Bubble sort
Code:

```
#include<iostream>
using namespace std;
void bubble(int ar[],int n)
{
    int temp=0;
        for(int i=0;i<n-1;i++)
        {
            for(int j=0;j<n-i-1;j++)
            {
                if(ar[j]>ar[j+1])
                {
                    temp=ar[j];
                    ar[j]=ar[j+1];
                    ar[j+1]=temp;
                }
            }
        }
}
void pri(int ar[],int n)
{
    cout << "The numbers are: "<<endl;
    for (int j = 0; j < n; ++j)
    {
        cout << ar[j] << " ";
    }
}
int main()
{
    int n,ar[100];
    cout<<"Enter the size of array: "<<endl;
    cin>>n;
    cout << "Enter "<<n<<" numbers: " << endl;
    for (int i = 0; i<n; ++i)
    {
        cin >> ar[i];
    }
    bubble(ar,n);
    pri(ar,n);
    return 0;
}
```

Output:

```
Enter the size of array: 5
Enter 5 numbers:
44 66 88 77 11
The numbers are:
11 44 66 77 88
```

Insertion sort

```
#include<iostream>
using namespace std;
void insertion(int ar[],int n)
{
    int temp,j;
    for(int i=1;i<n;i++)
    {
        temp = ar[i];
        j=i-1;
        while(j>=0 && temp<ar[j])
        {
            ar[j+1]=ar[j];
            j-=1; //j=j-1;
        }
        ar[j+1]=temp;
    }
}
void pri(int ar[],int n)
{
    cout << "The numbers are: "<<endl;
    for (int j = 0; j < n; ++j)
    {
        cout << ar[j] << " ";
    }
}

int main()
{
    int n;
    cout<<"INSERTION SORT: "<<endl;
    cout<<"Enter the size of array: "<<endl;
    cin>>n;
    int ar[n];
    cout << "Enter "<<n<<" numbers: " << endl;
    for (int i = 0; i<n; ++i)
    {
        cin >> ar[i];
    }
    insertion(ar,n);
    pri(ar,n);
    return 0;
}
```

Output

```
Enter the size of array :5

Enter 5 numbers:
```

2

412

5126

0

136

The numbers are: 0 2 136 412 5126

SELECTION SORT

Code:

```
#include<iostream>
#include<conio.h>

using namespace std;

void selection(int ar[],int n)
{
    int temp,swap;
    for (int i=0;i<n-1;i++)
    {
        temp=i;
        for(int j=i+1;j<n;j++)
        {
            if(ar[j]<ar[temp])
            temp=j;
        }
        swap=ar[temp];
        ar[temp]=ar[i];
        ar[i]=swap;
    }
}
void pri(int ar[],int n)
{
    cout << "The numbers are: "<<endl;
    for (int j = 0; j < n; ++j)
    {
        cout << ar[j] << " ";
    }
}
int main()
{
    int n,ar[100];
    cout<<"SELECTION SORT: "<<endl;
    cout<<"Enter the size of array: "<<endl;
    cin>>n;
    cout << "Enter "<<n<<" numbers: " << endl;
    for (int i = 0; i<n; ++i)
    {
        cin >> ar[i];
    }
    selection(ar,n);
    pri(ar,n);
    return 0;
}
```

Output:

```
Enter size of array: 4
Enter 4 numbers:
1245
```

```
23
56
1
The numbers are: 1 23 56 1245
```

