Experiment #7

Linked Stack and Linked Queue

Student's Name:	
Semester:	Date:

Assessment:

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

Comments:

Experiment #7:

Linked stack and linked queue in C++ Programming Language

Objectives:

1. To introduce the students with the concept of linked stack and linked queue
2. To implement linked stack and linked queue
3. To understand the advantage of link list-based implementation for the stack and the queue

Discussion:

Stacks can be implemented using link lists, where the insertion and deletion can be performed without memory limitations. Linked stack has the following operations:

1. IsEmpty(): to check whether the stack is empty or not using the head pointer.
2. Push(): insert nodes at the end of the link list.
3. Pop(): remove nodes from the end of the link list.

Also, queues can be implemented using link lists. Linked queue has the following operations:

1. IsEmpty(): to check whether the queue is empty or not using the head pointer.
2. Enqueue(): insert nodes at the end of the link list.
3. Dequeue(): remove nodes from the end of the link list.

Linked stack implementation

```
// Linked Stack using templates
// programmed by Dr.Aryaf Al-adwan

#include <iostream>
using namespace std;
template <class T>
class linkedstack
{
    private:
        template <class T>
        struct node
        {
          T data;
          node <T> *link;
                    } ;
                    node <T> *head;

  public:
        linkedstack();
                    void push(T element);
                    T pop ();
        void display();
        T count();
        ~linkedstack();
};

template <class T>
 linkedstack<T>::linkedstack()
{
    head = NULL;
}



template <class T>
void linkedstack<T>::push(T element)
{
          node <T>*q,*t;

  if( head == NULL )    // insert into empty stack
  {
     head = new node<T>;
```

```
      head->data = element;
      head->link = NULL;

   }
   else                                                    // append
   {
        q = head;
      while( q->link != NULL )
           q = q->link;

      t = new node<T>;
      t->data = element;
      t->link = NULL;
      q->link = t;
   }

}

template <class T>
T  linkedstack<T>::pop ()
{
           T x;
   if( head == NULL )    // check if the stack is empty
   {
     cout<<"empty stack";
           return 0;

   }

   else                                    // delete from the end of the stack
   {
   node <T>*q,*r;
   q = head;
```

```
  r = q;
  while( q->link!=NULL )
  {
     r = q;
    q = q->link;
  }
  r->link=NULL;
  x=q->data;
  delete q;
  return x;
  }

}

template <class T>
void linkedstack<T>::display()
{
   node <T>*q;
  cout<<endl;

  for( q = head ; q != NULL ; q = q->link )
      cout<<endl<<q->data;
}

template <class T>
T linkedstack<T>::count()
{
   node <T>*q;
  int c=0;
  for( q=head ; q != NULL ; q = q->link )
      c++;

  return c;
```

```
}

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

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

int main()
{
  linkedstack <int>ls;
  cout<<"No. of elements = "<<ls.count()<<endl;
  ls.pop();
  ls.push(12);
  ls.push(10);
  ls.push(4);
  ls.push(9);
  ls.push(20);
  ls.push(15);
  ls.display();
  cout<<"\nNo. of elements = "<<ls.count()<<endl;
  cout<<"\npop 1:"<<ls.pop();
  cout<<"\npop 2:"<<ls.pop();
  cout<<"\npop 3:"<<ls.pop();
```

```
   cout<<" \npop 4:"<<ls.pop();
   cout<<"\nNo. of elements = "<<ls.count();
   cout<<"\n\nthe final stack";
   ls.display();
   return 0;
}
```

Linked Queue implementation

```
// Linked Queue using templates
// programmed by Dr.Aryaf Al-adwan

#include <iostream.h>
template <class T>
class LinkedQueue
{
     private:
         template <class T>
         struct node
         {
           T data;
           node <T> *link;
                      } ;
                      node <T> *head;
   public:
         LinkedQueue();
            void enequeue(T element);
           T dequeue();
         void display();
         T count();
         ~LinkedQueue();
};
```

```
template <class T>
 LinkedQueue<T>::LinkedQueue()
{
     head = NULL;
}

template <class T>
void LinkedQueue<T>::enequeue(T element)
{
     node <T>*q,*t;
   if( head == NULL )    // insert into empty queue
   {
      head = new node<T>;
      head->data = element;
      head->link = NULL;
   }
   else                                                  // append
   {
        q = head;
      while( q->link != NULL )
           q = q->link;

      t = new node<T>;
      t->data = element;
      t->link = NULL;
      q->link = t;
   }
}

template <class T>
T LinkedQueue<T>::dequeue( )
{
   node <T>*q;
```

```
  T x;
  q = head;
  if(head==NULL)          // check if the queue is empty
  {
         cout<<"empty queue";
     return 0;
  }
  else
  {
  head = q->link;         // delete from the beginning of the queue
  x = q->data;
  delete q;
  return x;
  }
}

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

template <class T>
T LinkedQueue<T>::count()
{
   node <T>*q;
  int c=0;
  for( q=head ; q != NULL ; q = q->link )
      c++;
   return c;
}

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

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

int main()
{
   LinkedQueue <int>lq;
   cout<<"No. of elements = "<<lq.count()<<endl;
   lq.dequeue();
   lq.enequeue(12);
   lq.enequeue(10);
   lq.enequeue(7);
   lq.enequeue(11);
   lq.enequeue(17);
   lq.enequeue(4);
   lq.display();
           cout<<"\nNo. of elements = "<<lq.count();
           cout<<"\ndeque 1: "<<lq.dequeue();
           cout<<"\ndeque 2: "<<lq.dequeue();
```

```
          cout<<"\ndeque 3: "<<lq.dequeue();
          cout<<"\ndeque 4: "<<lq.dequeue();
          cout<<"\nNo. of elements = "<<lq.count();
          cout<<"\n\nthe final queue";
    lq.display();
   return 0;
}
```

Exercise 1:

Write a C++ program to create a **linked stack data structure**. This Stack data structure stores integer values. Your program should display a menu of choices to operate the Stack data structure. See the sample menu below:

```
==================================================
                    Stack  Operations Menu
==================================================
1. Add items

2. Delete items

3. Show the number of items

4. Show min and max items

5. Find an item

6. Print all items

7. Exit


Enter your choice:1
```

Solution to Exercise 1

Output