node { } ;          — permanent —
class LinkedList { } ; — permanent —
void Display( ) ;         — permanent —
  void SortList ( ){
  if (!Head || !Head -> next){
        return;
  }
  Node * current = 'head;
    while( current){
    Node * minnode = current;
    Node * temp = current;
     while ( temp ) {
    if ( temp -> data < minnode -> data){
        minnode = temp;
    }
    temp = temp -> next;
    }
   int tempData = current -> data;
    current -> data = minnode -> data;
    minNode -> data = tempData;
      current = current -> next;
  }
  int main ( ) {
```

```
    LinkedList list1;
list1.insert(1), insert(2), insert(3), inert(3),
insert(3), insert(8);
 cout << " original Linked list ";
    list1.display();
    list1.sortList();
   cout << " sorted List " << endl;
    list1.display();
```

(05)

class Node { } ; — permanent —

class LinkedList{ }; — permanent —

void Display(); — permanent —

```
void RemoveDuplicate(){
  Node * current = head;
  while( current && current->next){
  if (current->data == current->next->data)
      {
       Node * temp = current->next;
       current->next = current->next->next;
        delete temp;
          }else{
          current = current->next;
            }
            };
    int main(){
    LinkedList list1;
```

```
List1.insert(1), insert(2), insert(3), insert(3),
  insert(8);
  cout<<"Original Linked List"<<endl;
  List1.display();
  List1.RemoveDuplicate();
  cout<<"Modified Linked List"<<endl;
    List1.display();
```

(06)

```
class Node{};  — permanent —
class LinkedList{};  — permanent —
void Display();  — permanent —
  void swapNodes(){
  Node *current =head;
   while(current && current->next){
    int temp = current->data;
   current->data = current->next->next;
   current->next->data = temp;
     current = current->next->next;
        }
      };
  int main(){
  LinkedList List1;
List1.insert(1), insert(2), insert(3), insert(2),
  insert(0);
```

```
cout << "original link list :" << endl;
    list1.display();
list1.swapNodes();
cout << "modified link list" << endl;
  list1.display();
```

—(07)—

```
class Node { };  — permanent —
class LinkedList { };  — permanent —
void Display();  — permanent —
void Evenodd() {
if (!head || !head->next) {
    return;
  }
LinkedList evenList;
LinkedList oddlist;
Node * current = head;
 while( current) {
if( current -> data % 2 == 0) {
  evenlist.insert(current->data);
  } else {
  oddlist.insert(current->data);
    }
    current = current->next;
  }
```

```
head = evenlist.head;
Node *evencurrent = evenlist.head;
while(evencurrent->next){
evencurrent = evencurrent->next;
}
evencurrent->next = oddlist.head;
}};
int main(){
LinkedList list;
list.insert(1), insert(2), insert(3), insert(2),
insert(0);
cout<<"original linked list"<<endl;
list.Display();
list.display()
cout<<"Modified linked list"<<endl;
list.display();
```

(08)

class Node { }; — permanent —

class LinkedList { }; — permanent —

void Display(); — permanent —

```
void reverseList(){
  if(!head || !head->next){
      return;
    }
  Node *prev = NULL;
  Node *current = head;
  Node *next_node = NULL;
  while(current){
  next_node = current->next;
  current->next = prev;
    prev = current;
    current = next_node;}
     head = prev;
   }};
int main(){
Linked List list;
list.insert(1), insert(2), insert(3), insert(9),
insert(9), insert(0);
  cout << "Original Linked List" << endl;
  list.Display();
  list.ReverseList();
  cout << "Reversed List" << endl;
  list.display();
  return 0;
   }
```

Doubly Linked List

(01)

```
class Node {
public:
  int data;
  Node * prev;
  Node * next;
Node(int value) : data(value), prev(NULL),
  next(NULL) { }
  };
class DoublyLinkedList{
public:
  Node * head;
DoublyLinkedList( ): head(NULL) { }
  void  insert (int data){
  Node * newnode = new Node(data);
  if (!head) {
    head = new_node;
    }
  else if (data <= head -> data){
  newnode -> next = head;
  head -> prev = new_node;
   head = new_node;
    }
```

{permanent}

{permanent}

```
else {
    Node * current = head;
while ( current -> next && current -> next -> data
    < data) {
    current = current -> next;
    }
    new_node -> next = current -> next;
    new_node -> prev = current;
    if ( current -> next) {
     current -> next -> prev = new_node;
        }
    current -> next = new_node;
        }}
void Display() {
    Node * current = head;
    while ( current) {
    cout << " current -> data";
     if ( current -> next) {
        cout << " -> ";
        }
        current = current -> next;
    }
    cout << endl;
}
};
```

permanent

```
int main (){
DoublyLinkedList list;
list insert(1), insert(5), insert(8), insert(10);
  cout << " Odd list";
  list.display();
int newData;
  cin >> newData;
list.insert(newData);
  cout << New list";
  list.display ();
  return 0;
  }
```

(02)

```
class Node {};  — permanent —
class DoublyLinkedList { };  — permanent —
void Display ();  — permanent —
  void ReverseNodes (int n){
    if(!head || n <= 1) {
        return;
      }
      Node * current = head;
      Node * prevnode = NULL;
```

