DATA STRUCTURES

UNIT
2

By- MS.ARTI GAUTAM
A.P. (CSE),GLBITM

Department of Computer Science and Engineering

VISION

To build strong teaching environment that responds the need of industry and challenges of society.

MISSION

M1:Developing strong mathematical & computing fundamentals among the students.

M2: Extending the role of computer science and engineering in diverse areas.

M3: Imbibing the students with a deep understanding of professional ethics and high integrity to serve the nation.

M4: Providing an environment to the students for their growth both as individuals and as globally competent Computer Science professional.

M5: Outreach activities will contribute to the overall wellbeing of society.

Syllabus

Stacks: Abstract Data Type, Primitive Stack operations: Push & Pop, Array and Linked Implementation of Stack in C, Application of stack: Prefix and Postfix Expressions, Evaluation of postfix expression, Recursion, Tower of Hanoi Problem, Simulating Recursion, Principles of recursion, Tail recursion, Removal of recursion Queues, Operations on Queue: Create, Add, Delete, Full and Empty, Circular queues, Array and linked implementation of queues in C, Dequeue and Priority Queue.

Learning Objectives

The objectives of the following slide is to make student aware about the :

- Primitive operations on stack and queue
- Representation of stack and queue using array and linked list
- Applications of stack
- Types of queues
- Tower of Hanoi Problem

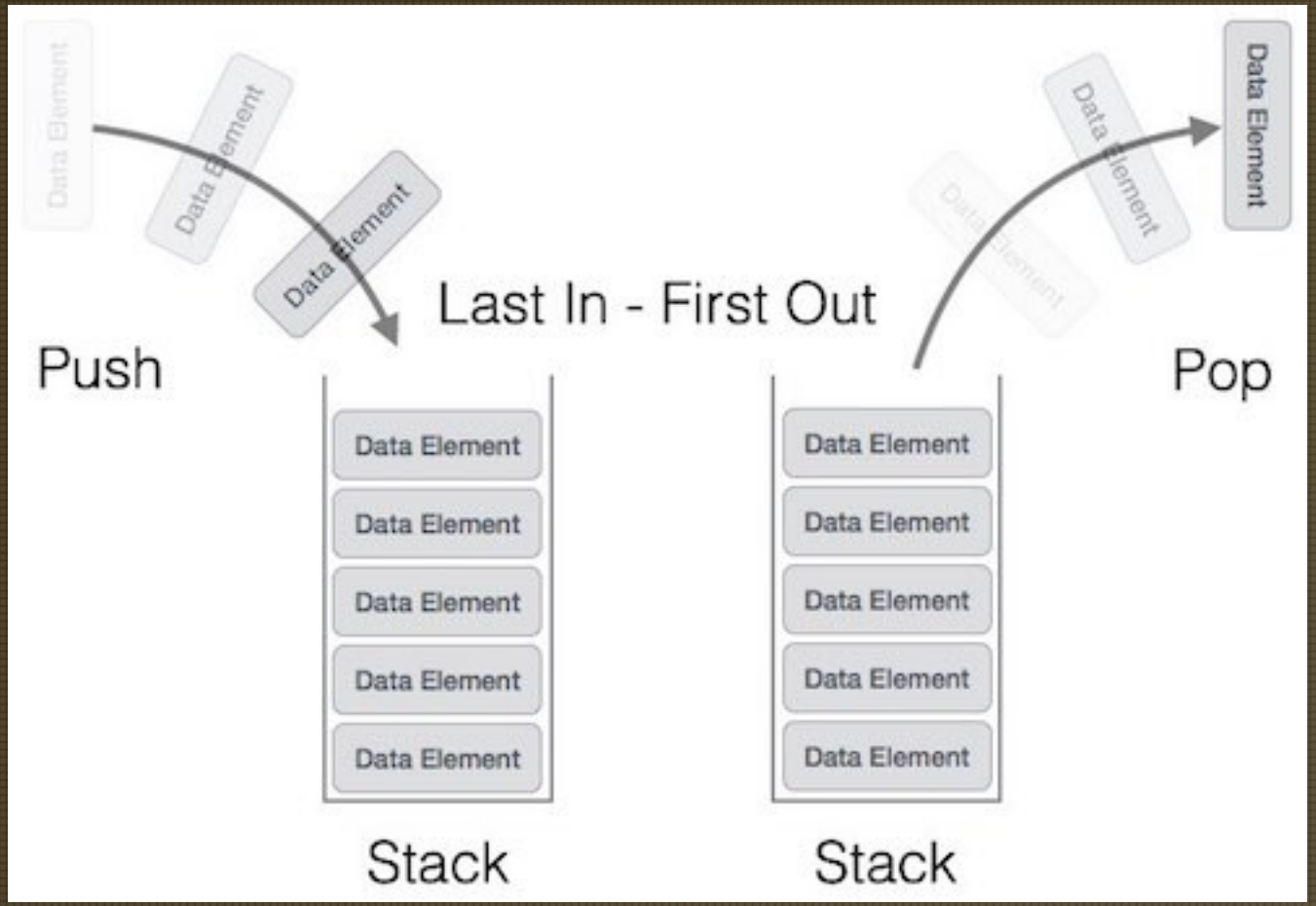
The Stack Abstract Data Type

The stack abstract data type is defined by the following structure and operations. A stack is structured, as described above, as an ordered collection of items where items are added to and removed from the end called the “top.” Stacks are ordered LIFO.

Primitive Stack Operations
Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations −
push() − Pushing (storing) an element on the stack.
pop() − Removing (accessing) an element from the stack.

