Module 3 Linked Lists

24. Code for the creation of new a node in a linked list  
void createnode(int value) { node \*temp=new node; temp->data=value; temp->next=NULL; if(head==NULL) { head=temp; tail=temp; temp=NULL; } else { tail->next=temp; tail=temp; } }

25. Code for displaying nodes of linked list void display() { node \*temp=new node; temp=head; while(temp!=NULL) { cout<<temp->data<<"\t"; temp=temp->next; } }

26. Code to insert the node at the start of the linked list void insert\_start(int value) { node \*temp=new node; temp->data=value; temp->next=head; head=temp; }

27. Code to insert the node at the particular position in the linked list  
void insert\_position(int pos, int value) { node \*pre=new node; node \*cur=new node;  
node \*temp=new node; cur=head; for(int i=1;i<pos;i++) { pre=cur; cur=cur->next; } temp->data=value; pre->next=temp; temp->next=cur; }

28. Code to delete the first node from the linked list  
void delete\_first() { node \*temp=new node; temp=head; head=head->next; delete temp; }

29. Code to delete the last node from the linked list  
void delete\_last() { node \*current=new node; node \*previous=new node; current=head; while(current->next!=NULL) { previous=current; current=current->next; } tail=previous; previous->next=NULL; delete current; }

30. Code to delete the node from a particular position from a linked list  
void delete\_position(int pos) { node \*current=new node; node \*previous=new node; current=head; for(int i=1;i<pos;i++) { previous=current; current=current->next; } previous->next=current->next; }

31. Program to calculate factorial of a given number using recursion in ‘C’ #include<iostream.h> #include<conio.h> void main() { int n,fact; int rec(int); clrscr(); cout<<"Enter the number:->"; cin>>n; fact=rec(n); cout<<endl<<"Factorial Result is:: "<<fact<<endl; getch(); } rec(int x) {  
int f; if(x==1) return(x); else {  
f=x\*rec(x-1); return(f); }  
}

32. Program to compute the nth number of Fibonacci series using Recursion. Fibonacci series is given by 0, 1, 1, 2, 3, 5, 8, 13, 21, ---- #include<iostream.h> #include<conio.h> int recfib(int n) { if(n==1)  
return 0; else if(n<3)  
return 1; else return (recfib(n-1)+recfib(n-2));  
} void main() { clrscr(); int n; cout<<"Fibonacci series generation using recursion"; cout<<"Enter the limit"; cin>>n; for(int i=1;i<=n;i++) cout<<recfib(i)<<" ";  
getch(); }

33. Program to implement Tower of Hanoi problem #include<iostream.h> void move(int n,char \*s,char \*i,char \*d) { if(n>0) { move(n-1,s,d,i); cout<<"disk "<<n<<" is moved from "<<s<<" to "<<d<<endl; move(n-1,i,s,d); } } void main() { cout<<"Enter the no. of disks "; int n; cin>>n; move(n,"sourcetower","intermediatetower","destination tower"); }

34. Code to PUSH item in Stack using Linked List  
push() { int value; struct node \*ptr; cout<<"\nPUSH Operationn"; cout<<"Enter a number to insert: "; cin>>value; ptr=new node; ptr->data=value; ptr->next=NULL; if(top!=NULL) ptr->next=top; top=ptr; cout<<"\nNew item is inserted to the stack!!!"; }

35. Code to POP item from stack using Linked List  
pop() { struct node \*temp; if(top==NULL) { cout<<"\nThe stack is empty!!!"; } temp=top; top=top->next; cout<<"\nPOP Operation........nPoped value is "<<temp->data;  
delete temp; }

36. Program to insert item in a Queue using Linked List #include<iostream> using namespace std; struct node { int data; node \*next; }\*front = NULL,\*rear = NULL,\*p = NULL,\*np = NULL; void push(int x) { np = new node; np->data = x; np->next = NULL; if(front == NULL) { front = rear = np; rear->next = NULL; } else { rear->next = np; rear = np; rear->next = NULL; } } int main() { int n,c = 0,x; cout<<"Enter the number of values to be pushed into queue"; cin>>n; while (c < n) { cout<<"Enter the value to be entered into queue"; cin>>x; push(x); c++; } }

37. Program to delete item from a Queue using Linked List #include<iostream> using namespace std; struct node { int data; node \*next; }\*front = NULL,\*rear = NULL,\*p = NULL,\*np = NULL;  
int remove() { int x; if(front == NULL) { cout<<"empty queue"; } else { p = front; x = p->data; front = front->next; delete(p); return(x); } }  
int main() { cout<<"Removed Values"; while(true) { if (front != NULL) cout<<remove()<<endl; else break; } return 0; }

