Lab Manual for Data Structures

Lab 7

Linked Lists as Stack and Queue

Table of Contents

Lab12: Linked Lists as Stack and Queue

1. Introduction

Objective of this lab is to make you understand the usage of linked list structure for implementation of stack and queue ADT (Abstract Data Types). If we need to maintain a stack whose individual elements are allocated memory space during program execution, we need to use linked list for that purpose, same is case for queue. In case of array based stacks and queues we may only store finite number of elements which will be limited to maximum size of array.

1.1 Linked Lists as Stack

Stack can be implemented by linked lists structure. In this case we need to maintain a pointer to node of linked list which always point to a node before the top of stack, when an element is inserted in list it is placed at top of stack and when an element is removed it is also removed from top of stack. In this way stacks can be used by linked list structures. As memory of elements of stack is dynamically allocated so stack will never be filled (we don't need to check for stack overflow conditions). Figure 1 provides information about the use of stack as a linked list structure.

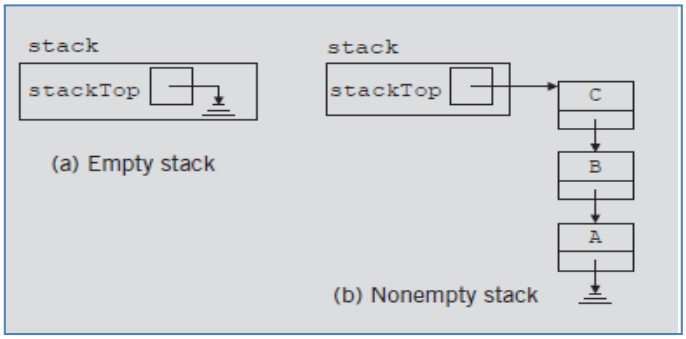

Figure 1: Use of stack as a linked list structure.

1.2 Linked Lists as Queue

Because the size of the array to stock up the queue elements is fixed, only a finite number of queue elements can be stored in the array. Also, the array implementation of the queue requires the array to be handled in a special way together with the values of the indices queueFront and queueRear. The linked list implementation of a queue simplifies many of the special cases of the array implementation and, because the memory to store a queue element is allocated dynamically, the queue is never full. It means we don't need to check for queue overflow condition, we need to maintain only queue underflow condition that is we need to check whether queue is empty or not while removing an element from the queue.

1.3Relevant Lecture Readings

a) Revise Lecture No. 23 and 24 available at \\dataserver\jinnah$\ in instructor's folder.
b) From books: C++ Data Structures by Nell Dale (Page280 - 317) and Data structures using C++ by D. S. Malik (Page415 - 423).

2. Activity Time boxing

Table 1: Activity Time Boxing

Task No.	Activity Name	Activity time	Total Time
5.1	Design Evaluation	20mins	25mins
6.2	Walk through tasks	20mins	25mins
7	Practice tasks	30mins for task 1 and 30 mins for task 2	60mins
9	Evaluation Task	60mins for all assigned tasks	60mins

3. Objectives of the experiment

- To understand and implement stacks using linked lists with their operations in C++.
- To understand and implement queues using linked lists with their operations in C++.

4. Concept Map

This concept map will help students to understand the main concepts of topic covered in lab.

4.1 Linked Lists as Stack in C++

The first operation that isto be considered is the default constructor. The default constructor initializes the stack to an empty state when an object of stack class is declared. Thus, this function sets stackTop pointer to NULL. The definition of this function is as follows:

```
template<class Type>
linkedStackType<Type>::linkedStackType()
{
stackTop = NULL;
}
```

The operations isEmptyStack and isFullStack are easy to understand. The stack is empty if stackTop is NULL. Also, because the memory for elements of stack is allocated and deallocated dynamically, the stack is never full. Thus, the function isFullStack always returns the value false. The definitions of the functions to implement these operations are asfollows:

```
template<class Type>
bool linkedStackType<Type>::isEmptyStack() const
{
return(stackTop == NULL);
```

```
} //end isEmptyStack
template<class Type>
bool linkedStackType<Type>::isFullStack() const
{
return false;
} //end isFullStack
```

Push operation for stack can be defined by following code segment.

```
template<class Type>
void linkedStackType<Type>::push(const Type &newElement)
{
nodeType<Type> *newNode; //pointer to create the new node
newNode = new nodeType<Type>; //create the node
newNode->info = newElement; //store newElement in the node
newNode->link = stackTop; //insert newNode before stackTop
stackTop = newNode; //set stackTop to point to the
//top node
} //end push
```