Primitive Stack Operations

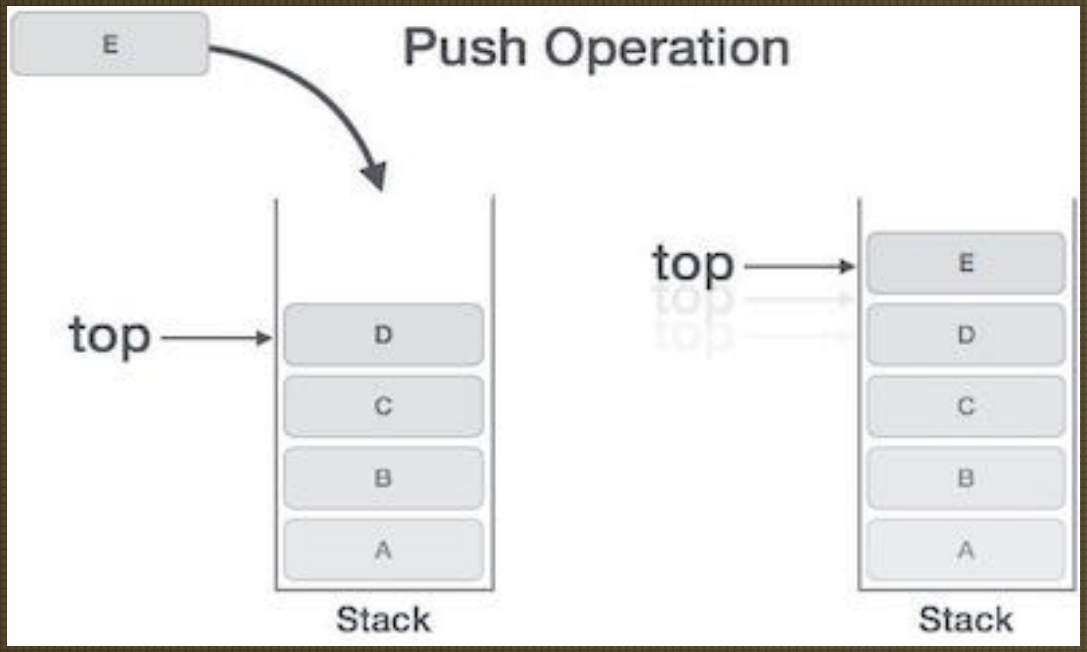
The following diagram depicts a stack and its operations −



Push Operation

The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps −

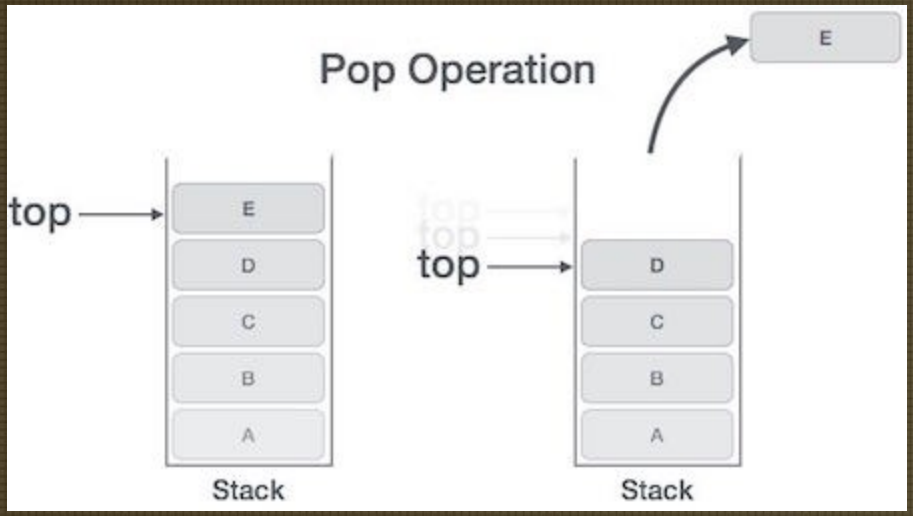
- **Step 1** − Checks if the stack is full.
- **Step 2** − If the stack is full, produces an error and exit.
- **Step 3** − If the stack is not full, increments **top** to point next empty space.
- **Step 4** − Adds data element to the stack location, where top is pointing.
- **Step 5** − Returns success.



Pop Operation

A Pop operation may involve the following steps −

- **Step 1** − Checks if the stack is empty.
- **Step 2** − If the stack is empty, produces an error and exit.
- **Step 3** − If the stack is not empty, accesses the data element at which **top** is pointing.
- **Step 4** − Decreases the value of top by 1.
- **Step 5** − Returns success.



Push

```
begin procedure push: stack, data
if stack is full
return null
endif top ← top + 1
stack[top] ← data
end procedure
```

Pop

```
begin procedure pop: stack
if stack is empty
 return null
endif data ← stack[top]
 top ← top – 1
 return data
end procedure
```

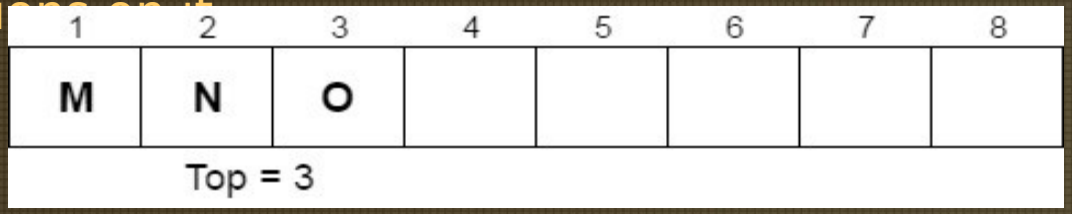
Array representation of Stack

Stack can be represented by means of a one way list or a linear array. A pointer variable top contains the locations of the top element of the stack and a variable max stk gives the maximum number of elements of the Stack that can be held by Stack.
the condition top=0 will indicate that the stack is empty.
This procedure pushes an item onto the Stack via Top.