SHELL SORT

Code:

```
using namespace std;
#include<iostream>
#include <conio.h>
void shell(int ar[],int n)
{
   for (int i=n/2;i>0;i/=2)
   {
      for (int j=i;j<n;j++)
      {
         int swap,temp=ar[j];
         for(swap=j;swap>=i&&ar[swap-i]>temp;swap-=i)
         {
            ar[swap]=ar[swap-i];
         }
         ar[swap]=temp;
      }
   }
}
void pri(int ar[],int n)
{
   cout << "The numbers are: "<<endl;
   for (int j = 0; j < n; ++j)
   {
      cout << ar[j] << " ";
   }
}
int main()
{
   int n,ar[100];
   cout<<"SHELL SORT: "<<endl;
   cout<<"Enter the size of array: "<<endl;
   cin>>n;
   cout << "Enter "<<n<<" numbers: " << endl;
   for (int i = 0; i<n; ++i)
   {
      cin >> ar[i];
   }
   shell(ar,n);
   pri(ar,n);
   return 0;
}
```

Output:

```
Enter size of array:

5
```

```
Enter 5 numbers:
45
2
576
9
124
The numbers are: 2 9 45 124 576
```

Radix SORT

Code:

```
using namespace std;
#include<iostream>
#include <conio.h>
int maxno(int ar[],int n)
{
    int max=ar[0];
    for(int i=1;i<n;i++)
    {
        if(max<ar[i])
        {
            max=ar[i];
        }
    }
    return max;
}
void sort(int ar[],int n,int i)
{
    const int m = 10;
    int output[n];
    int count[m];
    for (int j = 0; j < m; ++j)
        count[j] = 0;
    for (int j = 0; j < n; j++)
        count[(ar[j] / i) % 10]++;
    for (int j = 1; j < m; j++)
        count[j] += count[j - 1];
    for (int j = n - 1; j >= 0; j--)
    {
        output[count[(ar[j] / i) % 10] - 1] = ar[j];
        count[(ar[j] / i) % 10]--;
    }
    for (int j = 0; j < n; j++)
    ar[j] = output[j];
}
    void radix(int ar[],int n)
    {
        int m = maxno(ar,n);
        for (int i = 1; m / i > 0; i *= 10)
        {
            sort(ar,n,i);
        }
    }
void pri(int ar[],int n)
{
    cout << "The numbers are: "<<endl;
    for (int j = 0; j < n; ++j)
    {
        cout << ar[j] << " ";
    }
}
    int main()
    {
        int n,ar[100];
        cout<<"RADIX SORT: "<<endl;
        cout<<"Enter the size of array: "<<endl;
        cin>>n;
        cout << "Enter "<<n<<" numbers: " << endl;
        for (int i = 0; i<n; ++i)
        {
            cin >> ar[i];
        }
    radix(ar,n);
    pri(ar,n);
   return 0;
}
```

Output:

```
Enter the size of array:
5
Enter 5 numbers:
124
56
12
578
4
The numbers are:
4 12 56 124 578
```

BINARY SEARCH :

Code:

```
using namespace std;
#include<iostream>
#include <conio.h>
int sear1(int ar[],int search,int low,int high)
{
while (low <= high) {
int mid = low + (high - low) / 2;
if (ar[mid] == search)
return mid;
if (ar[mid] < search)
low = mid + 1;
else
high = mid - 1;
}
return -1;
}
int main(void)
{
int n;
int search;
cout<<"BINARY SEARCH: "<<endl;
cout<<"Enter the size of array: "<<endl;
cin>>n;
cout << "Enter "<<n<<" numbers: " << endl;
int ar[n];
for (int i = 0; i<n; ++i) {
cin >> ar[i];
}
cout << "Enter element to search: " << endl;
cin>>search;
int result=sear1(ar,search,0,n-1);
if (result==-1)
{
cout<<"Element not found"<<endl;
}
else
{
cout<<"Element "<<search<<" is at index:
"<<result<<endl;
}
return 0;
}
```

Output:

```
Enter the size of array :

5
```

```
Enter 5 numbers:
12
145
678
999
1000
Enter element to search:
678
Element 678 is at index:2
```

IMPLEMENTATION OF STACK USING ARRAY AND LINKED LIST

USING ARRAY :

