Queue implementation using linked list

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

struct node
 {  char name[20];
    int age;
    float height;
    node *nxt;
 };
node *start_ptr=NULL;

int main()
{

  void push ();
  void pop();
  char ch;
  clrscr();
  cout<<"Queue
";
  cout<<"-----";

  do
  {
   cout<<"
Select an operation";
   cout<<"
u->push
";
   cout<<"o->pop
";
   cout<<"e->exit
";

   cin>>ch;

   switch(ch)
   {
    case 'u':
                push();
                break;
    case 'o':
                pop();
                break;
    case 'e':
                exit(0);
   }
   }while(ch!='e');

             return 0;

}

void pop()
```

```
{
 node *temp1,*temp2;
 if(start_ptr==NULL)
   cout<<"The list is empty
";
 else
  {
   temp1=start_ptr;
   temp2=temp1;
  while(temp1->nxt!=NULL)
   {
    temp2=temp1;
    temp1=temp1->nxt;
   }
  if(temp1==temp2)
    {
     cout<<temp1->name<<",";
     cout<<temp1->age<<", ";
     cout<<temp1->height;
     start_ptr=NULL;
    }
  else
    {
     cout<<temp1->name<<", ";
     cout<<temp1->age<<", ";
     cout<<temp1->height;
     temp2->nxt=NULL;
     delete temp1;
    }
 }
}

void push ()
 {
    node *temp;

    temp = new node;
    cout << "Please enter the name of the person: ";
    cin >> temp->name;
    cout << "Please enter the age of the person: ";
    cin >> temp->age;
    cout << "Please enter the height of the person: ";
    cin >> temp->height;
    if (start_ptr == NULL)
              {
               temp->nxt=NULL;
               start_ptr = temp;
              }
    else
              {
      temp->nxt=start_ptr;
      start_ptr=temp;

              }
 }
```

Inserting, deleting and displaying elements in the Double Circular Linked List

Code :

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

class cirdlink
{
        struct node
        {
         int data;
         node *rnext;
         node *lnext;
        }*new1,*head,*tail,*ptr,*temp;

 public:

 cirdlink()
 {
         head=tail=NULL;
 }

 void creation();
 void insertion();
 void deletion();
 void display();
};

void cirdlink :: creation()
{
        if(head==NULL)
        {
          new1=new node[sizeof(node)];
          new1->rnext=NULL;
          new1->lnext=NULL;
          cout<<"enter student number  :";
          cin>>new1->data;
          head=new1;
          tail=new1;
          head->rnext=tail;
          head->lnext=tail;
          tail->rnext=head;
          tail->lnext=head;
        }
        else
        cout<<" creation done only once  !";
 }

 void cirdlink :: insertion()
 {
         int i,pos;
         new1=new node[sizeof(node)];
         new1->rnext=NULL;
         new1->lnext=NULL;
         cout<<"enter student number :";
         cin>>new1->data;
         cout<<"enter position you want to insert  :";
         cin>>pos;
```

```
        if(pos==1)
        {
         new1->rnext=head;
         head=new1;
         tail->lnext=head;
         tail->rnext=head;
         head->lnext=tail;
        }
        else
        {
         i=1;
         temp=head;
         while(i < pos-1 && temp->rnext!=tail)
         {
                  i++;
                  temp=temp->rnext;
         }
         if(temp->rnext==tail)
         {
                           new1->rnext=tail->rnext;
                           tail->rnext=new1;
                           new1->lnext=tail;
                           tail=new1;
                           head->lnext=tail;
         }
         else
         {
                           new1->rnext=temp->rnext;
                           new1->lnext=temp;
                           temp->rnext=new1;
                           new1->rnext->lnext=new1;
         }
        }
}
void  cirdlink :: deletion()
{
                  int pos,i;

                  cout<<"Enter Position you want to Delete ?";
                  cin>>pos;

                  if(pos==1 && head!=tail)
                  {
                           ptr=head;
                           head=head->rnext;
                           head->lnext=tail;
                           tail->rnext=head;
                           delete ptr;
                  }
                  else
                  {
                           i=1;
                           temp=head;
                           while(i < pos-1 && temp->rnext!=tail)
                           {
                             i++;
                             temp=temp->rnext;
```

```
                }
                if(temp->rnext!=tail)
                {
                        ptr=temp->rnext;
                        temp->rnext=ptr->rnext;
                        ptr->rnext->lnext=ptr->lnext;
                        delete ptr;
                }
                else
                {
                  if(temp->rnext==tail && head!=tail)
                  {
                                ptr=tail;
                                tail=temp;
                                tail->rnext=head;
                                head->lnext=tail;
                                delete ptr;
                        }
                        else
                        {
                          head=NULL;
                          tail=NULL;
                          delete head;
                          delete tail;
                  }
                }
          }
        }
void cirdlink::display()
{
          int ch;
          cout<<"1.forward
2.backward
:";
          cout<<"Enter your choice<1/2>?";
          cin>>ch;
          switch(ch)
          {
                case 1:  if(head!=NULL)
                                {
                                        temp=head;
                                        while(temp!=tail)
                                        {
                                         cout<<temp->data<<"    ";
                                         temp=temp->rnext;
                                        }
                                        if(temp==tail)
                                          cout<<temp->data;
                                        }
                                        break;

                case 2 :  if(tail!=NULL)
                                       {
                                         temp=tail;
                                         while(temp!=head)
                                         {
                                                cout<<temp->data<<"    ";
```

```
                                                        temp=temp->lnext;
                                                    }
                                                    if(temp==head)
                                                        cout<<temp->data;
                                                }
                                                break;
                        }
                }

                void main()
                {
                        cirdlink c1;

                        int ch;
                        char op;
                        do
                        {
                                clrscr();
                                cout<<"----------Menu------------
";
                                cout<<"1.Creation
2.Insertion
3.Deletion
4.Display
";
                                cout<<"Enter Your choice <1..4> ?";
                                cin>>ch;
                                switch(ch)
                                {
                                        case 1 :   c1.creation();
                                                                break;
                                        case 2 :   c1.insertion();
                                                                break;
                                        case 3 :c1.deletion();
                                                                break;
                                        case 4 :c1.display();
                                                                break;
                                }
                                cout<<"Do you want to continue <Y/N> ?";
                                cin>>op;
                        }while(op=='y' || op=='Y');
                        getch();
                }
```

