What is stack?

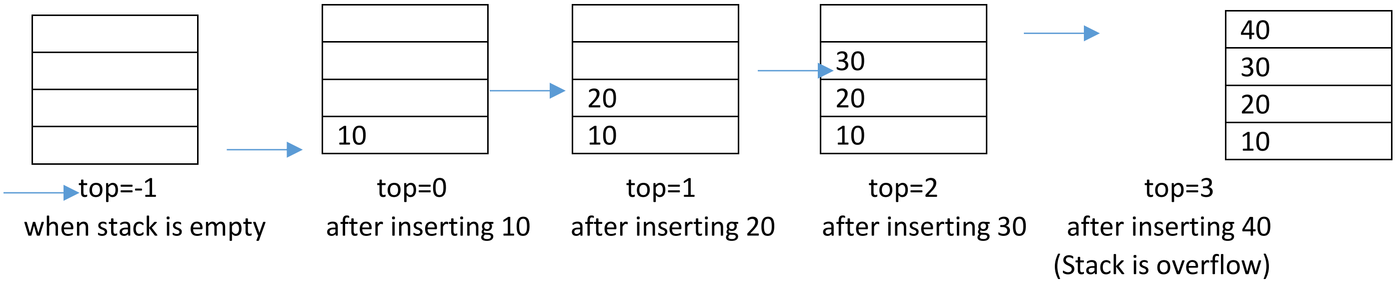
- It is a linear data structure or ordered collection of elements where the elements are processed in last in first out manner(LIFO).
- In stack insertion and deletions are made at one end i.e top
- Stack can be implemented by using array or linked list

Operations on stacks.

Special terminology is used for two basic operations associated with stacks are,

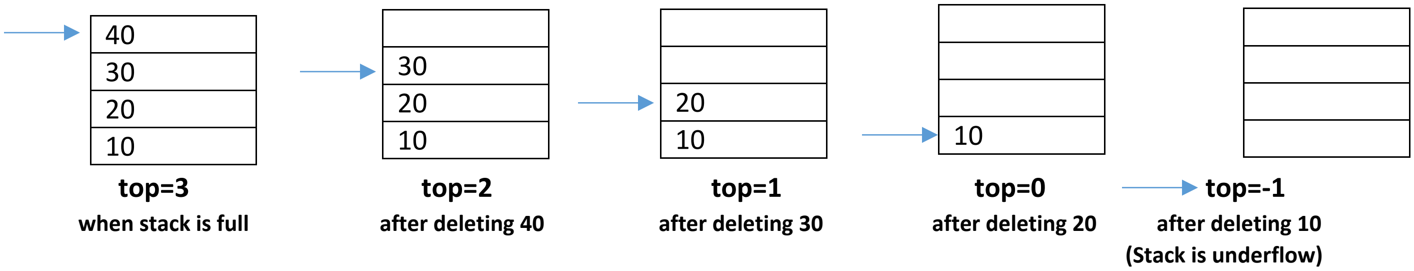
a. ”push” is the term used to insert an element into stack.
b. ”pop” is the term used to delete an element from stack.

a.PUSH operation.



- When top=-1 stack is empty i.e there are no elements in the stack
- After inserting first element top is incremented to 1st position(0th index)of stack
- Similarly top will get increment BY ONE after inserting every element in to stack
- When top value become MAXSTK-1(where MAXSTK is maximum number of elements that stack can have) stack is full that condition is called as ***stack overflow.***

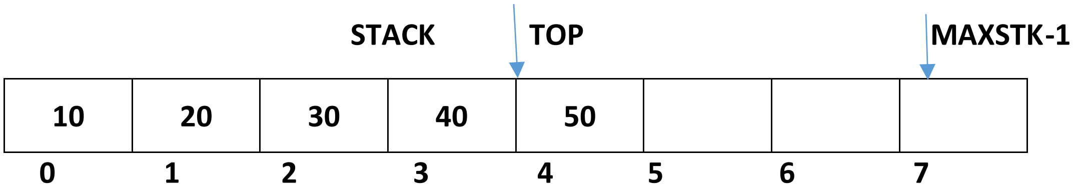
b.POP operation.



- If top ≠ -1 the top element of stack is deleted, otherwise stack is empty
- When an element of the stack is deleted top get decremented by one
- When top=-1 stack will become empty and that condtion is called ***stack underflow***

Array Representaion of stacks:

- Stacks may be represented in the computer in various ways,normally by using linear array or Linked list.
- Unless we specified or stated ,each of our satcks will be maintained by a linear array STACK.
- A pointer variable TOP ,which contains the location of the top element in the stack
  And a variable MAXSTK which gives the maximum number of elements that can STACK have.
- The condition TOP=-1 or TOP=NULL will indicate that the stack is empty



- The above array can be used to perform stack operations like push and pop

Note: **C program to perform stack operations.**
Refer lab program-3

ADT stack is(Abstract Data type)

Objects: a finite ordered list with zero or more elements
Functions:
Stack create(max)::= Create an empty satck,whose maximum size=max
Boolean ISFull(stack)::= return true if top=max-1 else return false.
Stack push(item)::=push element ,item to stack
Element pop()::= if(top=-1) stack is empty,else return topmost element

Stack using Dynamic Arrays

- When a stack is implemented by using static arrays the size of the stack is bound to MAXSTK(i.e maximum elements a stack can have)but some time stack need be dynamic.
- This can be achieved by using dynamic arrays
- The memory for the stack is dynamically allocated by using memory allocation functions such as malloc or calloc
- When the stack is full , memory of the stack is doubled by using realloc() function and the capacity of the stack is doubled(MAXSTK is doubled)

The implementation of Stack by using dynamic arrays is as follows

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int *stack;
int capacity=1;
int top=-1,item;
void push()
{
      if(top==capacity-1)
      {
         stackfull();//which double the memory when stack is full
      }
      printf("enter an item to insert \n");
      scanf("%d",&item);
      stack[++top]=item ;
}
void pop()
{
      if(top==-1)
      {
           printf("underflow\n");
           return;
      }
      item=stack[top--];
      printf("item deleted is %d \n",item);
}
void stackfull()
 {

     stack=realloc(stack,capacity*2*sizeof(int));// doubling the memory
         if(stack==NULL)// IF MEMORY IS IN SUFFICIENT
          {
           printf(“memory is insuffient\n”);
           exit(0);
          }
      capacity=capacity*2;   // doubling the stack size