Code:

```
#include <iostream>
using namespace std;
int stack[5], n=100, top=-1;
void push(int val)
{
   if(top>=n-1)
      cout<<"Stack Overflow"<<endl;
   else
   {
      top++;
      stack[top]=val;
   }
}
void pop()
{
   if(top<=-1)
      cout<<"Stack Underflow"<<endl;
   else
   {
      cout<<"The popped element is "<< stack[top] <<endl;
      top--;
   }
}
void display()
{
   if(top>=0)
   {
      cout<<"Stack elements are:";
      for(int i=top; i>=0; i--)
         cout<<stack[i]<<" ";
         cout<<endl;
   }
   else
      cout<<"Stack is empty";
}
int main()
{
   int ch, val;
   cout<<"1. Push in stack"<<endl;
   cout<<"2. Pop from stack"<<endl;
   cout<<"3. Print"<<endl;
   cout<<"4. Exit"<<endl;
   do
   {
      cout<<"Select: "<<endl;
```

```
        cin>>ch;
        switch(ch)
        {
            case 1:
            {
                cout<<"Enter value to push:"<<endl;
                cin>>val;
                push(val);
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
                break;
            }
            case 4:
            {
                cout<<"Exit"<<endl;
                break;
            }
            default:
            {
                Cout
<<"Invalid Choice"<<endl;
            }
        }
    }while(ch!=4);
    return 0;
}
```

```
Output: Push in stack

Pop from stack

Print

Exit

Select:

1

Enter value to push:

1
```

```
Select:

1

Enter value to push:

2

Select:

1

Enter value to push:

3

Select:

3

Stack elements are:3 21

Select:

2

The popped element is 3

Select:

3

Stack elements are:21

Select:

4

Exit
```

IMPLEMENTATION OF STACK USING ARRAY AND LINKED LIST

USING LINKED LIST :

Code:

```
#include <iostream>
using namespace std;
struct Node
{
   int data;
   struct Node *next;
};
struct Node* top = NULL

void push(int val)
{
   struct Node* newnode = (struct Node*) malloc(sizeof(struct Node));
   newnode->data = val;
   newnode->next = top;
   top = newnode;
}
void pop()
{
   if(top==NULL)
   cout<<"Stack Underflow"<<endl;
   else
   {
      cout<<"The popped element is "<< top->data <<endl;
      top = top->next;
   }
}
void display()
{
   struct Node* ptr;
   if(top==NULL)
   cout<<"stack is empty";
   else
   {
      ptr = top;
      cout<<"Stack elements are: ";
      while (ptr != NULL)
      {
         cout<< ptr->data <<" ";
         ptr = ptr->next;
      }
   }
   cout<<endl;
}
```

```
int main()
{
   int ch, val;
   cout<<"1 Push in stack"<<endl;
   cout<<"2 Pop from stack"<<endl;
   cout<<"3 Print"<<endl;
   cout<<"4 Exit"<<endl;
   do
   {
      cout<<"SELECT: "<<endl
      cin>>ch;
      switch(ch)
      {
         case 1:
         {
            cout<<"Enter value to push:"<<endl;
            cin>>val;
            push(val);
            break;
         }
         case 2:
         {
            pop();
            break;
         }
         case 3:
         {
            display();
            break;
         }
         case 4:
         {
            cout<<"Exit"<<endl;
            break;
         }
         default:
         {
            cout<<"Invalid Choice"<<endl;
         }
      }
   }while(ch!=4);
   return 0;
}
```

Output:

Push in stack

```
Pop from stack

Print

Exit

SELECT:

1

Enter value to push:

1

SELECT:

1

Enter value to push:

2

SELECT:

1
```

```
Enter value to push:

3

SELECT:

3

Stack elements are: 321

SELECT:

2

The popped element is 3

SELECT:

3

Stack elements are:

SELECT:
```

POSTFIX EVALUATION :