```
for(int i=0; i<n-1 && current; i++){
    prevNode = current = head;
    Node *
    current = current->next;
  }
  if(!current){
    return;
  }
Node *nextNode = NULL;
Node * tailNode = prevNode;
for(int i=0; i<n; i++){
nextnode = current->next;
current->next = prevNode;
current->prev = nextNode;
 prevNode = current;
  current = NextNode;
}
if(tailNode){
  tailNode->next = prevNode;
  prevNode -> prev = tailNode;
}
else{
 head = prevNode;
 prevNode->prev = NULL;
}}};
```

```
int main(){

    DoublyLinkedList list;
list.insert(1), insert(5), insert(8), insert(10),
insert(2), insert(4), insert(8), insert(a),
 insert(11), insert(34);
    cout << " Old list";
   list.display();
   cout << " Enter n " << endl;
     int n;
   cin >> n;
   cout << " New list: ";
    list.Display;
      return 0;
    }
```

Circular Linked List

Task 01

```
class Node {
    public:
    int Data;
    Node * next;
Node(int value) : data(value), next(NULL) { }
};
class CircularLinkedList {
    public:
    Node * head;
CircularLinkedList() : head(NULL) { }
void insert(int data) {
    Node * new_node = new Node(data);
    if(!head) {
        head = new_node;
        head->next = head;
    }
    else {
    Node *current = head;
    while(current->next != head) {
        current = current->next;
    }
```

```
current->next = new_node;
new_node->next = head;
}}
void Display(){
Node * current = head;
do {
cout << current->data;
if(current->next != head){
cout << "->";
}
current = current->next;
while(current != head){
cout << endl;
}
bool IsCircular(){
if(!head){
return false;
}
Node * current = head;
while(current->next != head){
if(current->next == NULL){
return false;
}
current = current->next;
}
```

```
    return true;
}
void MakeCircular(){
  if(!isCircular()){
  Node * current = head;
  while(current->next != NULL){
        current = current -> next;
    }
    current -> next = head;
  }
void addNodeToCircularList(int data){
if (isCircularList()){
Node * new_node = new Node (data);
 Node * last = head;
  while(last -> next != head){
    last = last -> next;
  }
  last -> next = new_node;
  new_node -> next = head;
  }}};
```

```
int main (){
CircularLinkedList List;
List.insert(1), insert(5), insert(8), insert(10),
insert(2), insert(4), insert(8), insert(9),
  insert(11), insert(34);

  cout << " List";
  List.display();
  if (List.isCircularList()) {
  cout << " Output: List is circular "<<endl;
    }
  else{
    cout << " Output: List is singly
  Linked List " <<endl;
    int newData;
  cout << " Enter new node"
  cin >> newData;
  List.addNodetoCircularList (newData);
  cout << " List with the new node";
  List.display;
```

Special Task

```
class Employee{
public:
  int, Id, age; Salary;
  string Name;
   Employee * Next;
Employee(int EmpID, const string & empName, int empAge,
  double empSalary): Id(empName), emp,
  Id(empID), name(empName), age(empAge),
  salary(empSalary), next(NULL){ }
 class LinkedList {
 private:
   Employee * head;
 public:
  LinkedList(): head(NULL){ }
 void Insert(int Id, const string & name, int age,
  double salary){
   Employee * newEmployee = new
       Employee(id, name, age, salary);
   if(!head || name < head->name){
    newEmployee->next = head;
    head = newEmployee;
   }
```

```
else {
    Employee * current = head;
    while (current -> next && name >
           current -> next -> name) {
        current = current -> next;
    }
    newEmployee -> next = current -> next;
    current -> next = newEmployee;
void Display() {
cout << "ID" << current -> ID <<
"Name" << current -> name << "Age" <<
current -> age << "Salary" << current ->
salary << endl;
    }
}
Employee * findmaxsalaryEmployee() {
if (!head) {
    return NULL;
}
Employee * current = head;
Employee * maxsalaryEmployee = head;
double maxsalary = head -> salary;
```

```
while (current) {
if ( current->salary > maxsalary) {
 maxsalary = current->salary;
 maxsalaryEmployee = current;
      }
 current = current->next;
     return maxsalaryEmployee;
      }
void  DeleteEmployee(int ID){
    if (! head){
     return;
    }
  if(head-> id == id){
Employee * temp = head;
   head = head-> next;
     delete temp;
      return;
     }
  Employee * current = head;
while(current -> next && current->next-> id !=
     id){
     current = current -> next;
      }
```

```
if(current->next){
Employee *temp = current->next;
current->next = current->next->next;
delete temp;
}}
void updateEmployee(int ID, const
string &newName, int newAge, double newSalary){
Employee *current = head;
while(current){
if(current->ID == ID){
current->name = newName;
current->age = newAge;
current->salary = newSalary;
break;
}
current = current->next;
}};
int main(){
```

```
LinkedList employeeList;
employeeList.Insert(101, "Ali", 30, 60000);
employeeList.Insert(102, "Raz", 25, 55000),
(103, "Ahmed", 30, 600000),
(104, "Churram", 35, 100000);
cout << "Employee In Alphabatical order"
<<endl;
employeeList.display();

Employee * maxsalaryEmployee = employeeList.
findMaxSalaryEmployee();
if(maxSalaryEmployee){
cout << "Employee with maximum
salary" <<endl;
cout << "Every Detail about that
particular Employee"
maxSalaryEmployee -> ID, -> name,
-> age, -> salary <<endl;
employeeList.deleteEmployee(102);
employeeList.display();
employeeList.updateEmployee(101, "Irfan"
,31, 65000);
employeeList.display();
```