RESEARCH PAPER ON STACK AND QUEUE

Nitesh, Manbir Singh, Rahul Yadav
Deparment Of Electronics And Communication
Dronacharya College Of Engineering
Farruknagar, Khetawas, Gurgaon

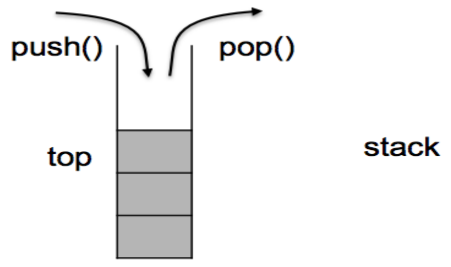
***Abstract*- This paper involves the concept of stack and queue used in data structure basically two of the more common data objects found in computer algorithms are stacks and queues. Both of these objects are special cases of the more general data object, an ordered list.**

A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle. In the pushdown stacks only two operations are allowed: push the item into the stack, and pop the item out of the stack. A stack is a limited access data structure - elements can be added and removed from the stack only at the top. push adds an item to the top of the stack, pop removes the item from the top. If we talk about the daily life example like a stack of books; you can remove only the top book, also you can add a new book on the top.

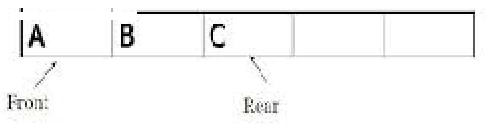
I. INTRODUCTION

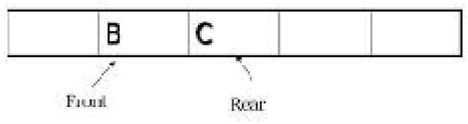
STACK Stop. Since the last item added to the list is the first removed from the list, stacks are also known as “Last A stack is an ordered list of items. Items are added to the list at the top and items are removed from the In First Out” (LIFO) lists. A stack is easily implemented in an array, requiring only a

pointer variable to point to the position of the top element of the stack.



queue
A *queue* is an ordered list in which all insertions take place at one end, the *rear*, while all deletions take place at the other end, the *front*. Given a stack S=(a[1],a[2],.......a[n]) then we say that a1 is the bottommost element and element a[i]) is on top of element a[i-1], 1<i<=n. When viewed as a queue with a[n] as the rear element one says that a[i+1] is behind a[i], 1<i<=n.queue operation are more helpful as campared to stack operation.





The restrictions on a stack imply that if the elements A,B,C,D,E are added to the stack, n that order, then thefirst element to be removed/deleted must be E. Equivalently we say that the last element to be inserted into the stack will be the first to be removed. For this reason stacks are sometimes referred to as Last In First Out (LIFO) lists. The restrictions on queue imply that the first element which is inserted into the queue will be the first one to be removed. Thus A is the first letter to be removed, and queues are known as First In First Out (FIFO) lists. Note that the data object queue as defined here need not necessarily correspond to the mathemathical concept of queue in which the insert/delete rules may be different.

II. STACK ITS APPLICATIONS AND IMPLEMENTATION

Applications
- The simplest application of a stack is to reverse a word. You push a given word to stack - letter by letter - and then pop letters from the stack.
- Another application is an "undo" mechanism in text editors; this operation is accomplished by keeping all text changes in a stack.

Backtracking. This is a process when you need to access the most recent data element in a series of elements. Think of a labyrinth or maze - how do you find a way from an entrance to an exit?
Once you reach a dead end, you must backtrack. But backtrack to where? to the previous choice point. Therefore, at each choice point you store on a stack all possible choices. Then backtracking simply means popping a next choice from the stack.
Language processing:

space for parameters and local variables is created internally using a stack.
compiler's syntax check for matching braces is implemented by using stack.
support for recursion

Implementation

In the standard library of classes, the data type stack is an *adapter* class, meaning that a stack is built on top of other data structures. The underlying structure for a stack could be an array, a vector, an ArrayList, a linked list, or any other collection. Regardless of the type of the underlying data structure, a Stack must implement the same functionality. This is achieved by providing a unique interface:

```
public interface StackInterface<AnyType>
{
public void push(AnyType e);
publicAnyType pop();
publicAnyType peek();
publicbooleanisEmpty();
}
```

The following picture demonstrates the idea of implementation *by composition*.

Another implementation requirement (in addition to the above interface) is that all stack operations must run in **constant time O(1)**. Constant time means that there is some constant k such that an operation takes k nanoseconds of computational time regardless of the stack size.