Code:

```
#include<iostream>
#include<cmath>
#include<stack>

using namespace std;
float scanNum(char ch)
{
   int value;
   value = ch;
   return float(value-'0');
}
int isOperator(char ch)
{
   if(ch == '+'|| ch == '-'|| ch == '*'|| ch == '/' || ch =='^')
      return 1;
   return -1;
}
int isOperand(char ch)
{
   if(ch >= '0' && ch <= '9')
   return 1;
   return -1;
}
float operation(int a, int b, char op)
{
   if(op == '+')
      return b+a;
   else if(op == '-')
      return b-a;
   else if(op == '*')
      return b*a;
   else if(op == '/')
      return b/a;
   else
   return pow(b,a);
}
float postfixEval(string postfix)
{
   int a, b;
   stack<float> stk;
   string::iterator it;
   for(it=postfix.begin();it!=postfix.end(); it++)
   {
      if(isOperator(*it) != -1)
      {
         a = stk.top();
```

```
            stk.pop();
            b = stk.top();
            stk.pop();
            stk.push(operation(a, b, *it));
        }else if(isOperand(*it) > 0)
        {
            stk.push(scanNum(*it));
        }
}
return stk.top();
}
int main()
{
    string post = "45*35+1-53";
    cout << "OUTPUT: "<<postfixEval(post);
}
```

Output: 3

BALANCING OF PARENTHESIS:

Code:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
bool areBracketsBalanced(string expr)
{
stack<char> s;
char x;
for (int i = 0; i < expr.length(); i++)
{
if (expr[i] == '(' || expr[i] == '['
|| expr[i] == '{')
{
s.push(expr[i]);
continue;
}
if (s.empty())
return false;
switch (expr[i]) {
case ')':
x = s.top();
s.pop();
if (x == '{' || x == '[')
return false;
break;
case '}':
x = s.top();
s.pop();
if (x == '(' || x == '[')
return false;
break;
case ']':
x = s.top();
s.pop();
if (x == '(' || x == '{')
return false;
break;
}
}
return (s.empty());
}
int main()
{
//string expr = "({(})}";
string expr="[ (5+6)*7-{7/4}+(3*2-8]";
/*
[ (5+6)*7-{7/4}+(3*2)-8]
*/
if (areBracketsBalanced(expr))
cout << "Balanced";
else
cout << "Not Balanced";
return 0;
}
```

Output :

```
Not Balanced
```

Implement all different types of queues. circular queue/priority queue/Double ended queue

Code:

```
#include <iostream>
using namespace std;
int cqueue[5];
int front = -1, rear = -1, n=5;
void insertCQ(int val) {
if ((front == 0 && rear == n-1) || (front == rear+1)) {
cout<<"Queue Overflow \n";
return;
}
if (front == -1) {
front = 0;
rear = 0;
} else {
if (rear == n - 1)
rear = 0;
else
rear = rear + 1;
}
cqueue[rear] = val ;
}
void deleteCQ() {
if (front == -1) {
cout<<"Queue Underflow\n";
return ;
}
cout<<"Element deleted from queue is :
"<<cqueue[front]<<endl;
if (front == rear) {
front = -1;
rear = -1;
} else {
if (front == n - 1)
front = 0;
else
front = front + 1;
}
}
void displayCQ() {
int f = front, r = rear;
if (front == -1) {
cout<<"Queue is empty"<<endl;
return;
}
cout<<"Queue elements are :\n";
if (f <= r) {
while (f <= r){
cout<<cqueue[f]<<" ";
f++;
}
} else {
while (f <= n - 1) {
cout<<cqueue[f]<<" ";
f++;
}
f = 0;
while (f <= r) {
cout<<cqueue[f]<<" ";
f++;
}
}
cout<<endl;
}
int main() {
int ch, val;
cout<<"1)Insert\n";
cout<<"2)Delete\n";
cout<<"3)Display\n";
cout<<"4)Exit\n";
do {
cout<<"Enter choice : "<<endl;
cin>>ch;
switch(ch) {
case 1:
cout<<"Enter the value: "<<endl;
cin>>val;
insertCQ(val);
break;
case 2:
deleteCQ();
break;
case 3:
displayCQ();
break;
case 4:
cout<<"Exit\n";
break;
default: cout<<"Incorrect!\n";
}
} while(ch != 4);
return 0;
}
```

Output:

```
1) Insert
```

```
2)Delete
3)Display
4) Exit
Enter choice :
1
Enter the value:
3
Enter choice :
1
Enter the value:
2
Enter choice :
```

```
1

Enter the value:

5

Enter choice:

1

Enter the value:

6

Enter choice:

3

Queue elements are:

3256

Enter choice:

2
```

```
Element deleted from queue is: 3

Enter choice:

3

Queue elements are:

256
```

Implementation of Priority Queue:

Code:

```
#include<iostream>
#include<conio.h>
using namespace std;
class queue
{
	public:
	int count;
	struct node
	{
		int data,priority;
		node*next;
	}*p;
	void insert(int no,int priority)
	{
		node*temp,*q;
		temp=new node;
		temp->data=no;
		temp->priority=priority;
		if(p==NULL||priority<p->priority)
		{
			temp->next=p;
			p=temp;
		}
		else
		{
			q=p;
			while(q->next!=NULL && q->next->priority<=priority)
			q=q->next;
			temp->next=q->next;
			q->next=temp;
		}
	}
	void del( )
	{
		node *ptr;
		if(p==NULL)
		{
			cout<<"queue overflow";
		}
		else
		{
			ptr=p;
			cout<<"\ndelete item :"<<ptr->data;
			p=p=p->next;
		}
	}
```

```
        void display( )
        {
                node *temp=p;
                if(p==NULL)
                {
                        cout<<"queue empty";
                }
                else
                {
                        cout<<"\npriority\titem\n";
                        do
                        {
                                cout<<temp->priority<<"\t\t"<<temp->data<<endl;
                                temp=temp->next;
                        }while(temp!=NULL);
                }
        }
        queue( )
        {p=NULL;}
};
int main( )
{
        queue l;
        cout<<"\nelements in the que are:\n";
        l.insert(20,5);
        l.insert(30,1);
        l.insert(40,2);
        l.insert(50,4);
        l.display( );
        l.del( );
        cout<<"\nafter deletion:\n\n";
        l.display( );
        return 0;
}
```

Output:

```
elements in the que are:



priority



1



2
```

```
4

5

item

30

40

se

20

delete item : 30

after deletion:

priority

2

4
```

5

item

40

50

20

Implementation of Double Ended Queue:

Code:

```
#include<iostream>
using namespace std;
#define SIZE 10
class dequeue {
int a[20],f,r;
public:
dequeue();
void insert_at_beg(int);
void insert_at_end(int);
void delete_fr_front();
void delete_fr_rear();
void show();
};
dequeue::dequeue() {
f=-1;
r=-1;
}
void dequeue::insert_at_end(int i) {
if(r>=SIZE-1) {
cout<<"\n insertion is not possible, overflow!!!!";
} else {
if(f==-1) {
f++;
r++;
} else {
r=r+1;
}
a[r]=i;
cout<<"\nInserted item is"<<a[r];
}
}
void dequeue::insert_at_beg(int i) {
if(f==-1) {
f=0;
a[++r]=i;
cout<<"\n inserted element is:"<<i;
} else if(f!=0) {
a[--f]=i;
cout<<"\n inserted element is:"<<i;
} else {
cout<<"\n insertion is not possible, overflow!!!";
}
}
void dequeue::delete_fr_front() {
if(f==-1) {
cout<<"deletion is not possible::dequeue is empty";
return;
```