38. Code to find element in the Binary Search Tree  
struct node { int info; struct node \*left; struct node \*right; }\*root;  
void BST::find(int item, node \*\*par, node \*\*loc) { node \*ptr, \*ptrsave; if (root == NULL) { \*loc = NULL; \*par = NULL; return; } if (item == root->info) { \*loc = root; \*par = NULL; return;  
} if (item < root->info) ptr = root->left; else ptr = root->right; ptrsave = root; while (ptr != NULL) { if (item == ptr->info) { \*loc = ptr; \*par = ptrsave; return; } ptrsave = ptr; if (item < ptr->info) ptr = ptr->left; else ptr = ptr->right; } \*loc = NULL; \*par = ptrsave; }

39. Code to insert element into Binary Search Tree  
struct node { int info; struct node \*left; struct node \*right; }\*root;  
void BST::insert(node \*tree, node \*newnode) { if (root == NULL) { root = new node; root->info = newnode->info; root->left = NULL; root->right = NULL; cout<<"Root Node is Added"<<endl; return; } if (tree->info == newnode->info) { cout<<"Element already in the tree"<<endl; return; } if (tree->info > newnode->info) { if (tree->left != NULL) {  
insert(tree->left, newnode); } else { tree->left = newnode; (tree->left)->left = NULL; (tree->left)->right = NULL; cout<<"Node Added To Left"<<endl; return; } } else { if (tree->right != NULL) { insert(tree->right, newnode); } else { tree->right = newnode; (tree->right)->left = NULL; (tree->right)->right = NULL; cout<<"Node Added To Right"<<endl; return; } } }

40. Code for Preorder Traversal of Binary Search Tree  
void BST::preorder(node \*ptr) { if (root == NULL) { cout<<"Tree is empty"<<endl; return; } if (ptr != NULL) { cout<<ptr->info<<" "; preorder(ptr->left); preorder(ptr->right); } }

41. Code for Inorder Traversal of Binary Search Tree  
void BST::inorder(node \*ptr) { if (root == NULL) { cout<<"Tree is empty"<<endl; return;  
} if (ptr != NULL) { inorder(ptr->left); cout<<ptr->info<<" "; inorder(ptr->right); } }

42. Code for Postorder Traversal of Binary Search Tree  
void BST::postorder(node \*ptr) { if (root == NULL) { cout<<"Tree is empty"<<endl; return; } if (ptr != NULL) { postorder(ptr->left); postorder(ptr->right); cout<<ptr->info<<" "; } }

43. Code for Displaying Binary Search Tree Structure  
void BST::display(node \*ptr, int level) { int i; if (ptr != NULL) { display(ptr->right, level+1); cout<<endl; if (ptr == root) cout<<"Root->: "; else { for (i = 0;i < level;i++) cout<<" "; } cout<<ptr->info; display(ptr->left, level+1); } }

44. Code to create Circular Linked List  
struct node { int info; struct node \*next; }\*last;  
void circular\_llist::create\_node(int value) { struct node \*temp; temp = new(struct node); temp->info = value; if (last == NULL) { last = temp; temp->next = last; } else { temp->next = last->next; last->next = temp; last = temp; } }

45. Code to insert element in a Circular Linked List  
struct node { int info; struct node \*next; }\*last;  
void circular\_llist::add\_begin(int value) { if (last == NULL) { cout<<"First Create the list."<<endl; return; } struct node \*temp; temp = new(struct node); temp->info = value; temp->next = last->next; last->next = temp; }

46. Code to insert element at a particular place in a Circular Linked List  
struct node { int info; struct node \*next; }\*last;  
void circular\_llist::add\_after(int value, int pos) { if (last == NULL) { cout<<"First Create the list."<<endl; return; }  
struct node \*temp, \*s; s = last->next; for (int i = 0;i < pos-1;i++) { s = s->next; if (s == last->next) { cout<<"There are less than "; cout<<pos<<" in the list"<<endl; return; } } temp = new(struct node); temp->next = s->next; temp->info = value; s->next = temp; /\*Element inserted at the end\*/ if (s == last) { last=temp; } }