Array-based implementation

In an array-based implementation we maintain the following fields: an array A of a default size (≥ 1), the variable *top* that refers to the top element in the stack and the *capacity* that refers to the array size. The variable *top* changes from -1 to capacity - 1. We say that a stack is empty when top = -1, and the stack is full when top = capacity-1.

In a dynamic stack abstraction when *top* reaches *capacity*, we double up the stack size.

Linked List-based implementation

Linked List-based implementation provides the best (from the efficiency point of view) dynamic stack implementation.

III. QUEUE ITS APPLICATION AND IMPLEMENTIONS

A queue is a container of objects (a linear collection) that are inserted and removed according to the first-in first-out (FIFO) principle. An excellent example of a queue is a line of students in the food court of the UC. New additions to a line made to the back of the queue, while removal (or serving) happens in the front. In the queue only two operations are allowed **enqueue** and **dequeue**. Enqueue means to insert an item into the back of the queue, dequeue means removing the front item. The picture demonstrates the FIFO access.

The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added

Implementation

In the standard library of classes, the data type queue is an *adapter* class, meaning that a queue is built on top of other data structures. The underlying structure for a queue could be an array, a Vector, an ArrayList, a LinkedList, or any other collection. This isachieved by providing a unique interface.

```
interfaceQueueInterface‹AnyType>
{
publicbooleanisEmpty();
publicAnyTypegetFront();
publicAnyTypedequeue();
public void enqueue(AnyType e);
public void clear();
}
```

Each of the above basic operations must run at constant time O(1). The following picture demonstrates the idea of implementation by composition.

IV. ALGORITHM ADDING AN ELEMET IN A STACK

procedure add(item : items); {add item to the global stack stack; top is the current top of stack and n is its maximum size} **begin if** top = n **then**stackfull; top := top+1; stack(top) := item; **end:** {of add}

V. DELETION IN STACK

procedure delete(**var** item : items); {remove top element from the stack stack and put it in the item} **begin if** top = 0 **then**stackempty; item := stack(top); top := top-1; **end;** {of delete}

These two procedures are so simple that they perhaps need no more explanation. Procedure delete actually combines the functions TOP and DELETE, stackfull and stackempty are procedures which are left unspecified since they will depend upon the particular application. Often a stackfull condition will signal that more storage needs to be allocated and the program re-run. Stackempty is often a meaningful condition.

VI. ADDITION IN QUEUE

procedureaddq (item : items); {add item to the queue q} **begin if** rear=n **then**queuefull**else begin** rear :=rear+1; q[rear]:=item; **end;end;**{of addq}

VII. DELETION IN QUEUE

```
procedure deleteq (var item : items);{delete from the front of q and put into item}
begin
    if front = rear then queueempty
    else begin
        front := front+1
        item := q[front];
    end;
end; {of deleteq}
```

VIII. CONCLUSIONS

- Linear data structures maintain their data in an ordered fashion.
- Stacks are simple data structures that maintain a LIFO, last-in first-out, ordering.
- The fundamental operations for a stack are push, pop, and isEmpty.
- Queues are simple data structures that maintain a FIFO, first-in first-out, ordering.
- The fundamental operations for a queue are enqueue, dequeue, and isEmpty
- Stacks are very useful for designing algorithms to evaluate and translate expressions.
- Stacks can provide a reversal characteristic.
- Queues can assist in the construction of timing simulations.
- Simulations use random number generators to create a real-life situation and allow us to answer "what if" types of questions.
- Deques are data structures that allow hybrid behavior like that of stacks and queues.
- The fundamental operations for a deque are addFront, addRear, removeFront, removeRear, and isEmpty.
- Lists are collections of items where each item holds a relative position.
- A linked list implementation maintains logical order without requiring physical storage requirements.
- Modification to the head of the linked list is a special case.

REFERENCES

- "Queue (Java Platform SE 7)". Docs.oracle.com. 2014-03-26. Retrieved 2014-05-22.
- Donald Knuth. *The Art of Computer Programming*, Volume 1: *Fundamental Algorithms*, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89683-4. Section 2.2.1: Stacks, Queues, and Deques, pp. 238–243.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 10.1: Stacks and queues, pp. 200–204.
- William Ford, William Topp. *Data Structures with C++ and STL*, Second Edition. Prentice Hall, 2002. ISBN 0-13-085850-1. Chapter 8: Queues and Priority Queues, pp. 386–390.
- Adam Drozdek. *Data Structures and Algorithms in C++*, Third Edition. Thomson Course Technology, 2005. ISBN 0-534-49182-0. Chapter 4: Stacks and Queues, pp. 137–169.