```
}
else {
cout<<"the deleted element is:"<<a[f];
if(f==r) {
f=r=-1;
return;
} else
f=f+1;
}
}
void dequeue::delete_fr_rear() {
if(f==-1) {
cout<<"deletion is not possible::dequeue is empty";
return;
}
else {
cout<<"the deleted element is:"<<a[r];
if(f==r) {
f=r=-1;
} else
r=r-1;
}
}
void dequeue::show() {
if(f==-1) {
cout<<"Dequeue is empty";
} else {
for(int i=f;i<=r;i++) {
cout<<a[i]<<" ";
}
}
}
int main()
{
int c,i;
dequeue d;
do {
cout<<"\n 1.insert at beginning";
cout<<"\n 2.insert at end";
cout<<"\n 3.show";
cout<<"\n 4.deletion from front";
cout<<"\n 5.deletion from rear";
cout<<"\n 6.exit";
cout<<"\n enter your choice:";
cin>>c;
switch(c) {
case 1:
cout<<"enter the element to be inserted: ";
cin>>i;
d.insert_at_beg(i);
break;
case 2:
cout<<"enter the element to be inserted: ";
cin>>i;
d.insert_at_end(i);
break;
case 3:
d.show();
break;
case 4:
d.delete_fr_front();
break;
case 5:
d.delete_fr_rear();
break;
case 6:
exit(1);
break;
default:
cout<<"invalid choice";
break;
}
}
while(c!=7);
}
```

Output:

```
1.insert at beginning

2.insert at end

3.show

4.deletion from front

5.deletion from rear

6.exit

enter your choice:1

enter the element to be inserted: 3

inserted element is:3

1. insert at beginning

2.Insert at end

3.show
```

```
4.deletion from front
5.deletion from rear
6.exit
enter your choice:2
enter the element to be inserted: 4
Inserted item 154
1.insert at beginning
2. Insert at end
3.show
4.deletion from front
5.deletion from rear
6.exit
enter your choice:3
34
1.insert at beginning
2. insert at end
3.show
4.deletion from front
5.deletion from rear
6.exit
enter your choice:5
the deleted element is:4
1.insert at beginning
2. insert at end
3.show
4.deletion from front
```

```
5.deletion from rear

6.exit

enter your choice:3

3
```

Demonstrate applications of queues
A) Priority Queue:
B) Breadth first search:

Code:

A) Priority Queue:

```
#include<iostream>
#include<conio.h>
using namespace std;
class queue
{
        public:
        int count;
        struct node
        {
                int data,priority;
                node*next;
        }*p;
        void insert(int no,int priority)
        {
                node*temp,*q;
                temp=new node;
                temp->data=no;
                temp->priority=priority;
                if(p==NULL||priority<p->priority)
                {
                        temp->next=p;
                        p=temp;
                }
                else
                {
                        q=p;
                        while(q->next!=NULL && q->next->priority<=priority)
                        q=q->next;
                        temp->next=q->next;
                        q->next=temp;
                }
        }
        void del( )
        {
                node *ptr;
                if(p==NULL)
                {
                        cout<<"queue overflow";
                }
                else
                {
                        ptr=p;
                        cout<<"\ndelete item :"<<ptr->data;
                        p=p=p->next;
                }
        }
        void display( )
        {
                node *temp=p;
                if(p==NULL)
                {
                        cout<<"queue empty";
                }
                else
                {
                        cout<<"\npriority\titem\n";
                        do
                        {
                                cout<<temp->priority<<"\t\t"<<temp->data<<endl;
                                temp=temp->next;
                        }while(temp!=NULL);
                }
        }
        queue( )
        {p=NULL;}
};
int main( )
{
        queue l;
        cout<<"\nelements in the que are:\n";
        l.insert(20,5);
        l.insert(30,1);
        l.insert(40,2);
        l.insert(50,4);
        l.display( );
        l.del( );
        cout<<"\nafter deletion:\n\n";
        l.display( );
        return 0;
}
```

```
Output:Elements in the que are:

Priority        item

    1               30
    2               40
    4               50
    5               20