Program for Priority Queue

Code :

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

        const int MAX=5;

        class pqueue
        {
```

```
                int front,rear;
               public:
                struct data
                {
                 int val,p,o;
                }d[MAX];

                pqueue()
                {
                        front=rear=-1;
                }
                void insert(data d1);
                data deletion();
                void display();
        };
        void pqueue :: insert(data d1)
        {
                if(rear==MAX-1)
                  cout<<"Priority Queue is Full
";
                else
                {
                  rear++;
                  d[rear]=d1;
                  if(front==-1)
                          front=0;
                  data temp;
                  for(int i=front;i<=rear;i++)
                          for(int j=i+1;j<=rear;j++)
                          {
                                  if(d[i].p > d[j].p)
                                  {
                                          temp=d[i];
                                          d[i]=d[j];
                                          d[j]=temp;
                                  }
                                  else
                                  {
                                          if(d[i].p==d[j].p)
                                          {
                                                  if(d[i].o > d[j].o)
                                                  {
                                                          temp=d[i];
                                                          d[i]=d[j];
                                                          d[j]=temp;
                                                  }
                                          }
                                  }
                }
        }
        data pqueue :: deletion()
        {
                data d1;
                if(front==-1)
                  cout<<"Priority Queue is Empty
";
```

```
                else
                {
                  d1=d[front];
                  if(front==rear)
                            front=rear=-1;
                  else
                            front++;
                }
                return d1;
        }
        void pqueue :: display()
        {
                 if(front==-1)
                            cout<<"Priority Queue is Empty
";
                 else
                 {
                            for(int i=front;i<=rear;i++)
                            {
                                      cout<<"Object  :"<<i+1<<endl;
                                      cout<<"Value ="<<d[i].val<<endl;
                                      cout<<"Priority="<<d[i].p<<endl;
                                      cout<<"Order =  "<<d[i].o<<endl;
                            }
                 }
        }
        void main()
        {
          pqueue p1;

          data d1;
          char op;
          do
          {
                    int ch;

                    clrscr();
                    cout<<"----------Menu----------
";
                    cout<<"1.Insertion
2.Deletion
3.Display
4.Exit
";
                    cout<<"Enter your Choice<1..4> ?";
                    cin>>ch;
                    switch(ch)
                    {
                      case 1 :  cout<<"Enter Value ?";
                                                cin>>d1.val;
                                                cout<<"Enter Priority?";
                                                cin>>d1.p;
                                                cout<<"Enter Order ?";
                                                cin>>d1.o;
                                                p1.insert(d1);
                                                break;
                      case 2 :  d1=p1.deletion();
```

```
                                        cout<<"Value = "<<d1.val<<endl;
                                        cout<<"Priority = "<<d1.p<<endl;
                                        cout<<"Order ="<<d1.o<<endl;
                                        break;
                    case 3 :  p1.display();
                                        break;
                    }
                    cout<<"Do You Want to Continue <Y/N> ?";
                    cin>>op;
                  }while(op=='Y' || op=='y');
                  getch();
          }
```

Binary Search is a good searching algorithm.
And also Fast inprocessing. Here you have to input sorted number in an array first then find out the number

```
//Binary Searching

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

void main()
{
        clrscr();
        int l_v=1;
        int h_v;
        int a[51];
        int middle;
        int num,i=0;
        cout<<"Input your sorted num :
";
        while(scanf("%d",&a[i])!=EOF)
                i++;
        h_v=i;
        cout<<"
The input numbers are :
";
        i=0;
        while(i<h_v)
        {
                cout<<a[i]<<endl;
                i++;
        }
        getch();
        cout<<"

Input your searching number :
";
        cin>>num;
        for(int n=0;a[middle]!=num;n++)
        {
```

```
                middle = (l_v + h_v)/2;
                if(a[middle]>num)
                        h_v = middle - 1;
                else if(a[middle]<num)
                        l_v = middle + 1;
                else if(a[middle]==num)
                        cout<<"
Found your number in "<<middle+1<<" position.";
        }
        cout<<"

This program's loop is executed "<<n<<" times to find out
the
number.";
        getch();
}
```

A Templated Stack Data Structure Example

```
#include <dos.h>          // For sleep()
#include <iostream.h>     // For I/0
#include <windows.h>      // FOR MessageBox() API
#include <conio.h>