   }


void display()
{
      int i;
       if(top==-1)
        {
        printf(“stack is empty \n”);
        Return;
        }
      for(i=top;i>=0;i--)
      printf("%d",*(stack+i));
 }

 void main()
 {
    int choice=1;
   stack=malloc(capacity*sizeof(int));//allocating memory dynamically
    while(choice)
    {
    printf("enter your choice\n 1.push\n 2.pop\n 3.display \n 4 exit\n");
    scanf("%d",&choice);
    switch(choice)
    {
          case 1:push();
                 break;
          case 2:pop();
                 break;
          case 3:display();
                 break;
          case 4:printf("invalid operation\n");
                 exit(0);
    }
  }
  free(stack);// deallocating memory
  stack=NULL; //avoiding dangling pointer
}
```

In the above implementation the function ***stackfull()*** doubling the size and memory of the stack.

<u>Applications of stacks:</u>

- Stack is used to convert expressions .ex: infix to postfix.
- Stack can be used to evaluate the expressions
- During function call stack is used

Arithmatic Expressions:

An expression is a one which is a combination of meaning full Arithmatic operators and operands.

The prcedence levels of the Arithmatic operators as follows.

Highest: Exponentiation(↑)or(^)or($)

Next Highest: Multiplication(*) , division(/) and modulus(%).

Lowest: Addition(+) and substraction(-)

There are different notations are used to represent Arithmetic expressions.

a.infix notation

b.polish/prefix notation

c.postfix/suffix/reverse polish notation.

Infix notation:

In most common arithmetic opeartions,the opeartor is placed b/w the two operands.

1) A+B 2)C-D 3) E*F 4) G/H 5)(a+b)*c

It is called infix notation,it can be parantheised or parantheses free expressions.

Polish /Prefix notation:

It is named after the polish mathematician Jan Lukasiewicz,refers to the notation in which the operator symbol is placed before its two operations.

Example: +AB, -CD, *+ABC

Convert infix in to Polish(prefix):

1. (a+b)*c= [+ab]*c= *+abc . where [ ] indicates partial translation.
2. A+(B*C)=A+[*BC]=+A*BC
3. (A+B)/(C-D)=?

Postfix/Suffix/Reverse polish notation:

In which operator is placed after the operands.

AB+ CD* AB+C* ABC*+

- Computer usually evaluates an arithmetic expression written in infix notation in two steps
  i)it converts the expression into potfix notation
  ii)then it evaluate that expression.
- stack is the main tool used to acomplish this task.

Converting infix expression into postfix/suffix/reverse polish expression

Procedure to convert by using satck

- read one input symbol at a time from the array of characters(infix expression)
- if a input symbol is an operand .write it into output(postfix)
- if a input synmbol is an operator,push it in to stack,if the stack is empty.if stack is not empty compare the precedance of stack symbol and input symbol. pop all the stack symbol having higher or equal prority(write poped entries into stack) and then only push operator in to stack.
- If the input symbol is left parentheses '(' push it into satck.
- If input symbol is right parentheses ')",pop entries of satck and write those entries into output till you find "(' in satck. (don't write '(' into output).
- When you finish reading the string,pop all the left out synbols from the stack and store in output.

Ex: convert (A*B)+C into postfix using stack.

Input symbol	Stack				Top	Output[postfix]
	0	1	2	3		
(	(				0	
A	(				0	A
*	(	*			1	A
B	(	*			1	AB
)					-1	AB*
+	+				0	AB*
C	+				0	AB*C

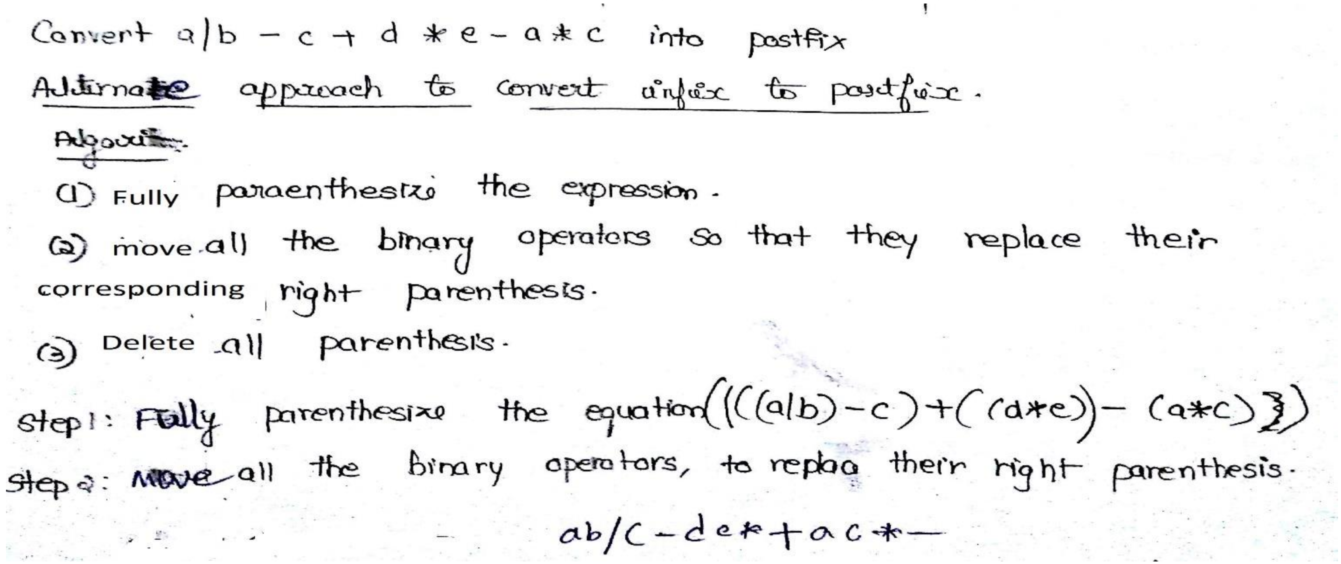
Now the infix expression is ended: check out the status of stack if it is not empty pop the entries of satck till it become empty and write all the poped entries into output. After making stack empty,

The postfix expression of (A+B)*C= AB*C+

NOTE: for more problems refer class notes or lab observation book.