PUSH(Stack, Top, MaxStk, Item)

1. If Top == MaxStk //check Stack already fill or not
   then print "Overflow" and return
2. Set Top = Top + 1 //increase top by 1
3. Stack[Top] = Item //insert item in new top position 4. Return

Let MaxStk=8, the array Stack contains M, N, O in it. Perform operations on it



Array representation of Stack

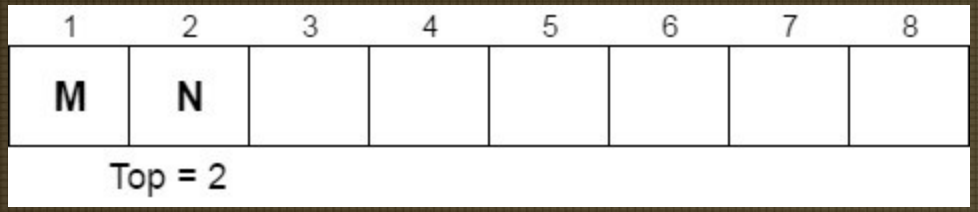
Pop operation on Stack

This procedures deletes the top element of Stack and assigns it to the variable item.

POP(Stack, top, Item)

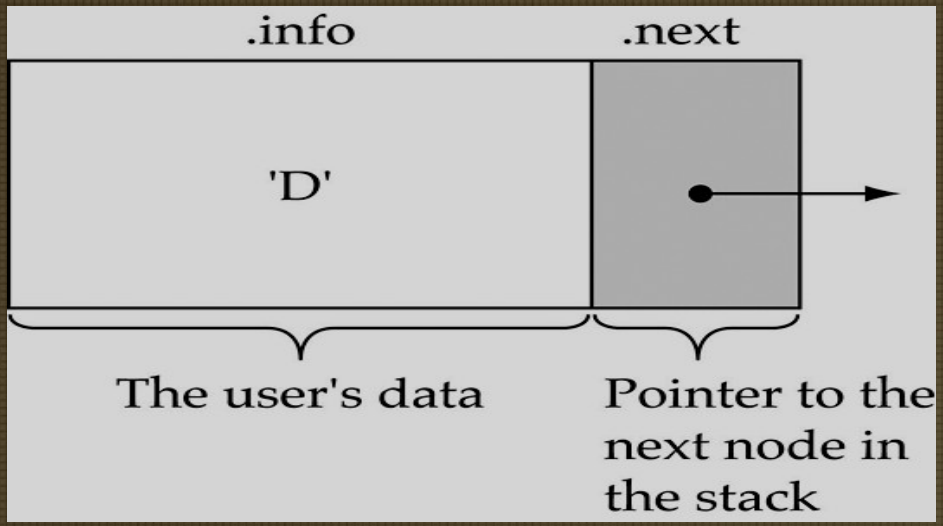
1. If Top == Null //check Stack top element to be deleted is empty
   then print "Underflow" and return
2. Item = Stack[Top] //assign top element to item
3. Set Top = Top - 1 //decrease top by 1
4. Return

pop 3 elements



Linked Representation of Stack

- Each node in the stack should contain two parts:
  - info: the user's data
  - next: the address of the next element in the stack



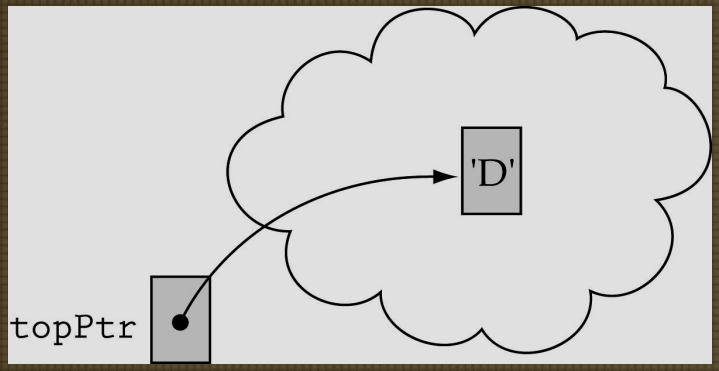
Node Type

```
template<class ItemType>
struct NodeType {
  ItemType info;
  NodeType* next;
};
```

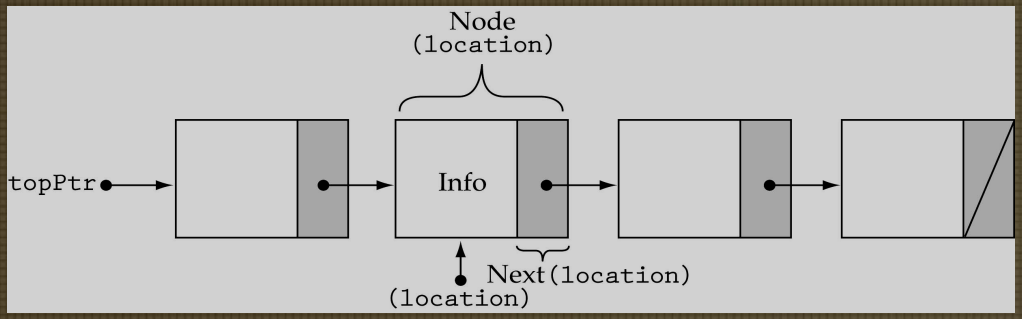
Linked Representation of Stack

First and last stack elements

We need a data member to store the pointer to the top of the stack



The *next* element of the last node should contain the value *NULL*



APPLICATIONS OF STACKS

1)Reversing Strings:

• A simple application of stack is reversing strings.To reverse a string , the characters of string are pushed onto the stack one by one as the string

is read from left to right.