#define MAX 10        // MAXIMUM STACK CONTENT

template <class T>    // Using Templates so that any type of data can be
                                                        // stored in Stack without multiple defination of
class
class stack
{

 protected:

 T arr[MAX];      // Contains all the Data

 public:
  T item,r;
  int top;       //Contains location of Topmost Data pushed onto Stack
                stack()       //Constructor
                {
                        for(int i=0;i<MAX;i++)
                                {
                                  arr[i]=NULL;          //Initialises all Stack Contents to NULL
                                }

                        top=-1;     //Sets the Top Location to -1 indicating an empty stack
                }

                void push(T a)  // Push ie. Add Value Function
                {
                        top++;       // increment to by 1
                        if(top<MAX)
                        {
                                arr[top]=a;        //If Stack is Vacant store Value in Array
                        }
                        else    // Bug the User
                        {
```

```
                                        MessageBox(0,"STACK IS FULL","STACK
WARNING!",MB_ICONSTOP);
                                        top--;
                                }
                }

                T pop()          // Delete Item. Returns the deleted item
                {
                                if(top==-1)
                                {
                                        MessageBox(0,"STACK IS EMPTY
","WARNING",MB_ICONSTOP);
                                        return NULL;
                                }
                                else
                                {
                                        T data=arr[top];    //Set Topmost Value in data
                                        arr[top]=NULL;      //Set Original Location to NULL
                                        top--;            // Decrement top by 1
                                        return data;       // Return deleted item
                                }
                }
};


void main()
{
 stack <int>a;    // Create object of class a with int Template
 int opt=1;
 while (opt!=3)
 {
 clrscr();
 cout<<" MAX STACK CAPACITY="<<((MAX-a.top)-1)<<"



";
 cout<<"1) Push Item
";
 cout<<"2) Pop Item
";
 cout<<"3) Exit

";
 cout<<"Option?";
 cin>>opt;
 switch(opt)
 {
  case 1:
  cout<<"Which Number should be pushed?";
  cin>>a.item;
  a.push(a.item);
  break;

  case 2:
  a.r=a.pop();
  cout<<"Item popped from Stack is:"<<a.r<<endl;
```

```
 sleep(2);
 break;
 }
 }
 }
```

Merge Two Sorted Linked List To Form A Third Linked List

Code :

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

//   Creating a NODE Structure
struct node
{
  int data;  // data
  struct node *next;  // link to next node and previous node
};

// Creating a class LIST
class list
{
  struct node *start;
  public:
    void create(); // to create a list
    void show();   // show
    void merge(list,list);  // Merge two list's
};

// Main function
int main()
{
  clrscr();
  list l1,l2,l3;
  cout<<"Enter the First List in ascending order.
";
  l1.create(); // to create a first list
  cout<<"
Enter the Second List in ascending order.
";
  l2.create(); // to create a second list
  cout<<"
The first list is
";
  l1.show();
  cout<<"
The second list is
";
  l2.show();
  l3.merge(l1,l2);
  l3.show();
  getch();
  return (0);
```

```
}

//    Functions

// Creating a new node
void list::create()
{
  struct node *nxt_node,*pre_node;
  int value,no,i;
  start=nxt_node=pre_node=NULL;
  cout<<"
How many nodes : ";
  cin>>no;
  cout<<"Enter "<<no<<" Elements: ";
  for(i=1;i<=no;i++)
  {
    cin>>value;
    nxt_node=new node;
    nxt_node->data=value;
    nxt_node->next=NULL;
    if(start==NULL)
          start=nxt_node;
    else
          pre_node->next=nxt_node;
    pre_node=nxt_node;
  }
  cout<<"
The list is created!

";
}

// Displaying LIST
void list::show()
{
  struct node *ptr=start;
  cout<<"

The List is
";
  while(ptr!=NULL)
  {
    cout<<ptr->data<<" -> ";
    ptr=ptr->next;
  }
  cout<<"□□□   ";
}

void list::merge(list l1,list l2)
{
  struct node *nxt_node,*pre_node,*pptr,*qptr;
  int dat;
  pptr=l1.start;
  qptr=l2.start;
  start=nxt_node=pre_node=NULL;
  while(pptr!=NULL && qptr!=NULL)
  {
```

```
    if(pptr->data<=qptr->data)
    {
          dat=pptr->data;
          pptr=pptr->next;
    }
    else
    {
          dat=qptr->data;
          qptr=qptr->next;
    }
    nxt_node=new node;
    nxt_node->data=dat;
    nxt_node->next=NULL;
    if(start==NULL)
          start=nxt_node;
    else
          pre_node->next=nxt_node;
    pre_node=nxt_node;
  }
  if(pptr==NULL)
  {
    while(qptr!=NULL)
    {
          nxt_node=new node;
          nxt_node->data=qptr->data;
          nxt_node->next=NULL;
          if(start==NULL)
            start=nxt_node;
          else
            pre_node->next=nxt_node;
          pre_node=nxt_node;
          qptr=qptr->next;
    }
  }
  else if(qptr==NULL)
  {
    while(pptr!=NULL)
    {
          nxt_node=new node;
          nxt_node->data=pptr->data;
          nxt_node->next=NULL;
          if(start==NULL)
            start=nxt_node;
          else
            pre_node->next=nxt_node;
          pre_node=nxt_node;
          pptr=pptr->next;
    }
  }
  cout<<"
The lists are merged.";
  return;
}
```

This program will take two doubly-linked lists of nodes and merges them into another doubly-linked list of nodes.