Convert the following infix expression to postfix expression

1. a*(b+c)*d 2.(a+b)*d+e/(f+a*d)+c 3.(a*b)+c/d 4.(((a/b-c)+(d*e))-(a*c))



Convert a/b - c + d * e - a * c into postfix

Alternate approach to convert infix to postfix.

Algorithm

(1) Fully paraenthesize the expression.

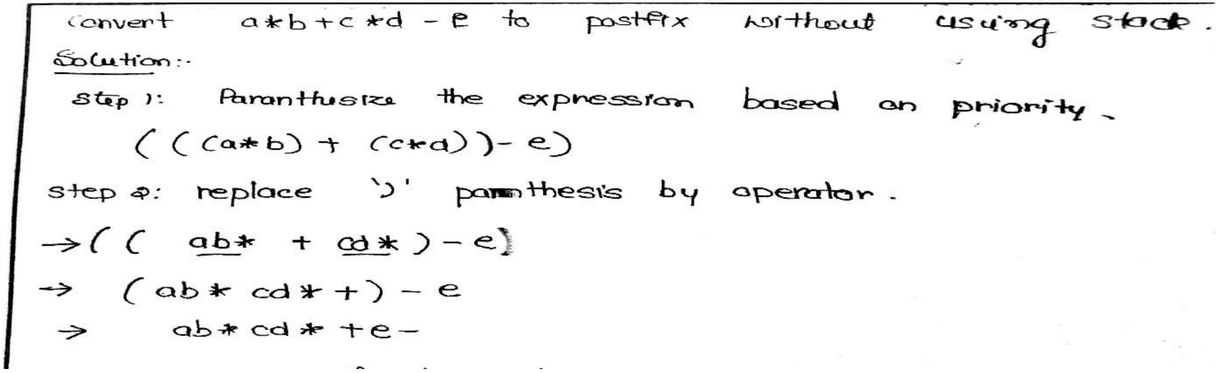
(2) move all the binary operators so that they replace their corresponding right parenthesis.

(3) Delete all parenthesis.

Step1: Fully parenthesize the equation(((a/b)-c)+((d*e))-(a*c)})

Step 2: Move all the binary operators, to replace their right parenthesis.

ab/c-de*+ac*-



Convert a*b+c*d-e to postfix without using stack.

Solution:-

Step 1: Paranthesize the expression based on priority.

(((a*b) + (c*d))-e)

Step 2: replace ')' parenthesis by operator.

→((ab* + cd*)-e)

→ (ab* cd* +) - e

→ ab* cd* +e-

NOTE: C program to convert infix to postfix:(refer lab program-4)

EVALUATION OF POSTFIX EXPRESSION:

<u>Procedure to evaluate :-</u>

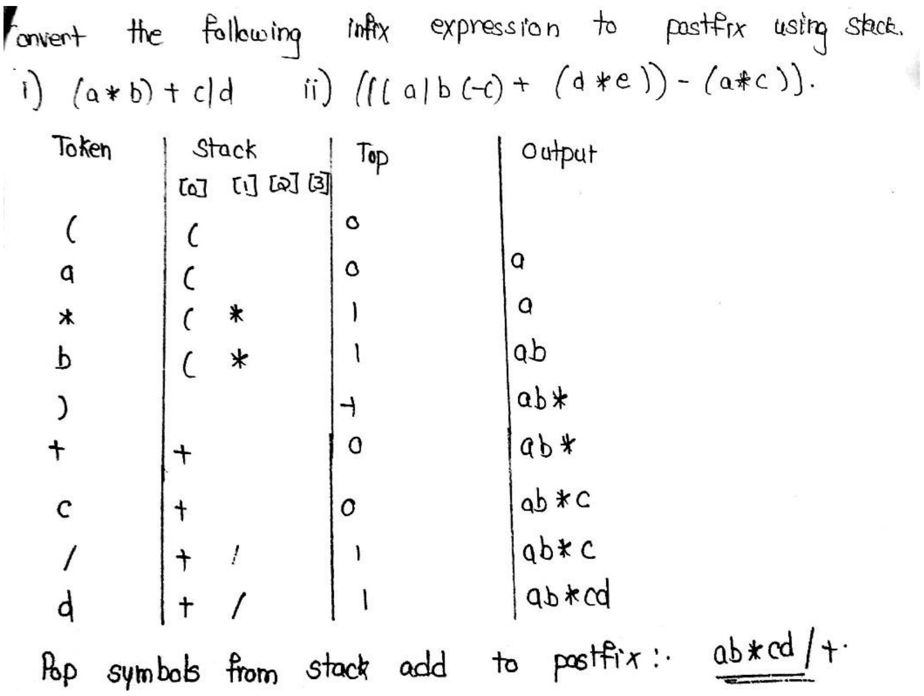
1.read only one input symbol at a time from postfix expression.
2.if input symbol is operand ,push operand in to stack(before pushing convert that into integer).
3.if input symbol is opeartor ,pop two operands from stack perform operation and push result into satck.
4.repeat the process till the expression become empty.
5.finally result will be stored in the top of the stack(top==0).

Evaluate the postfix expression :- 62/3-42*+

Input	Stack						TOP=-1 (intially)	Process
	0	1	2	3	4	5		
6	6						0	push 6 into stack
2	6	2					1	Push 2 into stack
/	3						0	Pop 6 ,2 and divide ,push result into stack
3	3	3					1	push 3 into stack
-	0						0	Pop 3 ,3 and substract,push result into stack
4	0	4					1	Push 4 into stack
2	0	4	2				2	Push 2 into satck
*	0	8					1	Pop 2 and 4 apply opeartor and push result int stack
+	8						0	Pop 8 and 0 from the stack apply opeartor and push result into stack.

Result=8

Examples:

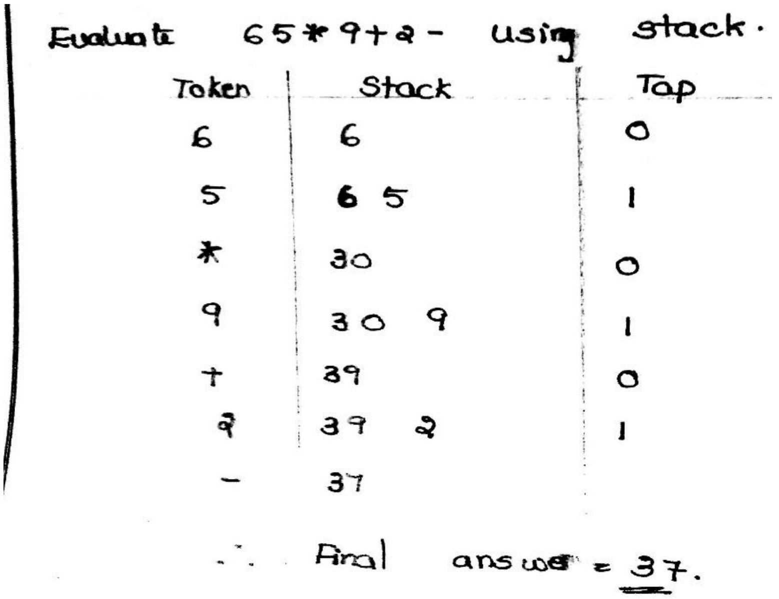


Convert the following infix expression to postfix using stack.

i) (a*b)+c/d ii) (((a/b(-c)+ (d*e)) - (a*c)).

Token	Stack [0]	[1]	[2]	[3]	Top	Output
(	(				0	
a	(				0	a
*	(	*			1	a
b	(	*			1	ab
)					-1	ab*
+	+				0	ab*
c	+				0	ab*c
/	+	/			1	ab*c
d	+	/			1	ab*cd

Pop symbols from stack add to postfix :· <u>ab*cd</u>/+·



Evaluate 65*9+2- using stack.

Token	Stack	Tap
6	6	0
5	6 5	1
*	30	0
9	30 9	1
+	39	0
2	39 2	1
-	37	

∴ Final answer = 37.

NOTE: **C program to evaluate suffix/postfix expression:(refer lab-program-5a)**

Recursion:

Recursion is the name given for expressing anything in terms of itself.

A function which contains a call to itself or call to another function ,which eventually causes the first function to be called,is known as a ***recursive function***.

- Recursive procedures generally solve a given problem by reducing the problem to an instance of the same problem with smaller input.
- Once the function is called an activation record is created on the stack
- Call to it self is repeated till a base condition is reached.
- Once a base condition or terminal condition is reached,the function returns the result to previous copy of the function.
- A sequence of returns ensures that the solution to the original problem obtained.

The recursive procedure(function) must have the following properties.

- There must be certain condition ,called base condition,for which the procedure(function) does not call itself.
- Each time the procedure(function) does call itself (directly or indirectly),it must be closer to the base condition.

Examples for recursive function.

1. **Factorial function**

   The product of the positive integers from 1 to n,inclusive,is called “n factorial” and is usually denoted by n!.

n!=1.2.3…..(n-2)(n-1)n.

Recursive procedure to find the factorial of N.(algorithm)