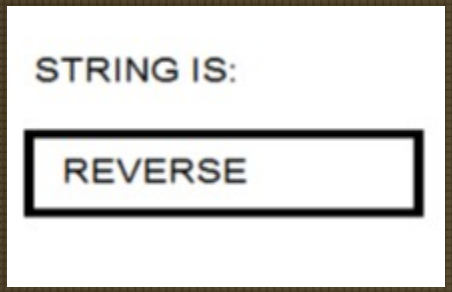
• Once all the characters of string are pushed onto stack, they are popped one by one. Since the character last pushed in comes out first, subsequent pop

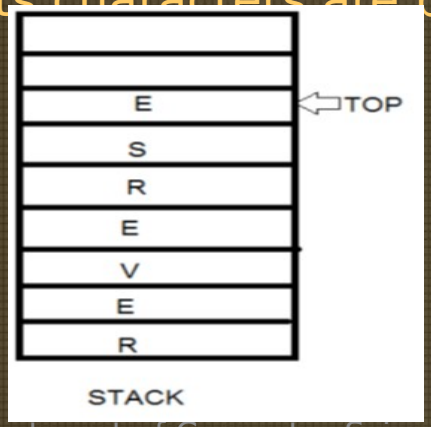
operation results in the reversal of the string.

For example:

To reverse the string ‘REVERSE’ the string is

read from left to right and its characters are pushed . LIKE:





APPLICATIONS OF STACKS

2)Evaluating arithmetic expressions:

- INFIX notation:

The general way of writing arithmetic expressions is known as infix notation.

e.g, (a+b)

- PREFIX notation:

e.g, +AB

- POSTFIX notation:

e.g: AB+

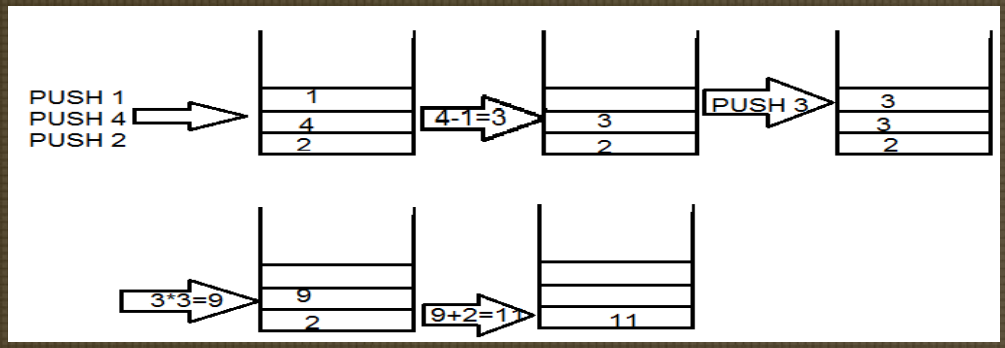
Conversion of INFIX to POSTFIX conversion:

Example: 2+(4-1)*3 step1

2+41-*3 step2

2+41-3* step3

241-3*+ step4



CONVERSION OF INFIX INTO POSTFIX
2+(4-1)*3 into 241-3*+

CURRENT SYMBOL	ACTION PERFORMED	STACK STATUS	POSTFIX EXPRESSION
(	PUSH C	C	2
2			2
+	PUSH +	(+	2
(	PUSH (	(+(	24
4			24
–	PUSH –	(+(–	241
1	POP		241–
)		(+	241–
*	PUSH *	(+*	241–
3			241–3
	POP *		241–3*
	POP +		241–3*+
)			

APPLICATIONS OF STACKS

3) CONVERSION OF INFIX INTO POSTFIX EXPRESSION

- Output operands as encountered
- **Stack** left parentheses
- When ‘)’
  - repeat
    - **pop** stack, output symbol
  - until ‘(‘
    - ‘(‘ is poped but not output
- If symbol +, *, or ‘(‘
  - **pop** stack until entry of lower priority or ‘(‘
    - ‘(‘ removed only when matching ‘)’ is processed
  - **push** symbol into stack
- At end of input, pop stack until empty

Algorithm to Evaluate a Postfix Expression

- Use a **stack,** assume binary operators +,*
- Input: postfix expression
- Scan the input
  - If operand,
    - **push** to stack
  - If operator
    - **pop** the stack twice
    - apply operator
    - **push** result back to stack

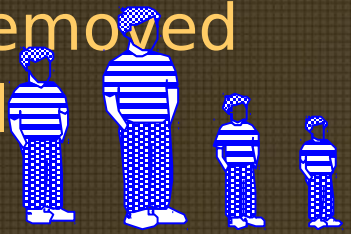
Queue

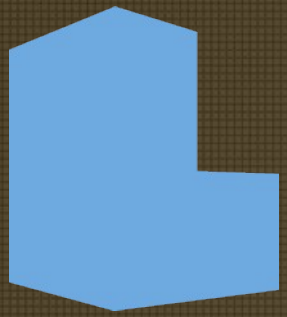
Stores a set of elements in a particular order

Queue principle: FIRST IN FIRST OUT= FIFO

It means: the first element inserted is the first one to be removed

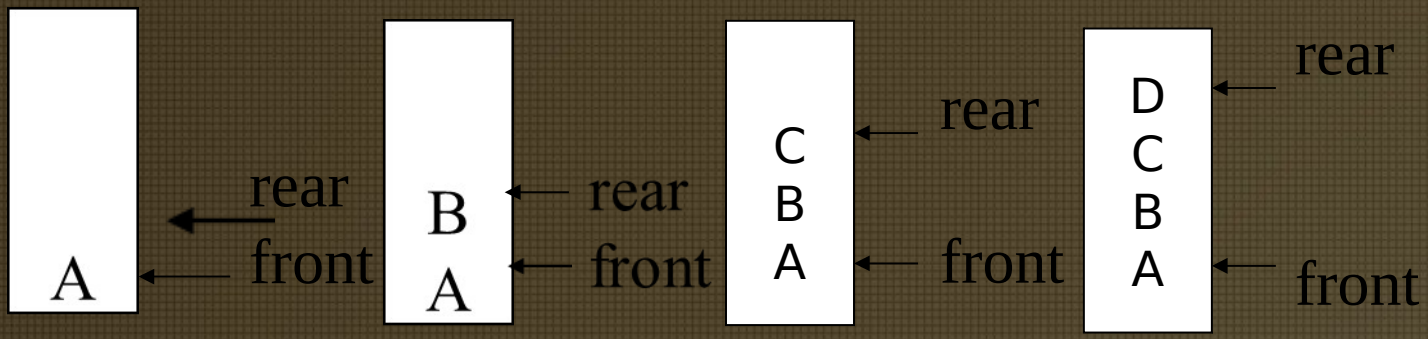
Example:





The first one in line is the first one to be served.