Code :

```
#include<iostream>
using namespace std;
class Node
{
  public:
    int data;
    Node *next;
    Node *prev;
    Node()
    {
      data=0;
      next=prev=NULL;
    }
};
class Node1
{
    public:
      int data;
      Node1 *next;
      Node1 *prev;
      Node1()
      {
        data=0;
        next=prev=NULL;
      }
};
class Node2
{
   public:
     int data;
     Node2 *next;
     Node2 *prev;
     Node2()
     {
        data=0;
        next=prev=NULL;
     }
};
class DoublyLinkedList
{
    private:
       Node *headNode, *tailnode;
       Node1 *headNode1, *tailNode1;
       Node2 *headNode2, *tailNode2;
    public:
         DoublyLinkedList()
         {
           headNode=tailNode=NULL;
           headNode1=tailNode1=NULL;
           headNode2=tailNode2=NULL;
         }
        void Insert();//Insert data into the two lists
        void Merge() ;//Merging two lists into one
        void Display();//Display only the merged list
};
```

```
void DoublyLinkedList::Insert()
{
   char option;
   //This section inserts elements into the nodes of the first list
   do
   {
     Node *newnode = new Node();
     cin>>newnode->data;
     newnode->next=NULL;
     newnode->prev=NULL;
     if(headNode==NULL)
     {
       headNode= tailNode=newnode;
     }
     else
     {
       Node *curr = headNode;
       while(curr->next!=NULL)
        {
          curr=curr->next;
        }
        curr->next=tailNode=newnode;
        newnode->prev=curr;
     }
     cout<<"Enter y to continue adding data into the first list :";
     cin>>option;
   }
   while(option=='y'||option=='Y');

   //The section inserts the elements into the nodes of the second
list
   do
   {
     Node1 *newnode = new Node1();
     cin>>newnode->data;
     newnode->next=NULL;
     newnode->prev=NULL;
     if(headNode1==NULL)
     {
       headNode1=tailNode1=newnode;
     }
     else
     {
       Node1 *curr = headNode1;
       while(curr->next!=NULL)
        {
          curr=curr->next;
        }
        curr->next= tailNode1=newnode;
        newnode->prev=curr;
     }
     cout<<"Enter y to continue adding data into the second list :";
     cin>>option;
   }
   while(option=='y'||option=='Y');
}
void DoublyLinkedList::Merge()
```

```
{
  Node *currentNode=headNode;
  Node1 *currentNode1=headNode1;
  //This section of code copies all the data from list 1 into the new
list
  while(currentNode!=NULL)
  {
    Node2 *newnode = new Node2();
    newnode->data = currentNode->data;
    newnode->next = NULL;
    newnode->prev = NULL;
    if(headNode2==NULL)
    {
     headNode2=tailNode2=newnode;
    }
    else
    {
     Node2 *temp = headNode2;
     while(temp->next!=NULL)
     {
      temp=temp->next;
     }
     temp->next=tailNode2=newnode;
     newnode->prev=temp;
    }
     currentNode=currentNode->next;
   }

   //This section of code appends the new list with data from the
second
list
   while(currentNode1!=NULL)
   {
    Node2 *newnode = new Node2();
    newnode->data = currentNode1->data;
    newnode->next = newnode->prev=NULL;
    if(tailNode2->next==NULL)
    {
      tailNode2->next=newnode;
      newnode->prev = tailNode2;
      tailNode2=newnode;
    }
    currentNode1= currentNode1->next;
 }
}
void DoublyLinkedList::Display()
{
   Node2 *currentNode2 = headNode2;
   while(currentNode2!=NULL)
   {
     cout<<currentNode2->data<<" ";
     currentNode2=currentNode2->next;
   }
   cout<<endl;
}

void main()
```

```
{
   DoublyLinkedList DLL;
   DLL.Insert();
   DLL.Merge();
   DLL.Display();
}
```

Matrics Addition

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

struct matrics
{
int a[4][4], m, n;
};

matrics add_mat(matrics m1, matrics m2);                    //prototype declared

int main()
{
matrics x1, x2, x3;
int i, j;
cout<<"
Enter the size of matrics
";
cout<<"
Enter rows
";
cin>>x1.m;
cout<<"
Enter columns
";
cin>>x1.n;
cout<<"
Enter the elements (Row-wise & Column-wise)
          for(i=0; i<x1.m; i++)
          {
                    for(j=0; j<x1.n; j++)
                    cin>>x1.a[i][j];
          }
cout<<"
Enter the size of 2nd matrics
";
cout<<"
Enter rows
";
cin>>x2.m;
cout<<"
Enter columns
";
cin>>x2.n;
cout<<"
Enter the elements (Row-wise & column-wise)
          for(i=0; i<x2.m; i++)
          {
```