```
FACTORIAL(N,FACT)
1.IF N=0,then:Set Fact:=1 and return
2 call FACTORIAL(N-1,FACT);
3.Set FACT:= N*FACT;
 4.return.
```

Recursive function in C

```
 int fact(int n)
{
   If(n==0)
        return 1;
     return (n*fact(n-1));
}
```

Note: Write a iterative function(by using loop) to find factorial of n.

2.Fibonacci Sequence:

The fibonacci sequence is series of terms where each suceeding term is a sum of two preceding terms.

0,1,1,2,3,5,8……

Here,F_0=0, F_1=1.

$F_3=F_0+F_1=0+1=2$

Similarly, $F_n=F_{n-2}+F_{n-1}$

Recursive Procedure to find fibanocci sequence.

```
FIBONACCI(FIB,N)
 1)If N=0 or N=1,then :Set FIB:=N, and Return
 2)Call FIBONACCI(FIBA,N-2).
 3)Call FIBONACCI(FIBB,N-1).
 4)Set FIB:=FIBA+FIBB.
 5)Return.
```

Recursive function in C to find nth fibonacci number

```
  int fibonacci(int n)
  {
     if(n==0)
            return 0;
   if(n==1)
             return 1;
       return (fibonacci(n-2)+fibonacci(n-1));
}
```

Note: Write a iterative function(by using loop) to generate n fibanocci sequence.

3.Ackermann function.

- It is a non primitive or nested recursive function.(a primitive recursive is a one which can be implemented by using loops ex: factorial,GCD etc.)
- In computability theory, the Ackermann function, named after ***Wilhelm Ackermann***
- ***Main use of Ackermann function is in mathematical logic***
- ***It is one of the classical example for recursion.***
- After Ackermann's publication of his function (which had three nonnegative integer arguments), many authors modified it to suit various purposes, so that today "the Ackermann function" may refer to any of numerous variants of the original function. One common version, the **two-argumen**t is defined as follows for nonnegative integers m and n:

Ackermann function. A(m,n)

1)A(m,n)=n+1,   when m=0

2)A(m,n)=A(m-1,1), when m>0, n=0

3) A(m,n)=A(m-1, A(m,n-1))   when m>0 and n>0

Ackermann function in C

```
int acker(int m,int n)
{
  if(m==0)
          return n+1;
  else if(m>0&&n==0)
          return acker(m-1,1);
   else return acker(m-1,acker(m,n-1));
}
```

What is the value of A(1,3)?

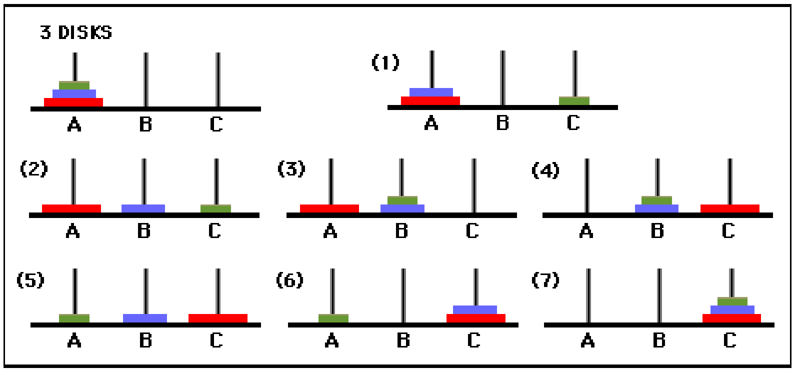
4.Tower of Hanoi:-

- It is a popular game
- It is one of the best example how recursion used as tool in developing an algorithm to solve a particular problem.
- Consider 3 pegs A , B & C & suppose on peg A there are placed a finite number n of disks with decreasing order

  Rules

  1)Only one disk may be moved at a time

  2)At no time can a larger disk be placed on a smaller disk



- For n=3 : A→C ,A→B ,C→B , A→C, B→A, B→C, A→C
- For completeness ,we also give the solution to the Towers of Hanoi problem for n=1 & n=2

  n=1 : A→C (one move)

  n=2 : A→B , A→C , B→C (3 moves)

Technique of recursion to develop a general solution

1. Move the top n-1 disks from peg A to peg B
2. Move the top disk from peg A to peg C
3. Move the top n-1 disks from peg B to peg C

Recursive procedure to Tower of Hanoi:- TOWER(N,BEG,AUX,END)