47. Code to delete element from a Circular Linked List struct node { int info; struct node \*next; }\*last;  
void circular\_llist::delete\_element(int value) { struct node \*temp, \*s; s = last->next; /\* If List has only one element\*/ if (last->next == last && last->info == value) { temp = last; last = NULL; free(temp); return; } if (s->info == value) /\*First Element Deletion\*/ { temp = s; last->next = s->next; free(temp); return; } while (s->next != last) { /\*Deletion of Element in between\*/  
if (s->next->info == value) { temp = s->next; s->next = temp->next; free(temp); cout<<"Element "<<value; cout<<" deleted from the list"<<endl; return; } s = s->next; } /\*Deletion of last element\*/ if (s->next->info == value) { temp = s->next; s->next = last->next; free(temp); last = s; return; } cout<<"Element "<<value<<" not found in the list"<<endl; }

48. Code to search element in a Circular Linked List struct node { int info; struct node \*next; }\*last;  
void circular\_llist::search\_element(int value) { struct node \*s; int counter = 0; s = last->next; while (s != last) { counter++; if (s->info == value) { cout<<"Element "<<value; cout<<" found at position "<<counter<<endl; return; } s = s->next; } if (s->info == value) { counter++; cout<<"Element "<<value; cout<<" found at position "<<counter<<endl; return;  
} cout<<"Element "<<value<<" not found in the list"<<endl; }

49. Code to display Circular Linked List struct node { int info; struct node \*next; }\*last;  
void circular\_llist::display\_list() { struct node \*s; if (last == NULL) { cout<<"List is empty, nothing to display"<<endl; return; } s = last->next; cout<<"Circular Link List: "<<endl; while (s != last) { cout<<s->info<<"->"; s = s->next; } cout<<s->info<<endl; }

50. Code to update Circular Linked List struct node { int info; struct node \*next; }\*last;  
void circular\_llist::update() { int value, pos, i; if (last == NULL) { cout<<"List is empty, nothing to update"<<endl; return; } cout<<"Enter the node position to be updated: "; cin>>pos; cout<<"Enter the new value: "; cin>>value; struct node \*s; s = last->next; for (i = 0;i < pos - 1;i++) { if (s == last)  
{ cout<<"There are less than "<<pos<<" elements."; cout<<endl; return; } s = s->next; } s->info = value; cout<<"Node Updated"<<endl; }

51. Code to sort Circular Linked List struct node { int info; struct node \*next; }\*last;  
void circular\_llist::sort() { struct node \*s, \*ptr; int temp; if (last == NULL) { cout<<"List is empty, nothing to sort"<<endl; return; } s = last->next; while (s != last) { ptr = s->next; while (ptr != last->next) { if (ptr != last->next) { if (s->info > ptr->info) { temp = s->info; s->info = ptr->info; ptr->info = temp; } } else break; ptr = ptr->next; } s = s->next; } }

52. Code to create a node in a singly linked list  
node \*single\_llist::create\_node(int value) { struct node \*temp, \*s; temp = new(struct node); if (temp == NULL) { cout<<"Memory not allocated "<<endl; return 0; } else { temp->info = value; temp->next = NULL; return temp; } }

53. Code to insert element at the beginning of a singly linked list  
void single\_llist::insert\_begin() { int value; cout<<"Enter the value to be inserted: "; cin>>value; struct node \*temp, \*p; temp = create\_node(value); if (start == NULL) { start = temp; start->next = NULL; } else { p = start; start = temp; start->next = p; } cout<<"Element Inserted at beginning"<<endl; }

54. Code to insert element at the last position in a singly linked list  
void single\_llist::insert\_last() { int value; cout<<"Enter the value to be inserted: "; cin>>value; struct node \*temp, \*s; temp = create\_node(value); s = start;  
while (s->next != NULL) { s = s->next; } temp->next = NULL; s->next = temp; cout<<"Element Inserted at last"<<endl; }

55. Code to insert element at a given position in a singly linked list  
void single\_llist::insert\_pos() { int value, pos, counter = 0; cout<<"Enter the value to be inserted: "; cin>>value; struct node \*temp, \*s, \*ptr; temp = create\_node(value); cout<<"Enter the postion at which node to be inserted: "; cin>>pos; int i; s = start; while (s != NULL) { s = s->next; counter++; } if (pos == 1) { if (start == NULL) { start = temp; start->next = NULL; } else { ptr = start; start = temp; start->next = ptr; } } else if (pos > 1 && pos <= counter) { s = start; for (i = 1; i < pos; i++) { ptr = s; s = s->next; } ptr->next = temp; temp->next = s;  
} else { cout<<"Positon out of range"<<endl; } }

56. Code to sort a singly linked list  
void single\_llist::sort() { struct node \*ptr, \*s; int value; if (start == NULL) { cout<<"The List is empty"<<endl; return; } ptr = start; while (ptr != NULL) { for (s = ptr->next;s !=NULL;s = s->next) { if (ptr->info > s->info) { value = ptr->info; ptr->info = s->info; s->info = value; } } ptr = ptr->next; } }