```
                for(j=0; j<x2.n; j++)
                cin>>x2.a[i][j];
        }
x3=add_mat(x1,x2);
cout<<"
The added matrics is
";
        for(i=0; i<x3.m; i++)
        {
                for(j=0; j<x3.n; j++)
                cout<<x3.a[i][j]<<" ";
        }

matrics add_mat(matrics m1, matrics m2)
{
matrics m3;                     //object of matrics type
int k, l;
if((m1.m==m2.m) && (m1.n==m2.n))
        {  m3.m=m1.m+m2.m;
           m3.n=m1.n+m2.n;
        }

        for(k=0; k<m1.m; k++)
        {
                for(l=0; l<m1.n; l++)
                m3.a[k][l]=m1.a[k][l]+m2.a[k][l];
        return (m3);
        }
}//             end of If loop

else
cout<<"
Addition is not possible
";
}
```

Matrics Addition

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

struct matrics
{
int a[4][4], m, n;
};

matrics add_mat(matrics m1, matrics m2);                    //prototype declared

int main()
{
matrics x1, x2, x3;
int i, j;
cout<<"
Enter the size of matrics
";
cout<<"
```

```
Enter rows
";
cin>>x1.m;
cout<<"
Enter columns
";
cin>>x1.n;
cout<<"
Enter the elements (Row-wise & Column-wise)
        for(i=0; i<x1.m; i++)
        {
                for(j=0; j<x1.n; j++)
                cin>>x1.a[i][j];
        }
cout<<"
Enter the size of 2nd matrics
";
cout<<"
Enter rows
";
cin>>x2.m;
cout<<"
Enter columns
";
cin>>x2.n;
cout<<"
Enter the elements (Row-wise & column-wise)
        for(i=0; i<x2.m; i++)
        {
                for(j=0; j<x2.n; j++)
                cin>>x2.a[i][j];
        }
x3=add_mat(x1,x2);
cout<<"
The added matrics is
";
        for(i=0; i<x3.m; i++)
        {
                for(j=0; j<x3.n; j++)
                cout<<x3.a[i][j]<<" ";
        }

matrics add_mat(matrics m1, matrics m2)
{
matrics m3;                     //object of matrics type
int k, l;
if((m1.m==m2.m) && (m1.n==m2.n))
        {  m3.m=m1.m+m2.m;
           m3.n=m1.n+m2.n;
        }

        for(k=0; k<m1.m; k++)
        {
                for(l=0; l<m1.n; l++)
                m3.a[k][l]=m1.a[k][l]+m2.a[k][l];
        return (m3);
        }
```

```
}//                  end of If loop

else
cout<<"
Addition is not possible
";
}
```

To add and subtract two sparse matrices.
Give two matrices in its simple form.

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

int main()
{
  clrscr();
  int sparse1[10][3],sparse2[10][3],sum[10][3],diff[10][3];
  int m,n,p,q,t1,t2,s,d,element;
  int i,j;
  cout<<"Enter the number of rows and columns : ";
  cin>>m>>n;
  t1=t2=0;

  cout<<"
Enter the first matrix("<<m<<"*"<<n<<"):
";
  for(i=1;i<=m;i++)
  {
    for(j=1;j<=n;j++)
    {
          cin>>element;
          if(element!=0)
          {
            t1=t1+1;
            sparse1[t1][1]=i;
            sparse1[t1][2]=j;
            sparse1[t1][3]=element;
          }
    }
  }
  sparse1[0][1]=m;
  sparse1[0][2]=n;
  sparse1[0][3]=t1;
  cout<<"
Enter the second matrix("<<m<<"*"<<n<<"):
";
  for(i=1;i<=m;i++)
  {
    for(j=1;j<=n;j++)
    {
          cin>>element;
          if(element!=0)
```

```
            {
              t2=t2+1;
              sparse2[t2][1]=i;
              sparse2[t2][2]=j;
              sparse2[t2][3]=element;
            }
      }
    }
    sparse2[0][1]=m;
    sparse2[0][2]=n;
    sparse2[0][3]=t2;

    // displaying the first sparse matrix
    cout<<"

The first sparse matrix is :

Row      Column Element";
    cout<<"
----------------------
";
    for(i=0;i<=t1;i++)
    {

cout<<sparse1[i][1]<<"  "<<sparse1[i][2]<<"       "<<sparse1[i][3]<<"
";
    }
     // displaying the second sparse matrix
    cout<<"

The second sparse matrix is :

Row      Column Element";
    cout<<"
----------------------
";
    for(i=0;i<=t2;i++)
    {

cout<<sparse2[i][1]<<"  "<<sparse2[i][2]<<"       "<<sparse2[i][3]<<"
";
    }

    // Addition and subtraction
    i=j=s=d=1;
    while((i<=t1)&&(j<=t2))
    {
      if(sparse1[i][1]==sparse2[j][1])    // if rows are equal
      {
            if(sparse1[i][2]==sparse2[j][2]) // if columns are equal
            {
              sum[s][1]=diff[d][1]=sparse1[i][1];
              sum[s][2]=diff[d][2]=sparse1[i][2];
              sum[s][3]=sparse1[i][3]+sparse2[j][3];
              diff[d][3]=sparse1[i][3]-sparse2[j][3];
              i++;
              j++;
```