    Delete item 30

    After deletion:
```

Priority	item
2	40
4	50
5	20

B) Breadth first search:

Code:

```
#include<iostream>
#include <list>
using namespace std;
class Graph
{
int V;
list<int> *adj;
public:
Graph(int V);
void addEdge(int v, int w);
void BFS(int s);
};
Graph::Graph(int V)
{
this->V = V;
adj = new list<int>[V];
}
void Graph::addEdge(int v, int w)
{
adj[v].push_back(w);
}
void Graph::BFS(int s)
{
bool *visited = new bool[V];
for(int i = 0; i < V; i++)
visited[i] = false;
list<int> queue;
visited[s] = true;
queue.push_back(s);
list<int>::iterator i;
while(!queue.empty())
{
s = queue.front();
cout << s << " ";
queue.pop_front();
for (i = adj[s].begin(); i != adj[s].end(); ++i)
{
if (!visited[*i])
{
visited[*i] = true;
queue.push_back(*i);
}
}
}
}
int main()
```

```
{
Graph g(4);
g.addEdge(0, 1);
g.addEdge(0, 2);
g.addEdge(1, 2);
g.addEdge(2, 0);
g.addEdge(2, 3);
g.addEdge(3, 3);
cout << "Following is Breadth First Traversal "
<< "(starting from vertex 2) \n";
g.BFS(2);
return 0;}
```

Output

```
Following is Breadth Frst Traversal
2 0 3 1
```

Implementation of all types of linked lists.

A. Singly Linked List:

Code:

```
#include <iostream>
using namespace std;
struct Node {
int data;
struct Node *next;
};
struct Node* head = NULL;
void insert(int new_data) {
struct Node* new_node = (struct Node*)
malloc(sizeof(struct Node));
new_node->data = new_data;
new_node->next = head;
head = new_node;
}
void display() {
struct Node* ptr;
ptr = head;
while (ptr != NULL) {
cout<< ptr->data <<" ";
ptr = ptr->next;
}
}
int main() {
insert(1);
insert(2);
insert(3);
insert(4);
insert(5);
cout<<"The linked list is: ";
display();
return 0;
}
```

Output:

```
The linked list is : 5 4 3 2 1
```

B.Double Linked List:

Code:

```
#include <iostream>
using namespace std;
struct Node {
int data;
struct Node *prev;
struct Node *next;
};
struct Node* head = NULL;
void insert(int newdata) {
struct Node* newnode = (struct Node*) malloc(sizeof(struct Node));
newnode->data = newdata;
newnode->prev = NULL;
newnode->next = head;
if(head != NULL)
head->prev = newnode ;
head = newnode;
}
void display() {
struct Node* ptr;
ptr = head;
while(ptr != NULL) {
cout<< ptr->data <<" ";
ptr = ptr->next;
}
}
int main() {
insert(1);
insert(2);
insert(3);
insert(4);
insert(5);
cout<<"The doubly linked list is: ";
display();
return 0;
}
```

Output:

```
The doubly linked list is:5 4 3 2 1
```

C. Circular Linked List:

Code:

```
#include <iostream>
using namespace std;
struct Node {
int data;
struct Node *next;
};
struct Node* head = NULL;
void insert(int newdata) {
struct Node *newnode = (struct Node *)malloc(sizeof(struct Node));
struct Node *ptr = head;
newnode->data = newdata;
newnode->next = head;
if (head!= NULL) {
while (ptr->next != head)
ptr = ptr->next;
ptr->next = newnode;
} else
newnode->next = newnode;
head = newnode;
}
void display() {
struct Node* ptr;
ptr = head;
do {
cout<<ptr->data <<" ";
ptr = ptr->next;
} while(ptr != head);
}
int main() {
insert(1);
insert(2);
insert(3);
insert(4);
insert(5);
cout<<"The circular linked list is: ";
display();
return 0;
}
```

Output:

```
The circular linked list is: 5 4 3 2 1
```