Pop operation for stack can be defined by following code segment.

```
template<class Type>
void linkedStackType<Type>::pop()
{
nodeType<Type> *temp; //pointer to deallocate memory
if(stackTop != NULL)
{
temp = stackTop; //set temp to point to the top node
stackTop = stackTop->link; //advance stackTop to the
//next node
delete temp; //delete the top node
}
else
cout<< "Cannot remove from an empty stack." <<endl;
}//end pop
```

4.2Linked Lists as Queue in C++

Some operations related to implementation of queue as a linked list are provided in this section. The queue is empty if queueFront pointer is NULL. Memory to store the queue elements is allocated dynamically. Therefore, the queue is never full and so the function to implement the isFullQueue operation returns the value false. Implementation of functions for checking empty or full queue is given below:

```
template<class Type>
boollinkedQueueType<Type>::isEmptyQueue() const
```

```
{
return(queueFront == NULL);
} //end
template<class Type>
bool linkedQueueType<Type>::isFullQueue() const
{
return false;
} //end isFullQueue
```

The addQueue operation inserts a new element at the rear of the queue. To implement this operation, we will use the pointer queueRear. If the queue is non-empty, the operation front returns the first element of the queue and therefore the element of the queue indicated by the pointer queueFront is returned. If thequeue is empty, the function front stops execution of program. If the queue is nonempty, the operation back returns the last element of the queue and so the element of the queue indicated by the pointer queueRear is returned. If the queue isempty, the function back stops the execution of program. Similarly, if the queue is nonempty, the operation deleteQueue removes the first element of the queue, and so we access the pointer queueFront for this purpose. The definitions of the functions to implement these operations are as follows:

```
template<class Type>
void linkedQueueType<Type>::addQueue(const Type&newElement)
{
nodeType<Type> *newNode;
newNode = new nodeType<Type>; //create the node
newNode->info = newElement; //store the info
newNode->link = NULL; //initialize the link field to NULL
if (queueFront == NULL) //if initially the queue is empty
{
queueFront = newNode;
queueRear = newNode;
}
else //add newNode at the end
{
queueRear->link = newNode;
queueRear = queueRear->link;
}
}//end addQueue

template<class Type>
Type linkedQueueType<Type>::front() const
{
assert(queueFront != NULL);
return queueFront->info;
} //end front

template<class Type>
Type linkedQueueType<Type>::back() const
```

```
{
assert(queueRear!= NULL);
return queueRear->info;
} //end back

template<class Type>
void linkedQueueType<Type>::deleteQueue()
{
nodeType<Type> *temp;
if (!isEmptyQueue())
{
temp = queueFront; //make temp point to the first node
queueFront = queueFront->link; //advance queueFront
delete temp; //delete the first node
if (queueFront == NULL) //if after deletion the
//queue is empty
queueRear = NULL; //set queueRear to NULL
}
else
cout<< "Cannot remove from an empty queue" <<endl;
}//end deleteQueue
```

5. Homework before Lab

This homework will help students to study the concepts of topic before start of lab.

5.1 Problem Solution Modeling

After studying the introduction and concept map sections you should be ready to provide the solution of following problems. Design the solutions of the problems in C++ and bring your code to lab so that lab instructor should assess and grade it.

5.2 Problem description:

Write a program of linear linked list to implement a stack and a queue of objects of "person" class. Attributes of "person" class (privately defined) are per_id (int), per_name (string) and per_age (int). "person" class contains member functions (publicly defined): constructor, input and output functions. You are required to define two more classes: one of these classes should define the linked list implementation of stack for objects of "person" class, this class should contain member functions like push, pop, display and to check whether stack is empty or not? Other class should define the linked list implementation of queue for objects of "person" class, this class should contain the member functions like addQueue, RemoveQueue, display and check whether the queue is empty or not?

5.3Practices from home

5.3.1 Task-1

Compare linked list implementations of stack and queue and find at least three differences between these implementations.

5.3.2 Task-2

Provide one example of stack and one of queue from real world where linked list implementations are useful.

6. Procedure& Tools

This section provides information about tools and programming procedures used for the lab.

6.1 Tools

Microsoft Visual Studio 2017 with Visual C++ compiler configured.

6.2 Walk through Tasks [Expected time = 25mins]

Following screens in figure 2, 3 and 4 represent source code implementation of a linear linked list classas a stack in Microsoft Visual Studio 2017. You are required to type, debug and execute this program in Microsoft Visual Studio 2017.