```
            if(sum[s][3]!=0)
              s++;
            if(diff[d][3]!=0)
              d++;
          }
          else   // if columns are not equal
          {
            if(sparse1[i][2]<sparse2[j][2])
            {
              sum[s][1]=diff[d][1]=sparse1[i][1];
              sum[s][2]=diff[d][2]=sparse1[i][2];
              sum[s][3]=diff[d][3]=sparse1[i][3];
              i++;
              s++;
              d++;
            }
            else
            {
              sum[s][1]=diff[d][1]=sparse2[j][1];
              sum[s][2]=diff[d][2]=sparse2[j][2];
              sum[s][3]=sparse2[j][3];
              diff[d][3]=0-sparse2[j][3];
              j++;
              d++;
              s++;
            }
          }
    }
    else   //   if rows are not equal
    {
          if(sparse1[i][1]<sparse2[j][1])
          {
            sum[s][1]=diff[d][1]=sparse1[i][1];
            sum[s][2]=diff[d][2]=sparse1[i][2];
            sum[s][3]=diff[d][3]=sparse1[i][3];
            i++;
            d++;
            s++;
          }
          else
          {
            sum[s][1]=diff[d][1]=sparse2[j][1];
            sum[s][2]=diff[d][2]=sparse2[j][2];
            sum[s][3]=sparse2[j][3];
            diff[d][3]=0-sparse2[j][3];
            j++;
            s++;
            d++;
          }
    }
  }  // end of while
  if(i<=t1)
  {
    for(p=i;p<=t1;p++)
    {
          sum[s][1]=diff[d][1]=sparse1[p][1];
          sum[s][2]=diff[d][2]=sparse1[p][2];
```

```
            sum[s][3]=diff[d][3]=sparse1[p][3];
            s++;
            d++;
     }
   }
   else if(j<=t2)
   {
     for(p=j;p<=t2;p++)
     {
            sum[s][1]=diff[d][1]=sparse2[p][1];
            sum[s][2]=diff[d][2]=sparse2[p][2];
            sum[s][3]=sparse2[p][3];
            diff[d][3]=0-sparse2[j][3];
            s++;
            d++;
     }
   }
  // end of addition and subtraction
  sum[0][1]=diff[0][1]=m;
  sum[0][2]=diff[0][2]=n;
  sum[0][3]=s-1;
  diff[0][3]=d-1;

  // displaying the sum matrix
   cout<<"

The sum is :

Row      Column Element";
   cout<<"
----------------------
";
   for(i=0;i<s;i++)
   {
     cout<<sum[i][1]<<"   "<<sum[i][2]<<""<<sum[i][3]<<"
";
   }
    // displaying the difference matrix
   cout<<"

The difference is :

Row      Column Element";
   cout<<"
----------------------
";
   for(i=0;i<d;i++)
   {
     cout<<diff[i][1]<<"    "<<diff[i][2]<<"  "<<diff[i][3]<<"
";
   }
   getch();
   return 0;
}
```

Dictionary implimentation in C++ using Binary Trees

A dictionary in c++, where first, the dictionary is created by
taking an input by the user for the words and their meanings. Next, these
are stored in a binary search tree,aftre which the file is saved.From
here begins the actual program. Try it out!

```
#include <stdio.h>
#include<iostream.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include<dos.h>
#define LEFT  1
#define RIGHT 2

struct node
{
 char word[20],meaning[100];
 node *left,*right;
};

node *maketree(char[],char[]);

node* treefromfile();
void filefromtree(node*);
void addword(node*,char[],char[]);
void seperateword(char[],char[],char[]);
void displayall(node*);
node* bsearch(node*,char[]);
void showmenu();
FILE *file_ptr;

void prog()
{
 clrscr();
 char word[20],meaning[100];
 int menuchoice;
 node *temp;
 temp=treefromfile();
 if(temp==NULL)
 {
  printf("
File does not exist or dictionary is empty...");
  getch();
 }
 while(1)
 {
  clrscr();
  showmenu();
  scanf("
%d",&menuchoice);
  switch(menuchoice)
  {
   case 1:printf("
Enter word : ");
              scanf("%s",word);
              printf("
Enter meaning : " );
```

```
            flushall();
            gets(meaning);
            if(temp==NULL)
             temp=maketree(word,meaning);
            else
             addword(temp,word,meaning);
            break;
  case 2:if(temp==NULL)
             printf("
The dictionary is empty...");
            else
            {
             printf("
Find meaning of : ");
             flushall();
             gets(word);
             node *t;
             t=bsearch(temp,word);
             if(t==NULL)
              printf("
Word not found...");
             else
             {
              printf("

%s : ",t->word);
              puts(t->meaning);
             }
            }
            getch();
            break;
  case 3:if(temp==NULL)
             printf("
Dictionary is empty...");
            else
             displayall(temp);
            getch();
            break;
  case 4:filefromtree(temp);
            exit(1);
            break;
  default:cout<<"



Enter Again";
             delay(1000);
             prog();
             break;
 }
 }
}
void showmenu()
{
 printf("
                    COMPUTER DICTIONARY");
 printf("
```

```
[1].        Add a word.");
 printf("
[2].        Find meaning.");
 printf("
[3].        Display all.");
 printf("
[4]. Save and Close.