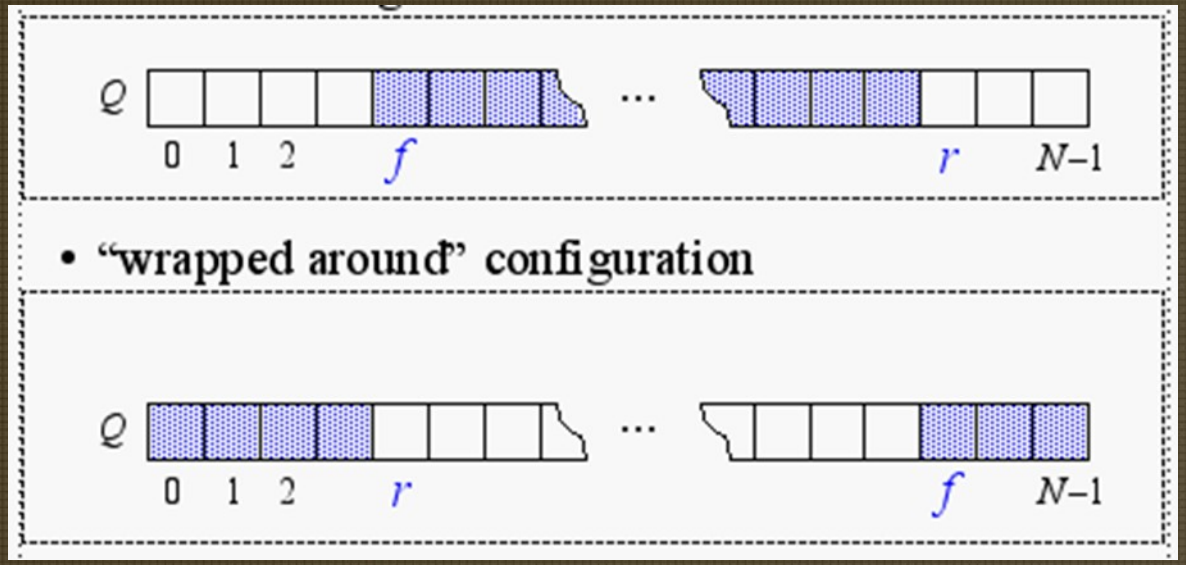
First In First Out



Array-based Queue Implementation

- As with the array-based stack implementation, the array is of fixed size
  - A queue of maximum N elements
- Slightly more complicated
  - Need to maintain track of both front and rear

Implementation 1



Implementation 2

Implementation 1: createQ, isEmptyQ, isFullQ

```
Queue createQ(max_queue_size) ::=
# define MAX_QUEUE_SIZE 100/* Maximum queue size
*/
typedef struct {
                int key;
                /* other fields */
                } element;
element queue[MAX_QUEUE_SIZE];
int rear = -1;
int front = -1;
Boolean isEmpty(queue) ::= front == rear
Boolean isFullQ(queue) ::= rear == MAX_QUEUE_SIZE-1
```

Implementation

enqueue

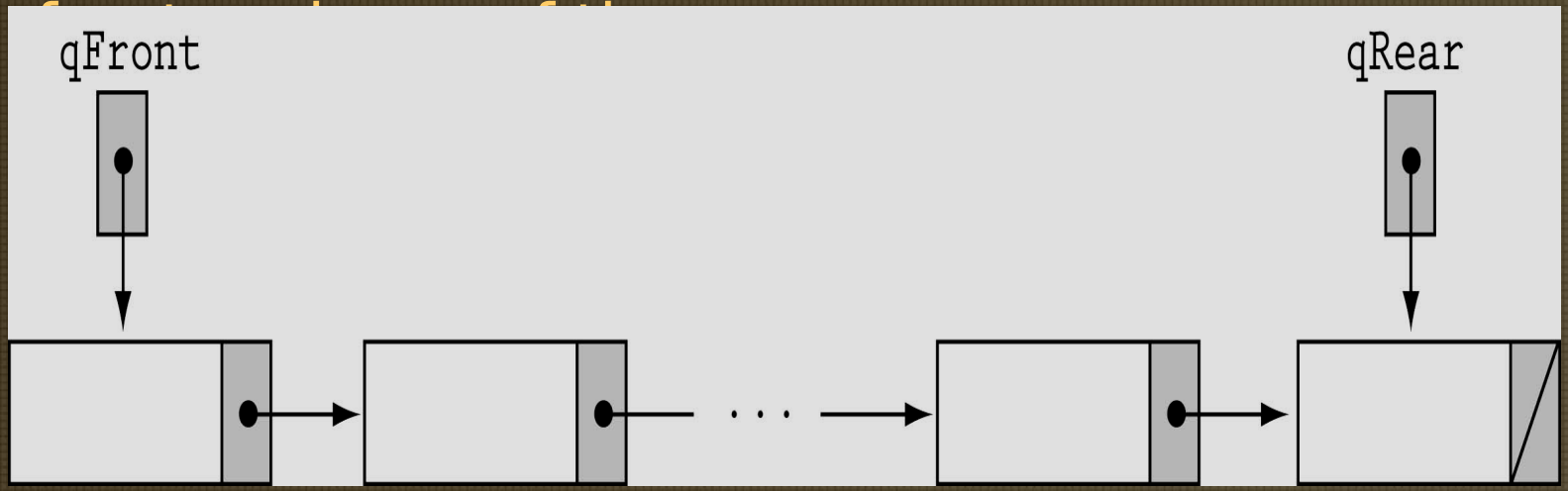
```
void enqueue(int *rear,
element item)
{
/* add an item to the queue */
    if (*rear ==
MAX_QUEUE_SIZE_1) {
       queue_full( );
       return;
    }
   queue [++*rear] = item;
}
```