```
1. When n=1,then
    a.  TOWER(1, BEG,AUX,END)     or Write:=    BEG→END
    b.  return
2. When n>1
    a. TOWER(N-1,BEG,END,AUX)     [ Move the top n-1 disks from peg A to peg B]
    b. TOWER(1, BEG,AUX,END) or Write:- BEG→END
    c. TOWER(N-1,AUX,BEG,END)     [Move the top n-1 disks from peg B to peg C ]
```

C program to Tower of Hanoi

```
 #include<stdio.h>
#include<conio.h>
int tower(int n,char beg,char aux,char end)
  {
               if(n==1)
               {
                          printf("thed disk 1 is move from %c to %c\n",beg,end);
                          return;
               }
      tower(n-1,beg,end,aux);
      printf("the disk  %d is moved from %c to %c\n",n,beg,end);
      tower(n-1,aux,beg,end);
 }
 void main()
   {
             int num;
              printf("enter the number of disk \n");
              scanf("%d",&num);
             tower(num,'A','B','C');
             getch();
   }
```

GCD recursive function:[

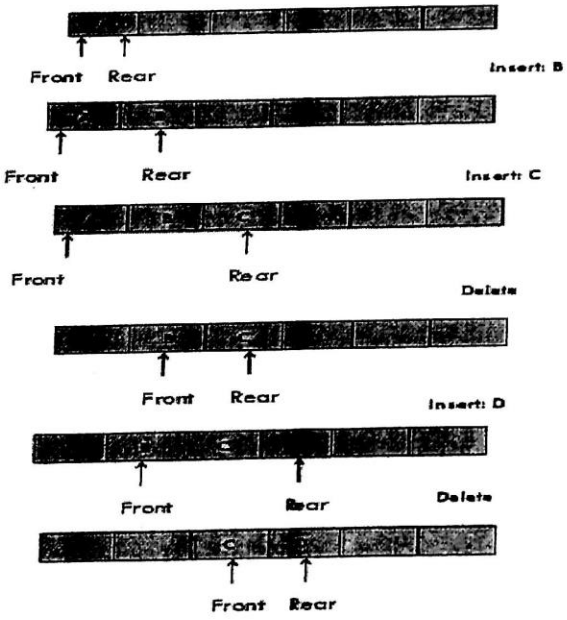
```
int gcd(int n1, int n2)

  {

     if (n2!=0)

     return gcd(n2, n1%n2);

     else

     return n1;