Enter Choice");
}
node* treefromfile()
{
 node *ptree=NULL;
 char word[20],meaning[100],str[120],*i;
 int flags=0;
 file_ptr=fopen("C:\dict.anu","r");
 if(file_ptr==NULL)
  ptree=NULL;
 else
 {

  while(!feof(file_ptr))
  {
            i=fgets(str,120,file_ptr);
            if(i==NULL)
            break;
            seperateword(str,word,meaning);
            if(flags==0)
            {
             ptree=maketree(word,meaning);
             flags=1;
            }
            else
             addword(ptree,word,meaning);
  }

  fclose(file_ptr);
 }
 return ptree;
}
node* maketree(char w[],char m[])
{
 node *p;
 p=new node;
 strcpy(p->word,w);
 strcpy(p->meaning,m);
 p->left=NULL;
 p->right=NULL;
 return p;
}
void seperateword(char str[],char w[],char m[])
{
 int i,j;
 for(i=0;str[i]!=' ';i++)
  w[i]=str[i];
```

```
 w[i++]=NULL;    //Append the null and skip the space.
 for(j=0;str[i]!='
';i++,j++)
 {
  m[j]=str[i];
 }
 m[j]=NULL;
}
void addword(node *tree,char word[],char meaning[])
{
 node *p,*q;
 p=q=tree;
 while(strcmp(word,p->word)!=0 && p!=NULL)
 {
  q=p;
  if(strcmp(word,p->word)<0)
   p=p->left;
  else
   p=p->right;
 }
 if(strcmp(word,q->word)==0)
 {
  printf("
This word already exists...");
  delay(1000);
 }
 else if(strcmp(word,q->word)<0)
  q->left=maketree(word,meaning);
 else
  q->right=maketree(word,meaning);
}
node* bsearch(node *tree,char word[])
{
 node *q;
 q=tree;
 while(q!=NULL)
 {
  //p=q;
  if(strcmp(word,q->word)<0)
   q=q->left;
  else if(strcmp(word,q->word)>0)
   q=q->right;
  if(strcmp(word,q->word)==0)
   break;
 }
 return q;
}
void filefromtree(node *tree)
{
 void travandwrite(node*);
 file_ptr=fopen("C:\dict.anu","w");
 if(file_ptr==NULL)
 {
  printf("
Cannot open file for writing data...");
 }
 else //if(tree==NULL)
```

```
 {
  if(tree!=NULL)
  {
   travandwrite(tree);
  }
  fclose(file_ptr);  //Close the file anyway.
 }
}
void travandwrite(node *tree)
{
 if(tree!=NULL)
 {
  fprintf(file_ptr,"%s %s
",tree->word,tree->meaning);
  travandwrite(tree->left);
  travandwrite(tree->right);
 }
}
void displayall(node *tree)
{
 if(tree!=NULL)
 {
  displayall(tree->left);
  printf("%s : %s
",tree->word,tree->meaning);
  displayall(tree->right);
 }
}

void intro()
{
int i;
clrscr();
gotoxy(20,20);
cout<<"DICTIONARY LOADING";
for(i=0;i<50;i++)
{
 gotoxy(15+i,21);
 cout<<"???";
 gotoxy(20,22);
 cout<<2*i<<"% completed";
 delay(150);
}
gotoxy(20,20);
cout<<"DICTIONARY LOADING COMPLETED";
clrscr();
}
void main()
{
clrscr();
intro();
prog();
}
```

Appending two linked list based upon two data members
of individual linked list objects.

Code :

```
class CClass1
{
public:
	char mStringData[10];;

	long int mDataMember1;
	long int mDataMember2;
	CClass1 *structpNextValue;

	void SetValue(CString string, long int a, long int b)
	{
		strcpy(mStringData, string);
		mDataMember1 = a;
		mDataMember2 = b;
	}
	CClass1(void);
	~CClass1(void);
};
//////////////////////////////////////////////////////////////////////
class CClass2
{
public:

	char mStringData[10];
	long int mDataMember1;
	long int mDataMember2;

	CClass2 *structpNextValue;

	void SetValue(CString string, long int a, long int b)
	{
		strcpy(mStringData, string);
		mDataMember1 = a;
		mDataMember2 = b;
	}

	CClass2(void);
	~CClass2(void);
};
//////////////////////////////////////////////////////////////////////

	 CClass1	*pstrTemp;

		lTemp = lNumOrphanRecord;

		pstrTemp = (CClass1*)malloc(sizeof(CClass1));
		pstrTemp->structpNextValue = NULL;

		CClass2 *pstrExcTemp = &mObject2[0];

		while(lTemp > 0)
		{
			pstrTemp->mDataMember1		= pstrExcTemp->mDataMember1;
			strcpy(pstrTemp->mStringData, pstrExcTemp->mStringData);
			pstrTemp->mDataMember2		= pstrExcTemp->mDataMember2;
```

```
                pstrExcTemp = pstrExcTemp->structpNextValue;


                if (mObject1->mDataMember1 == 0)
                {
                        mObject1 = pstrTemp;
                }
                else
                {
                        CClass1 *pstrPrev = NULL;
                        CClass1 *pstrCurr = mObject1;
                        long int tempSeqNum = mObject1->mDataMember1;
                        int Icount=0;

                        while ( (pstrCurr) &&(pstrCurr->mDataMember1 !=0) &&
(pstrCurr->mDataMember1 < pstrTemp->mDataMember1) )
                        {
                                pstrPrev = pstrCurr;
                                pstrCurr = pstrCurr->structpNextValue;
                        }

                        if ((pstrCurr) && (pstrCurr->mDataMember1 ==
pstrTemp->mDataMember1) )
                        {
                                if (pstrCurr->mDataMember2 < pstrTemp->mDataMember2)
                                {
                                        pstrTemp->structpNextValue = pstrCurr->structpNextValue;
                                        free(pstrCurr);
                                        pstrCurr = pstrTemp;