Dequeue

```
element dequeue(int *front,
int rear)
{
/* remove element at the
front of the queue */
    if ( *front == rear)
         return queue_empty( );
    /* return an error key */
    return queue [++ *front];
}
```

Implementing queues using linked lists

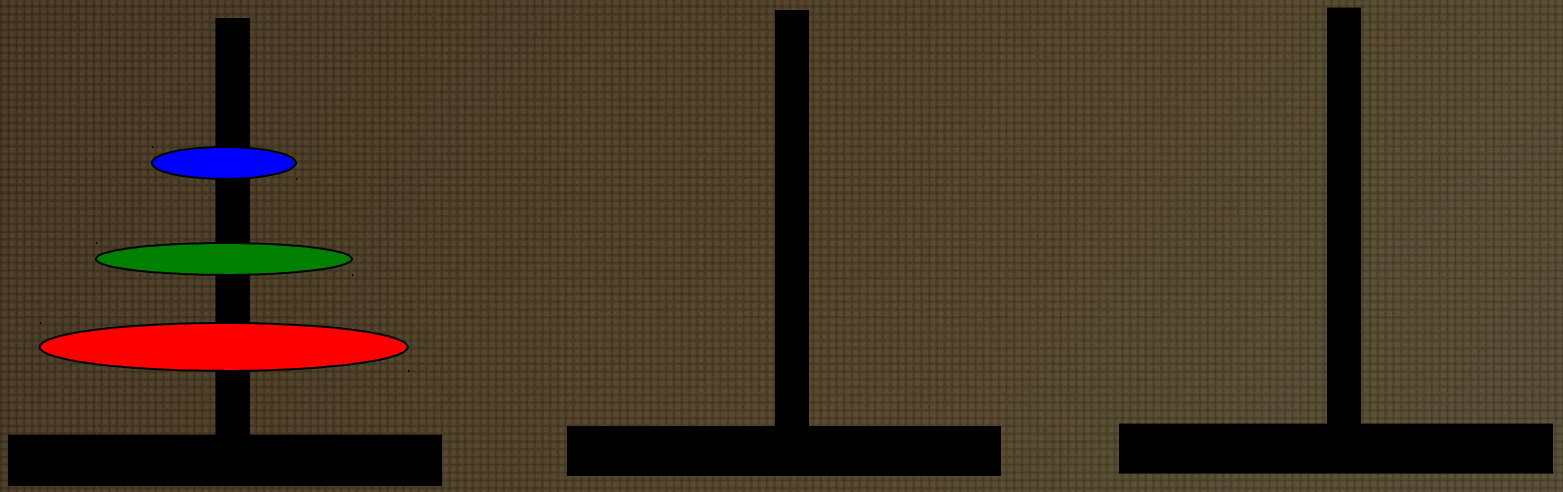
- Allocate memory for each new element dynamically
- Link the queue elements together
- Use two pointers, *qFront* and *qRear*, to mark the



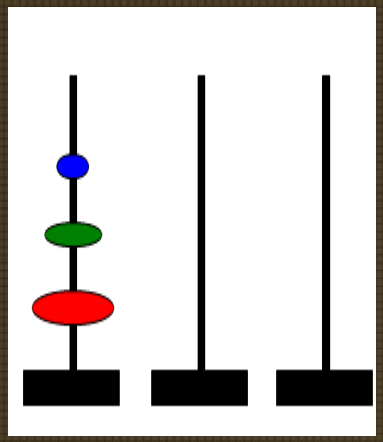
Tower of Hanoi

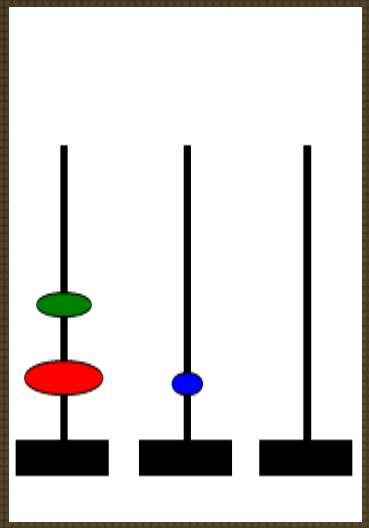
- There are three towers
- 3 gold disks, with decreasing sizes, placed on the first tower
- You need to move all of the disks from the first tower to the last tower
- Larger disks can not be placed on top of smaller disks
- The third tower can be used to temporarily hold disks
- The disks must be moved within one week. Assume one disk can be moved in 1 second. Is this possible?
- To create an algorithm to solve this problem, it is convenient to generalize the problem to the “N-disk” problem, where in our case N = 3.