  }
```

Queue

- Queue is also an ordered list data structure, in which the element is inserted from one end and deleted from the other end
- The end from which insertions are done is called **REAR** and the deletion of existing element takes place from end called as **FRONT**.
- This makes queue as **FIFO** (First in first out) data structure, which means that element inserted first will also be removed first.

Implementation of Queue

- Queue can be implemented using an Array or Linked List. The easiest way of implementing a queue is by using an Array. Initially the FRONT=0 and the REAR= -1. As we add elements to the queue, the **rear** keeps on moving ahead, always pointing to the position where the last element was inserted, while the **front** remains at the first index.
- Inserting and deleting elements in a queue.



Applications of Queue

Queue, as the name suggests is used whenever we need to have any group of objects in an order in which the first one coming in also gets out first while the others wait for their turn, like in the following scenarios:

1. Serving requests on a single shared resource, like a printer, CPU task scheduling etc.

2. In real life, Call Centerphone systems will use Queues, to hold people calling them in an order, until a service representative is free.
3. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive, First come first served.

Example :-

Job Scheduling in computer programming , makes use of Queue. Show the Queue (Sequential Queue) for scheduling of the jobs as given below.

J1, J2, J3 arrive in order Job J4 arrives after J1 is completed.

Front	Rear	Q[0]	Q[1]	Q[2]	Q[3]	Comments
0	-1					Queue is empty
0	0	J1				Job J1 arrives
0	1	J1	J2			Job J2 arrives
0	2	J1	J2	J3		Job J3 arrives
1	2		J2	J3		Job J1 completed
1	3		J2	J3	J4	Job J4 inserted.

Observations:-

* If a queue (sequential) is implemented using arrays, when rear points to max-1 queue is full.

* If front > rear queue is empty.

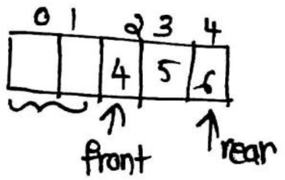
Important note:- → If rear= max-1 is Queue full? not always, see the above example size of array is 4, rear=3 (max-1) is it full.

The sequential queue is actually full, if front =0, when rear =max-1.

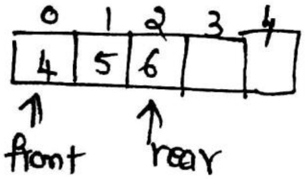
* If front > 0, when rear = max-1, then there are empty locations in the queues, as in the previous case.

• In such cases, copy the elements from front, to rear, to the begining of the ~~front~~ queue at starting position (zero) because array index begins at 0.

Ex :



rear= max-1 = 5-1,



after shifting elements.

* Shifting the elements is time consuming (draw back of linear or sequential queue.) Solution is use Circular Queue.

Circular Queue:

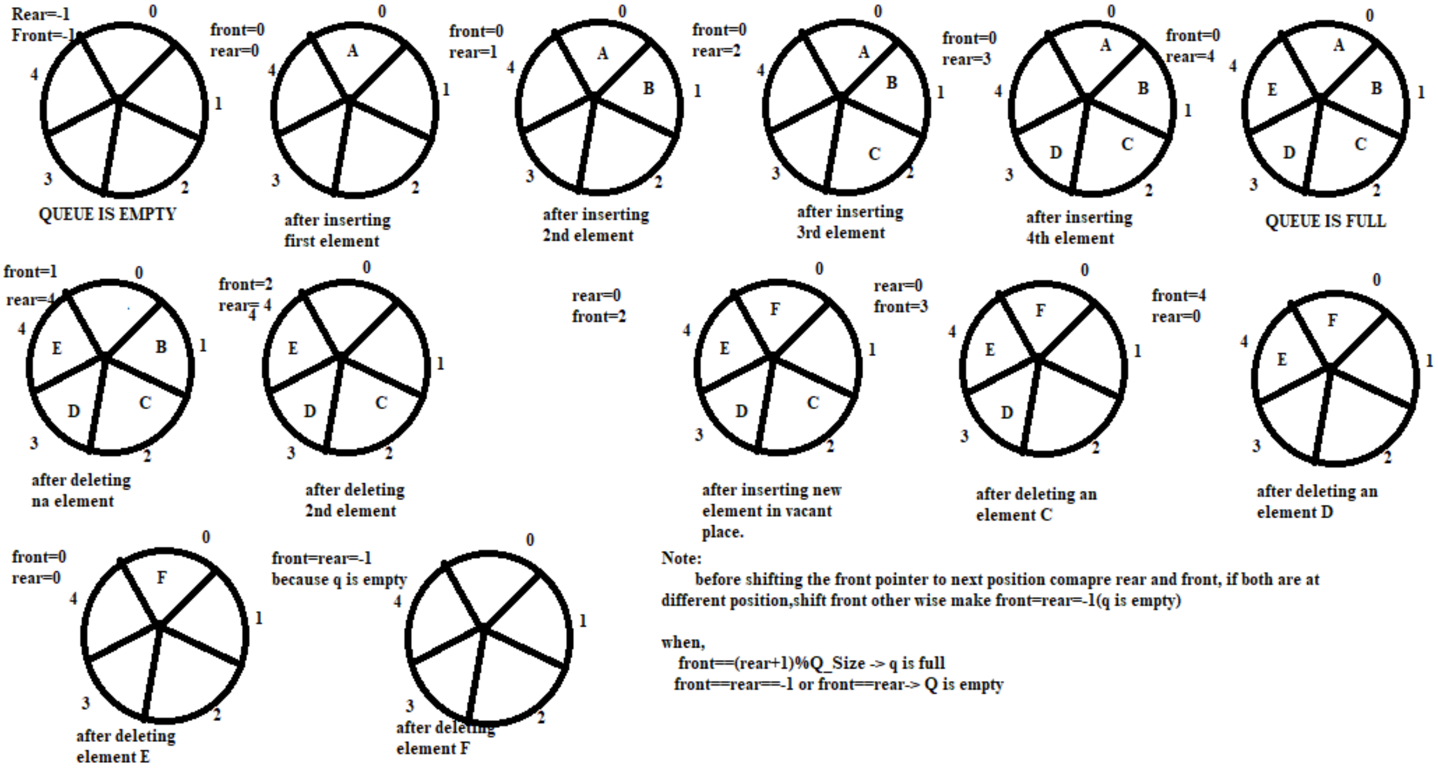
1.elements are represented in circular manner.

2.implemented by using arrays

3.two ends front and rear are moving in circular fashion

Implemeting circular queue by using arrays.

Implementation of circular queue by using arrays diagramtically represented as follows.



Note:

before shifting the front pointer to next position comapre rear and front, if both are at different position,shift front other wise make front=rear=-1(q is empty)

when,

front==(rear+1)%Q_Size -> q is full
front==rear==-1 or front==rear-> Q is empty

C program to implement circular queue (refer Lab program6)

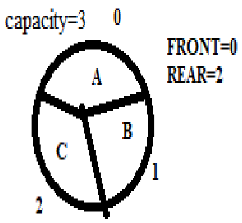
Dynamically allocated circular queue:

1.when queue is full(or) it is about full,the size of the queue is doubled.

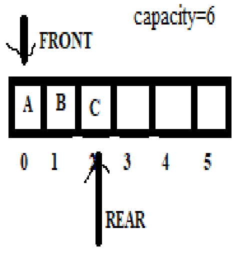
2.this can be done by creating a newarray dynamically whose memory will be twice than the current array.

3.after creating newarray the elements of current array will be copied in to new arra

Implementation of dynamically allocated circular queue is diagrammatically represented as follows.



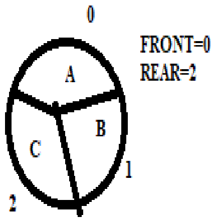
QUEUE IS FULL



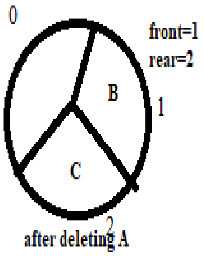
NEW ARRAY AFTER DOUBLING MENORY (flattened view)

ELEMENTS FROM CURRENT ARRAY ARE COPIED INTO NEW ARRAY i.e from queue[0] to rear.
Note: here it is only one segment
there will be no change in the values of rear and front in new array

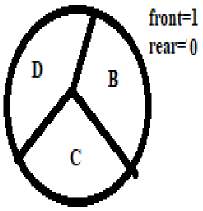
when queue has two segments.



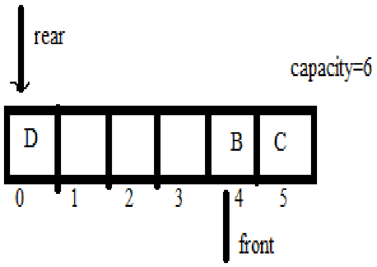
QUEUE IS FULL



after deleting A from queue



after inserting element D Q is full



elements from currents array are copied in to new array segment by segment.
Segment1(elements inserted earlier or first cycle elements) i.e elements from queue[front] to queue[capcity-1].
segment2(recently inserted elements or current cycle)i.e from queu[0] to queue[rear].new value of front is
front=front+capacity
no change in the value of rear

after copying, irresespective of segments following procedure should be followed.
capacity=capcity*2; //doubling capacity
free(queue);deleting current queue
queue=newq;// the current queue become newq;

Note:in prescribed book(sahani) he implemented by considering queue is about full(by keeping one place empty).

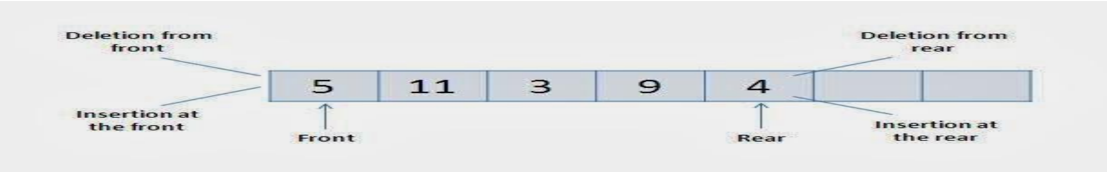
/*function to implement Dynamic circular queue.*/