57. Code to delete node at a given position in a singly linked list  
void single\_llist::delete\_pos() { int pos, i, counter = 0; if (start == NULL) { cout<<"List is empty"<<endl; return; } cout<<"Enter the position of value to be deleted: "; cin>>pos; struct node \*s, \*ptr; s = start; if (pos == 1) { start = s->next; }  
else { while (s != NULL) { s = s->next; counter++; } if (pos > 0 && pos <= counter) { s = start; for (i = 1;i < pos;i++) { ptr = s; s = s->next; } ptr->next = s->next; } else { cout<<"Position out of range"<<endl; } free(s); cout<<"Element Deleted"<<endl; } }

58. Code to update a given node in a singly linked list  
void single\_llist::update() { int value, pos, i; if (start == NULL) { cout<<"List is empty"<<endl; return; } cout<<"Enter the node position to be updated: "; cin>>pos; cout<<"Enter the new value: "; cin>>value; struct node \*s, \*ptr; s = start; if (pos == 1) start->info = value; else { for (i = 0;i < pos - 1;i++) { if (s == NULL) { cout<<"There are less than "<<pos<<" elements"; return;  
} s = s->next; } s->info = value; } cout<<"Node Updated"<<endl; }

59. Code to search an element in a singly linked list  
void single\_llist::search() { int value, pos = 0; bool flag = false; if (start == NULL) { cout<<"List is empty"<<endl; return; } cout<<"Enter the value to be searched: "; cin>>value; struct node \*s; s = start; while (s != NULL) { pos++; if (s->info == value) { flag = true; cout<<"Element "<<value<<" is found at position "<<pos<<endl; } s = s->next; } if (!flag) cout<<"Element "<<value<<" not found in the list"<<endl; }

60. Code to reverse a singly linked list  
void single\_llist::reverse() { struct node \*ptr1, \*ptr2, \*ptr3; if (start == NULL) { cout<<"List is empty"<<endl; return; } if (start->next == NULL) return; ptr1 = start; ptr2 = ptr1->next; ptr3 = ptr2->next; ptr1->next = NULL;  
ptr2->next = ptr1; while (ptr3 != NULL) { ptr1 = ptr2; ptr2 = ptr3; ptr3 = ptr3->next; ptr2->next = ptr1; } start = ptr2; }

61. Code to display the elements of singly linked list  
void single\_llist::display() { struct node \*temp; if (start == NULL) { cout<<"The List is Empty"<<endl; return; } temp = start; cout<<"Elements of list are: "<<endl; while (temp != NULL) { cout<<temp->info<<"->"; temp = temp->next; } cout<<"NULL"<<endl; }

94. Code for Deletion of element from the Doubly Linked List void double\_llist::delete\_element(int value) { struct node \*tmp, \*q; /\*first element deletion\*/ if (start->info == value) { tmp = start; start = start->next; start->prev = NULL; cout<<"Element Deleted"<<endl; free(tmp); return; } q = start; while (q->next->next != NULL) { /\*Element deleted in between\*/ if (q->next->info == value) { tmp = q->next; q->next = tmp->next; tmp->next->prev = q; cout<<"Element Deleted"<<endl; free(tmp);  
return; } q = q->next; } /\*last element deleted\*/ if (q->next->info == value) { tmp = q->next; free(tmp); q->next = NULL; cout<<"Element Deleted"<<endl; return; } cout<<"Element "<<value<<" not found"<<endl; }

95. Code to insert at a particular position in a Doubly Linked List void double\_llist::add\_after(int value, int pos) { if (start == NULL) { cout<<"First Create the list."<<endl; return; } struct node \*tmp, \*q; int i; q = start; for (i = 0;i < pos - 1;i++) { q = q->next; if (q == NULL) { cout<<"There are less than "; cout<<pos<<" elements."<<endl; return; } } tmp = new(struct node); tmp->info = value; if (q->next == NULL) { q->next = tmp; tmp->next = NULL; tmp->prev = q; } else { tmp->next = q->next; tmp->next->prev = tmp; q->next = tmp;  
tmp->prev = q; } cout<<"Element Inserted"<<endl; }

100. Code for insertion at the beginning in the Doubly Linked List  
void double\_llist::add\_begin(int value) { if (start == NULL) { cout<<"First Create the list."<<endl; return; } struct node \*temp; temp = new(struct node); temp->prev = NULL; temp->info = value; temp->next = start; start->prev = temp; start = temp; cout<<"Element Inserted"<<endl; }