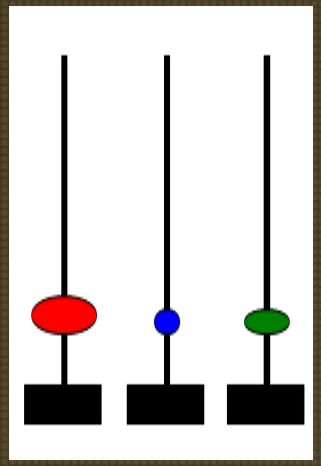
Recursive Solution

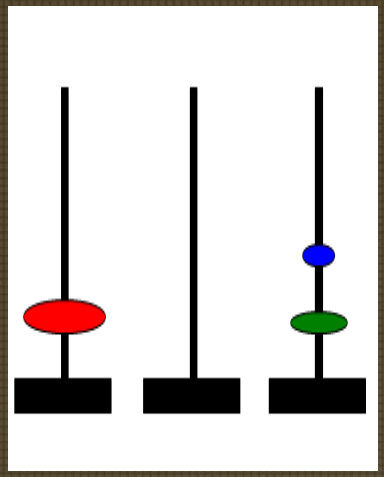


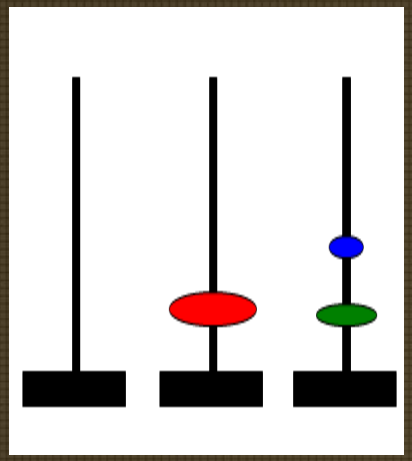
Recursive Solution

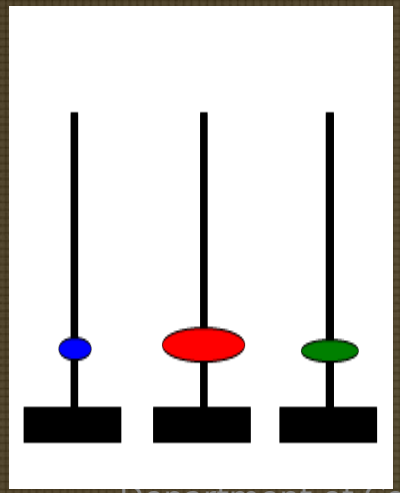


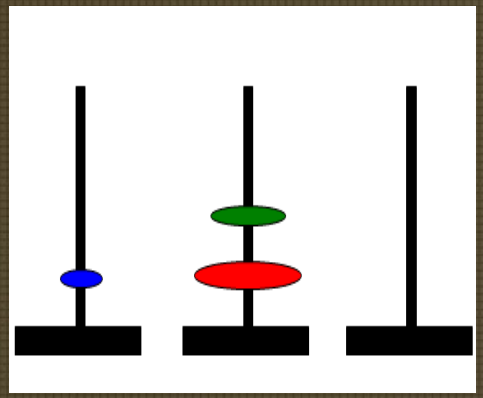


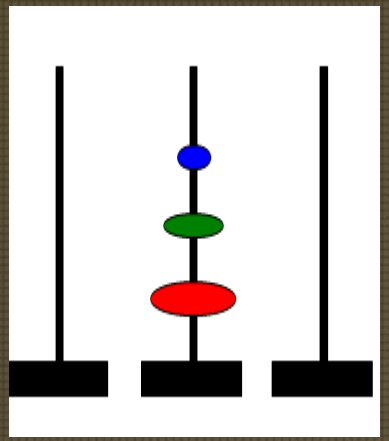






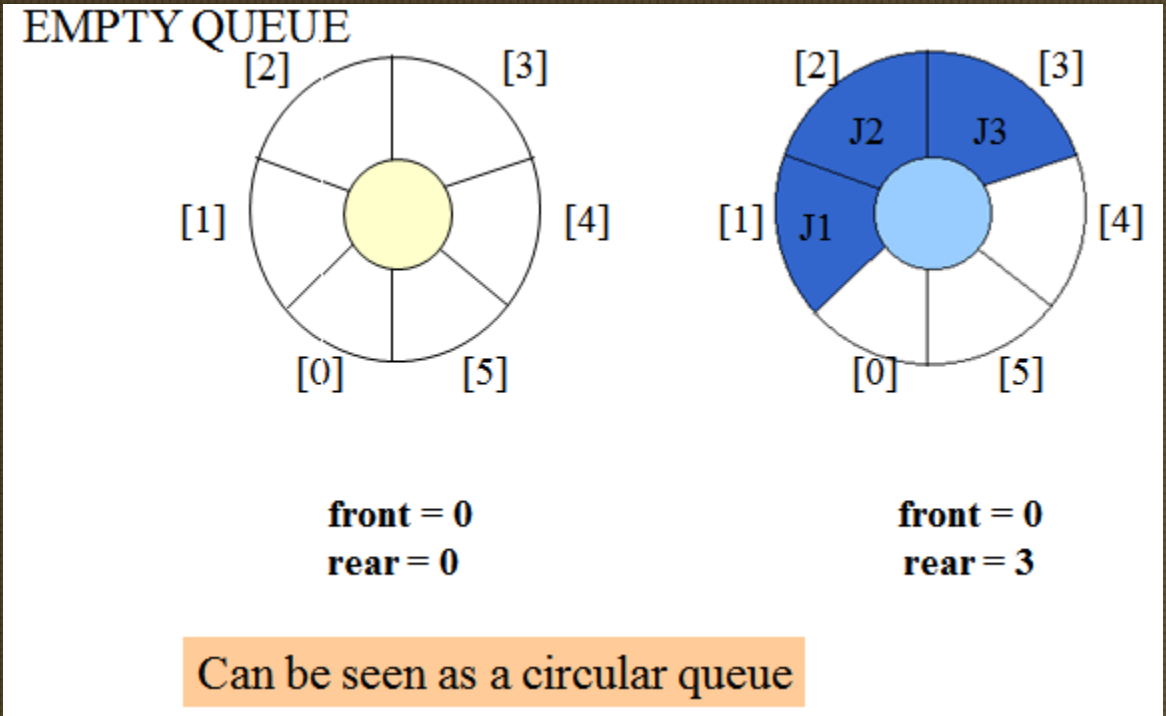






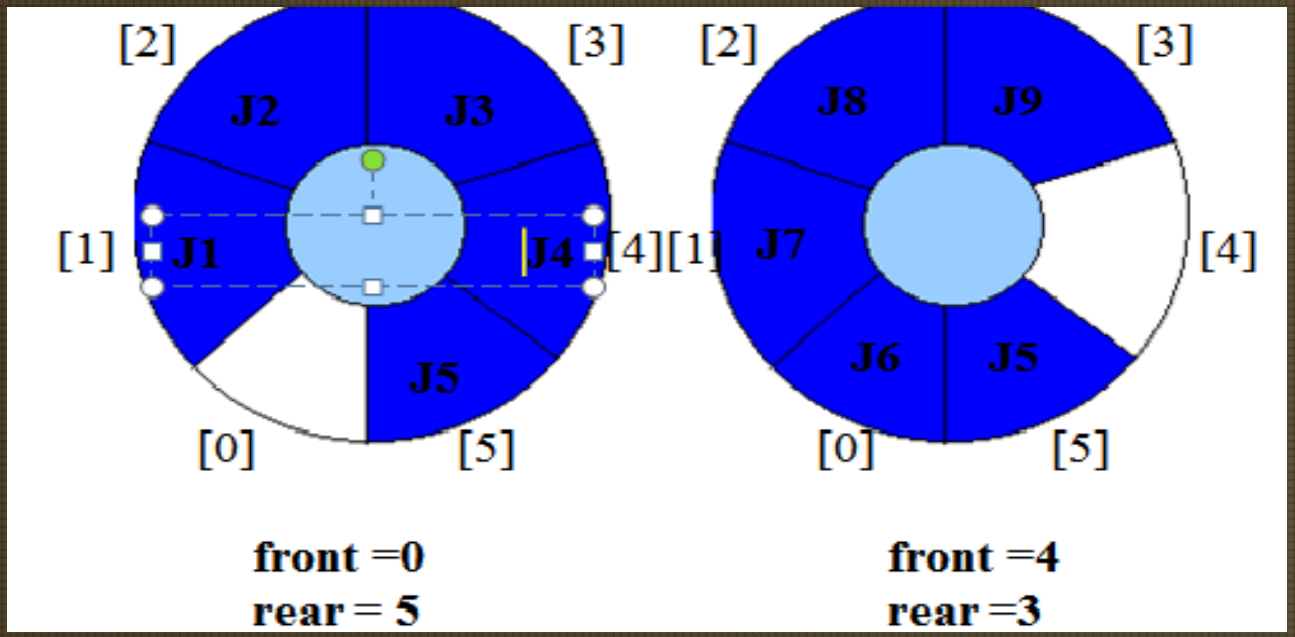
Circular Queue

EMPTY QUEUE



Leave one empty space when queue is full
Why?

FULL QUEUE FULL QUEUE



Abstract Data Type: Priority Queue

- A priority queue is a *collection* of zero or more items,
  - associated with each item is a priority

- A priority queue has at least three operations
  - *insert*(item i) (enqueue) a new item
  - *delete*() (dequeue) the member with the highest priority
  - *find*() the item with the highest priority
  - decreasePriority(item i, p) decrease the priority of ith item to p

- Note that in a priority queue "*first in first out*" does not apply in general.

Priority Queues: Assumptions

- The highest priority can be either the minimum value of all the items, or the maximum.
  - We will assume the highest priority is the minimum.
  - Call the delete operation deleteMin().
  - Call the find operation findMin().

- Assume the priority queue has n members

ASSIGNMENT QUESTIONS

1)Complete the class with all function definitions for a stack

```
class stack
{
  int data[10];
  int top;
public :
  stack(){top=-1;}
  void push();
  void pop();
}
```

2)Change the following infix expression postfix expression.

(A + B)*C+D/E-F

3)Convert the expression (True && False) || !(False || True) to postfix expression. Show the contents of the stack at every step.

ASSIGNMENT QUESTIONS

4)Use a stack to evaluate the following postfix expression and show the content of the stack after execution of each operation. Don't write any code. Assume as if you are using push and pop member functions of the stack.

AB-CD+E*+ (where A=5, B=3, C=5, D =4, and E=2)

5)Evaluate the following postfix expression using a stack and show the contents of stack after execution of each operation :

50,40,+,18, 14,-, *,+