```
  void qfull()
 {
   char *newq,i=0,j=0;
  // int start=front%SIZE;
   newq=(char *)malloc(2*capacity*sizeof(*newq)); // create a newq by doubling
the memory

   printf("queue is  full double the queue size\n");
   if(front==0)//only when one-segment
   {
     while(i<=capacity-1)
      newq[i++]=queue[i];
   }
   else
   {
     while(i<=rear)  //copying second segment(recently inserted) of queue from queue[0] to rear
     {
     newq[i++]=queue[j++];
     }
     i=i+capacity;
while(front<capacity)//copying first segment(early inserted)of queue from queue[front+1]to capacity-1
     {
     newq[i++]=queue[front++];
     }

     front=rear+(capacity+1); //new value of front in  newq
   }
   capacity=capacity*2;//capcity is doubled
   free(queue);// old queue is deleted
   queue=newq; // newq become old queue
   printf("the size of the queue is doubled re enter your choice,newque size=
%d\n",capacity);
}
```

Double Ended queue(deques):-

A deque is a linear list in which elements can be added or removed at either end but not in the middle.

Deque is maintained by a circular array DEQUE with pointers **front** and **rear.**



<u>There are two variations of Deque are</u>,

1.input restricted deque.

2.output restricted deque.

1.input restricted deque:-

Which allows the insertion from only one end of the list but allows deletions at both ends of the list.

2.output-restricted deque.

Which allows the deletion from only one end of the list but allows insertion at both ends of the list.

Note: insert front:- from right to left - max-1 to 0

Delete fron:- from left to right-   0 to max-1

Insert rear:- from left to right   - 0 to max-1

Delete rear:- from right to left   - max-1 to 0.

```
/* Program of input and output restricted dequeue using array*/
#include<stdio.h>
#define MAX 5
int deque_arr[MAX];
int front = -1;
int rear  = -1;
```

```
void input_que()
{
      int choice;
      while(1)
      {
            printf("1.Insert at rear\n");
            printf("2.Delete from front\n");
            printf("3.Delete from rear\n");
            printf("4.Display\n");
            printf("5.Quit\n");
            printf("Enter your choice : ");
            scanf("%d",&choice);
            switch(choice)
            {
                  case 1:insert_rear();
                        break;
                  case 2:delete_front();
                        break;
                  case 3:delete_rear();
                        break;
                  case 4:display_queue();
                        break;
                  case 5:exit();
                  default:printf("Wrong choice\n");
            }/*End of switch*/
      }/*End of while*/
}/*End of input_que() */
 void output_que()
{
      int choice;
      while(1)
      {
            printf("1.Insert at rear\n");
            printf("2.Insert at front\n");
            printf("3.Delete from front\n");
            printf("4.Display\n");
            printf("5.Quit\n");
            printf("Enter your choice : ");
            scanf("%d",&choice);
            switch(choice)
            {
                  case 1:insert_rear();
                        break;
                  case 2:insert_front();
                        break;
                  case 3:delete_front();
                        break;
                  case 4:display_queue();
                        break;
                  case 5:exit();
                  default:printf("Wrong choice\n");
            }/*End of switch*/
      }/*End of while*/
}/*End of output_que() */
void  insert_rear()
```

```
{
     int added_item;
     if((front == 0 && rear == MAX-1) || (front == rear+1))
     {
          printf("Queue Overflow\n");
          exit(0);
     }
     if (front == -1) /* if queue is initially empty */
     {
          front = 0;
          rear = 0;
     }
     else if(rear == MAX-1) /*rear is at last position of queue */
          rear = 0;
     else
          rear = rear+1;
     printf("Input the element for adding in queue : ");
     scanf("%d", &added_item);
     deque_arr[rear] = added_item ;
}/*End of insert_rear()*/

void insert_front()
{
     int added_item;
     if((front == 0 && rear == MAX-1) || (front == rear+1))
     {
          printf("Queue Overflow \n");
          return;
     }
     if (front == -1)/*If queue is initially empty*/
     {
          front = 0;
          rear = 0;
     }
     else
     if(front== 0)
          front=MAX-1;
     else
          front=front-1;
     printf("Input the element for adding in queue : ");
     scanf("%d", &added_item);
     deque_arr[front] = added_item ;
}/*End of insert_front()*/

void delete_front()
{
     if (front == -1)
     {
          printf("Queue Underflow\n");
          return ;
     }
     printf("Element deleted from queue is : %d\n",deque_arr[front]);
     if(front == rear) /*Queue has only one element */
     {
          front = -1;
```

```
            rear  = -1;
      }
      else if(front == MAX-1)
            front = 0;
      else
            front = front+1;
}/*End of delete_front()*/

void delete_rear()
{
      if (front == -1)
      {
            printf("Queue Underflow\n");
            return ;
      }
      printf("Element deleted from queue is : %d\n",deque_arr[rear]);
      if(front == rear) /*queue has only one element*/
      {
            front = -1;
            rear=-1;
      }
      else if(rear == 0)
            rear=MAX-1;
      else
            rear=rear-1;
}/*End of delete_rear() */

 void display_queue()
{
      int front_pos = front,rear_pos = rear;
      if(front == -1)
      {
            printf("Queue is empty\n");
            return;
      }
      printf("Queue elements :\n");
      if( front_pos <= rear_pos )
      {
            while(front_pos <= rear_pos)
            {
                  printf("%d ",deque_arr[front_pos]);
                  front_pos++;
            }
      }
      else
      {
            while(front_pos <= MAX-1)
            {
                  printf("%d ",deque_arr[front_pos]);
                  front_pos++;
            }
            front_pos = 0;
            while(front_pos <= rear_pos)
            {
                  printf("%d ",deque_arr[front_pos]);
```