                                        if (tempSeqNum == pstrTemp->mDataMember1)
                                        {
                                                mObject1 = pstrCurr;
                                        }
                                        if(pstrPrev)
                                        {
                                                pstrPrev->structpNextValue = pstrCurr;
                                        }
                                }
                                if(!pstrTemp)
                                        pstrTemp = NULL;

                                lNumOrphanRecord--;

                        }
                else
                {
                        if (pstrPrev)
                        {
                                pstrPrev->structpNextValue = pstrTemp;
                                pstrTemp->structpNextValue = pstrCurr;
                        }
                        else
                        {
                                pstrTemp->structpNextValue = pstrCurr;
                                mObject1 = pstrTemp;
```

```
                        }
                }
                }
                pstrTemp = (CClass1*)malloc(sizeof(CClass1));
                pstrTemp->structpNextValue = NULL;
                lTemp--;
        }

        lNumRecord += lNumOrphanRecord;
        pstrExcTemp = &mObject2[0];
        pstrExcTemp = pstrExcTemp->structpNextValue;

        while(mObject2->structpNextValue != NULL)
        {
                pstrExcTemp = mObject2->structpNextValue;
                mObject2->structpNextValue = pstrExcTemp->structpNextValue;
      if(!pstrExcTemp)
             pstrExcTemp = NULL;
        }
```

This program will take two doubly-linked lists of nodes and merges them into another doubly-linked list of nodes.

Code :

```
#include<iostream>
using namespace std;
class Node
{
 public:
   int data;
   Node *next;
   Node *prev;
   Node()
   {
    data=0;
    next=prev=NULL;
   }
};
class Node1
{
   public:
     int data;
     Node1 *next;
     Node1 *prev;
     Node1()
     {
       data=0;
       next=prev=NULL;
     }
};
class Node2
{
  public:
    int data;
    Node2 *next;
```

```
    Node2 *prev;
    Node2()
    {
       data=0;
       next=prev=NULL;
    }
};
class DoublyLinkedList
{
   private:
     Node *headNode, *tailnode;
     Node1 *headNode1, *tailNode1;
     Node2 *headNode2, *tailNode2;
   public:
        DoublyLinkedList()
        {
          headNode=tailNode=NULL;
          headNode1=tailNode1=NULL;
          headNode2=tailNode2=NULL;
        }
       void Insert();//Insert data into the two lists
       void Merge() ;//Merging two lists into one
       void Display();//Display only the merged list
};
void DoublyLinkedList::Insert()
{
   char option;
   //This section inserts elements into the nodes of the first list
   do
   {
    Node *newnode = new Node();
    cin>>newnode->data;
    newnode->next=NULL;
    newnode->prev=NULL;
    if(headNode==NULL)
    {
     headNode= tailNode=newnode;
    }
    else
    {
     Node *curr = headNode;
     while(curr->next!=NULL)
      {
        curr=curr->next;
      }
      curr->next=tailNode=newnode;
      newnode->prev=curr;
    }
    cout<<"Enter y to continue adding data into the first list :";
    cin>>option;
   }
   while(option=='y'||option=='Y');

   //The section inserts the elements into the nodes of the second
list
   do
   {
```

```
    Node1 *newnode = new Node1();
    cin>>newnode->data;
    newnode->next=NULL;
    newnode->prev=NULL;
    if(headNode1==NULL)
    {
     headNode1=tailNode1=newnode;
    }
    else
    {
     Node1 *curr = headNode1;
     while(curr->next!=NULL)
      {
       curr=curr->next;
      }
      curr->next= tailNode1=newnode;
      newnode->prev=curr;
     }
     cout<<"Enter y to continue adding data into the second list :";
     cin>>option;
   }
   while(option=='y'||option=='Y');
}
void DoublyLinkedList::Merge()
{
  Node *currentNode=headNode;
  Node1 *currentNode1=headNode1;
  //This section of code copies all the data from list 1 into the new
list
  while(currentNode!=NULL)
  {
    Node2 *newnode = new Node2();
    newnode->data = currentNode->data;
    newnode->next = NULL;
    newnode->prev = NULL;
    if(headNode2==NULL)
    {
     headNode2=tailNode2=newnode;
    }
    else
    {
     Node2 *temp = headNode2;
     while(temp->next!=NULL)
     {
       temp=temp->next;
     }
     temp->next=tailNode2=newnode;
     newnode->prev=temp;
    }
     currentNode=currentNode->next;
   }

   //This section of code appends the new list with data from the
second
list
   while(currentNode1!=NULL)
   {
```

```
    Node2 *newnode = new Node2();
    newnode->data = currentNode1->data;
    newnode->next = newnode->prev=NULL;
    if(tailNode2->next==NULL)
    {
      tailNode2->next=newnode;
      newnode->prev = tailNode2;
      tailNode2=newnode;
    }
    currentNode1= currentNode1->next;
  }
}
void DoublyLinkedList::Display()
{
   Node2 *currentNode2 = headNode2;
   while(currentNode2!=NULL)
   {
     cout<<currentNode2->data<<" ";
     currentNode2=currentNode2->next;
   }
   cout<<endl;
}

void main()
{
   DoublyLinkedList DLL;
   DLL.Insert();
   DLL.Merge();
   DLL.Display();
}
```