6)Evaluate the following postfix expression using a stack and show the contents of stack after execution of each operation :

TRUE, FALSE, TRUE, FALSE, NOT, OR, TRUE, OR, OR, AND

ASSIGNMENT QUESTIONS

7)Complete the class with all function definitions for a circular queue

```
class queue
{
 int data[10];
 int front, rear;
public :
 queue(){front=-1;rear=-1}
 void add();
 void remove();
}
```

8)Each node of a STACK contains the following information, in addition to required pointer field :

i) Roll number of the student

ii) Age of the student

Give the structure of node for the linked stack in question TOP is a pointer which points to the topmost node of the STACK. Write the following functions.

i) PUSH() - To push a node to the stack which is allocated dynamically

ii) POP() - To remove a node from the stack and release the memory.

9)Write a function in C++ to perform a DELETE operation in a dynamically allocated queue considering the following description :

ASSIGNMENT QUESTIONS

```
struct Node
{
      float U,V;
      Node *Link;
};
class QUEUE
{
      Node *Rear,*Front;
public:
      QUEUE(){Rear=NULL; Front=NULL;}
      void INSERT();
      void DELETE();
      ~QUEUE();
};
```

10) Give the necessary declaration of a linked list implemented queue containing float type values. Also write a user-defined function in C++ to delete a float type number from the queue.

TUITORIAL QUESTIONS

1)Discuss how to implement queue using stack.

2) Discuss how to implement stack using queue.

3) What are priority queues?

4)Implement Tower of Hanoi problem

5)WAP to convert infix expression into postfix expression.

Outcomes

After reading above topics students will be able to:

- Understand Linear Data Structure Stack and Queue
- Know applications of stacks and queues.

References

1)Data Structures and Algorithms Made Easy by *Narasimha Karumanchi*

2)Lipschutz, “Data Structures” Schaum’s Outline Series, Tata McGraw-hill Education (India) Pvt. Ltd.

3) Thareja, “Data Structure Using C” Oxford Higher Education.

4) AK Sharma, “Data Structure Using C”, Pearson Education India.

5)Rajesh K. Shukla, “Data Structure Using C and C++” Wiley Dreamtech Publication.