```
                front_pos++;
           }
      }/*End of else */
      printf("\n");
}
void main()
{
      int choice;
      printf("1.Input restricted dequeue\n");
      printf("2.Output restricted dequeue\n");
      printf("Enter your choice : ");
      scanf("%d",&choice);
      switch(choice)
      {
           case 1 :input_que();
                break;
           case 2:output_que();
                break;
           default:printf("Wrong choice\n");
      }/*End of switch*/
}/*End of main()*/
```

Priority queues:-

- Collection of elements such that each element has been assigned a priority & such that order in which elements are deleted & processed comes from the following rules

  1.An element of higher priority is processed before any element of lower priority

  2.Two elements with the same priority are processed according to the order

  There are various ways of maintaining a priority queue in memory.out of that two are discused here,

  a. One-way List Representation of priority queue.
  b. Array representation of priority queue.

a.One-way List Representation of priority queue.

•Each node in the list will contain 3 items of info : an info field INFO , a priority number PNR & a link number LINK .

•A node X precedes a node Y in the list

(1) When X has higher priority than Y

(2) When both have the same priority but X was added to the list before Y

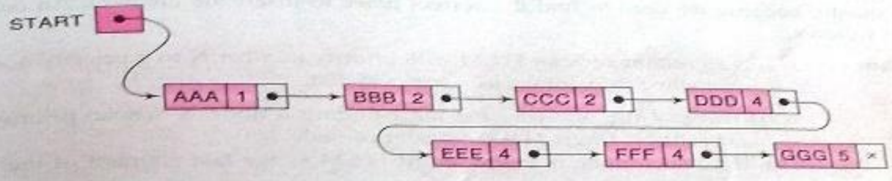
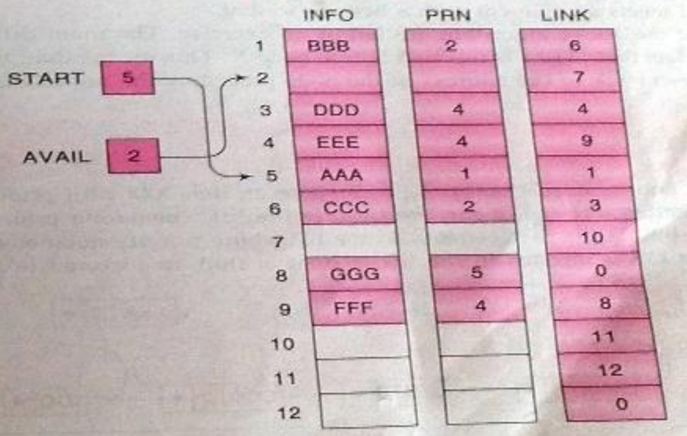


Fig. 6.27

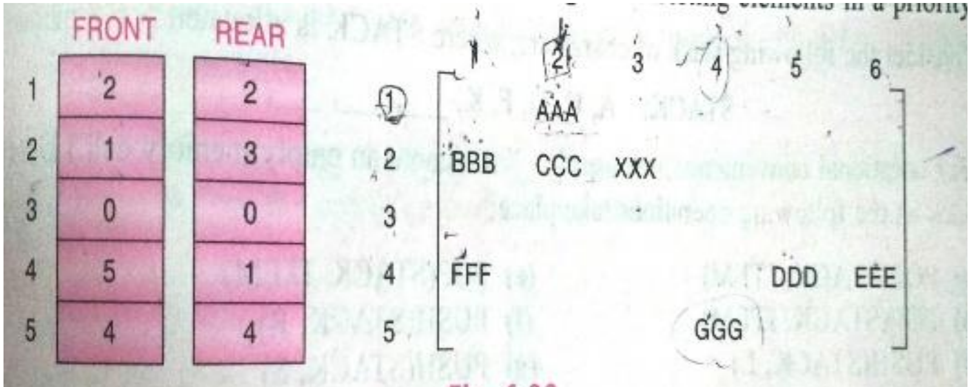


	INFO	PRN	LINK
1	BBB	2	6
2			7
3	DDD	4	4
4	EEE	4	9
5	AAA	1	1
6	CCC	2	3
7			10
8	GGG	5	0
9	FFF	4	8
10			11
11			12
12			0

Fig. 6.28

c. Array Representation of priority queue.

- It uses separate queue for each level of priority.
- Each queue will appear in its own circular array and must have its own pair of pointers i.e FRONT and REAR.
- Each queue is allocated same amount of space,a two dimensional array can be used instead of the linear arrays.



	FRONT	REAR
1	2	2
2	1	3
3	0	0
4	5	1
5	4	4

	1	2	3	4	5	6
1		AAA				
2	BBB	CCC	XXX			
3						
4	FFF				DDD	EEE
5				GGG		

Fig. 6.30

Multiple stacks

* If we Need to implement n stacks, divide the memory into n segments &

* Following notations are used:-

- boundary [i], 0 ≤ i ≤ max-no-of-stacks, points to the position immediatily to the left of the bottom element of stack i.

* top [i], 0 ≤ i < max-no-of-stack points to the top element

* Stack i is empty; iff boundary [i] = top [i].

* The declarations are as follows:-

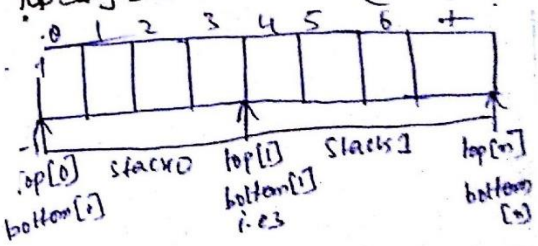
```
#define max-size
# define max-no-of-stacks 10
int top [max-no-of-stacks]
int bottom[max-no-of-stacks]; // bottom & boundary are used
                                           alternatively.
int n; // no. of stacks entered by user.
 top[0] = bottom [0] = -1;
 for(j=1; j<n; j++)
   top [j] = bottom[j] = ((max-size/n)*j)-1;
   bottom [n] = max-size -1;
```

Assume max-size = 8

let the no. of stack = 2

Initialize top[0] = bottom[0] = -1;

top [1] = bottom [1] = ((8/2)*1)-1 = 3



Add an element to stack i.

```
 Void push (int i, int element)
  {
      if ( top[i] == boundary [i+1])
         stackfull(i);      // call function to handle stack full.
        a[++ top[i] ]= element;
  }

int pop (int i)
{
    if (top[i] == bottom[i])
       return stackEmpty(i);
    return ~~mem~~ a [top[i]--];
}
```