Experiment #8

Doubly Linked lists

Student's Name:	
Semester:	Date:

Assessment:

Assessment Point	**Weight**	**Grade**
Methodology and correctness of results		
Discussion of results		
Participation		
Assessment Points' Grade:		

Comments:

Experiment #8:

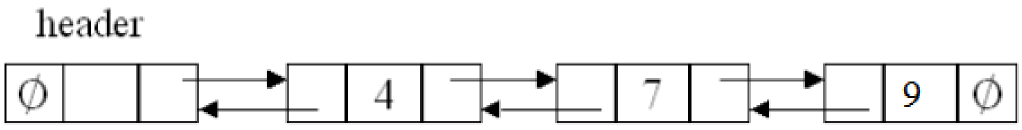
Doubly linked lists in C++ Programming Language

Objectives:

1. To introduce the students with the concept of doubly linked lists
2. To implement doubly linked lists
3. To implement different operations on doubly linked lists

Discussion:

Doubly linked list is a linked data structure that consists of a set of sequentially linked nodes. Each node contains three fields, the data field, the **prev** pointer references to the previous node and **next** pointer references to the next node. The dummy head nodes references to the first node in the list with no data stored in and a null previous pointer.



Doubly Linked list implementation

```
// double link list using templates
// programmed by Dr.aryaf aladwan


#include <iostream.h>
template <class T>
class doublylinklist
{
    private:
    template <class T>
        struct node
        {
          T data;
          node <T>*next;
```

```
                         node <T>*prev;
                       };
      node <T> *head;
   public:
         doublylinklist();
         void insert( T num );
         void add_as_first( T num );
           void display();
            int count();
         void addafter( T c, T num );
          T del( T num );
         ~doublylinklist();
};

template <class T>
doublylinklist<T>::doublylinklist()
{
     head = new node<T>;
            head->next=NULL;
            head->prev=NULL;
}

template <class T>
void doublylinklist<T>::insert(T num)
{
     node <T>*q,*t;

   if( head ->next==NULL )    // insert into empty list
   {
      q = new node <T>;
      q->data = num;
      q->next = NULL;
             q->prev=head;
            head->next=q;

   }
   else                                              // append
   {
        q = head;
      while( q->next != NULL )
           q = q->next;
      t = new node <T>;
      t->data = num;
      t->next= NULL;
            t->prev=q;
      q->next = t;
   }
}

template <class T>
void doublylinklist<T>::add_as_first(T num) // insert in the beginning
{
     node <T>*q;
   q = new node <T>;
   q->data = num;
   q->prev=head;
   q->next=head->next;
   head->next->prev=q;
   head->next=q;
}

template <class T>
int doublylinklist<T>::count()
{
    node <T>*q;
   int c=0;
```

```
  for( q=head->next ; q != NULL ; q = q->next)
      c++;
  return c;
}

template <class T>
void doublylinklist<T>::addafter( T c, T num)  // insert in the middle
{
  node <T> *q,*t;
  int i;
  for(i=1,q=head->next;i<c;i++)
  {
      q = q->next;
  }
  t = new node <T>;
  t->data = num;
  t->prev=q;
  t->next=q->next;
  q->next->prev=t;
  q->next=t;
}

template <class T>
T doublylinklist<T>::del( T num )
{
  node <T>*q;
  q = head->next;
  if( q->data == num )   // delete from the beginning
  {
    head->next = q->next;
         q->next->prev=q->prev;
    delete q;
    return 0;
```

```
   }
   while( q->next!=NULL )      // delete from middle
   {
     if( q->data == num )
     {
         q->prev->next=q->next;
         q->next->prev=q->prev;
       delete q;
       return 0;
     }
    q=q->next;

   if(q->data==num && q->next==NULL)   // delete from end
                 {
                 T z;
                 z=q->data;
                 q->prev->next=NULL;
                 delete q;
                 return z;
                 }
   }
   cout<<"\nElement "<<num<<" not Found.";
}

template <class T>
doublylinklist<T>::~doublylinklist()
{
    node <T>*q;
   if( head == NULL )
       return;

   while( head != NULL )
   {
        q = head->next;
      delete head;
      head = q;
   }
}

template <class T>
void doublylinklist<T>::display()
{
     node <T>*q;
   cout<<endl;
   for( q = head->next ; q != NULL ; q = q->next )
        cout<<endl<<q->data;
}


int main()
{
   doublylinklist <int>dl;
   dl.insert(12);
   dl.insert(13);
   dl.insert(23);
   dl.insert(43);
   dl.insert(44);
   dl.insert(50);
   dl.add_as_first(2);
   dl.add_as_first(111);
   cout<<"No. of elements = "<<dl.count();
   dl.addafter(2,333);
   dl.addafter(6,666);
   dl.display();
   cout<<"\nNo. of elements = "<<dl.count();
```

```
  dl.del(333);
  dl.del(50);
  dl.del(98);
  cout<<"\nNo. of elements = "<<dl.count();
   dl.display();
  return 0;
}
```

Exercise 1:

Write a c++ program to search the doubly linked list with an integer number, if it is found then multiply the number with 3, if it is not found then print “not found”?

Solution to Exercise 1

Output

Exercise 2:

write and test a method public void reverse() to reverse the order of the nodes in the doubly linked list. E.g. if the list a->b->c->d the call of the method reverse() will rearrange the list as d->c->b->a.

Solution to Exercise 2

Output