This linked list implementation of stack program provides functionality to use a stack. It contains a structure to represent the nodes of linked list and a class named “stack” which will be representing the stack as linked list. This class contains constructor, push, pop and display member functions. In main function we create an object of stack class and call member functions of class to push and pop elements from the stack.

```
#include<iostream>
#include<cstdlib>
#include<malloc.h>
#include<conio.h>
using namespace std;

struct node{
    int info;
    struct node *next;
    };

class stack{
    struct node *top;
    public:
        stack();
        void push();
        void pop();
        void display();
    };

stack::stack(){
    top = NULL;
    }
```

Figure 2: Implementation of stack as a linked list.

```
void stack::push(){
    int data;
    struct node *p;
    if((p=(node*)malloc(sizeof(node)))==NULL){
        cout<<"Memory Exhausted";
        exit(0);
        }
    cout<<"Enter a Number to insert:";
    cin>>data;
    p = new node;
    p->info = data;
    p->next = NULL;
    if(top!=NULL){
        p->next = top;
        }
    top = p;
    cout<<"\nNew item inserted"<<endl;
    }

void stack::pop(){
    struct node *temp;
    if(top==NULL){
        cout<<"\nThe stack is Empty"<<endl;
        }
    else{
        temp = top;
        top = top->next;
        cout<<"\nThe value popped is "<<temp->info<<endl;
        delete temp;
        }
    }
```

Figure 3: Implementation of stack as a linked list.

```
void stack::display(){
    struct node *p = top;
    if(top==NULL){
        cout<<"\nNothing to Display\n";
        }
    else{
        cout<<"\nThe contents of Stack\n";
        while(p!=NULL){
        cout<<p->info<<endl;
        p = p->next;
        }
    }
}

int main(){
    stack s;
    int choice;
    do{
        cout<<"\nEnter your choice:";
        cout<<"\n1. PUSH\n2. POP\n3. DISPLAY\n4. EXIT\n";
        cin>>choice;
        switch(choice){
            case 1: s.push(); break;
            case 2: s.pop();  break;
            case 3: s.display(); break;
            case 4: exit(0); break;
            default:cout<<"Invalid Choice";break;

            }
       }while(choice);
    getch();
    return 0;
    }
```

Figure 4: Implementation of stack as a linked list.

7. Practice Tasks

This section will provide information about the practice tasks which are required to be performed in lab session. Design solutions of problems discussed in each task and placesolution code in a folder specified by your lab instructor.

Lab instructors are guided to help students learn how ACM problems work and provide students with certain practice/home tasks on the pattern of ACM Problems.

7.1 Practice Task 1 [Expected time = 30mins]

Write a program to implement a stack using linked list. Stack is of Books (Title, Author, Price). your program should find or delete a book entered by user from the stack. Remember that to reach a book in stack all the books above must be checked first.

7.2 Practice Task 2 [Expected time = 30mins]

Organization ABC is taking online orders from its customers. Save the following information for each order:

OrderID (Auto incremented)

Customer name

Address

Number of items in the order

Payment of order

Orders will be processed on first come first server basis.

7.3Out comes

After completing this lab, student will be able to understand and develop programs related to linked list implementation of stacks and queuesin C++ using Microsoft Visual Studio 2017 environment.

7.4Testing

Test Cases for Practice Task-1

Sample Input	Sample Output
Enter a string:	
Evil olive	
	Entered string is palindrome
Enter a string:	
Computer	
	Entered string is not palindrome

Test Cases for Practice Task-2
Enter order details:
Customer name: Ali
Items:4
Total cost:10000

Customer name: Ahmed
Items:2
Total cost:1000

Customer name: Zain
Items:7
Total cost:25000

Customer name: Mariya
Items:1
Total cost:5000

Press 1 for case 1
Press 2 for case 2
Enter choice:1
Processed orders:
Customer name: Ali
Items:4
Total cost:10000

Customer name: Ahmed
Items:2
Total cost:1000

Customer name: Zain
Items:7
Total cost:25000

Customer name: Mariya
Items:1
Total cost:5000

8.Evaluation Task (Unseen) **[Expected time = 60mins for two tasks]**

The lab instructor will give you unseen task depending upon the progress of the class.

9. Evaluation criteria

The evaluation criteria for this lab will be based on the completion of the following tasks. Each

task is assigned the marks percentage which will be evaluated by the instructor in the lab whether the student has finished the complete/partial task(s).

Table 2: Evaluation of the Lab

Sr. No.	Task No	Description	Marks
1	4	Problem Modeling	20
2	6	Procedures and Tools	10
3	7,8	Practice tasks and Testing	35
4	8.1	Evaluation Tasks (Unseen)	20
5		Comments	5
6		Good Programming Practices	10

10. Further Readings

10.1 Web sites related to C++ tutorials about linked list implementation of stack and queues

1. http://www.programming-techniques.com/2011/11/stack-and-queue-implementation-of.html
3. http://stackoverflow.com/questions/3600245/fifo-queue-linked-list-implementation

10.2 Web sites containing supporting material

1. http://www.cs.cmu.edu/~rjsimmon/15122-s13/09-queuestack.pdf