CSE 2103 (Data Structures)

Lecture on Chapter-6: Stacks, Queues, Recursion

By

Dr. M. Golam Rashed

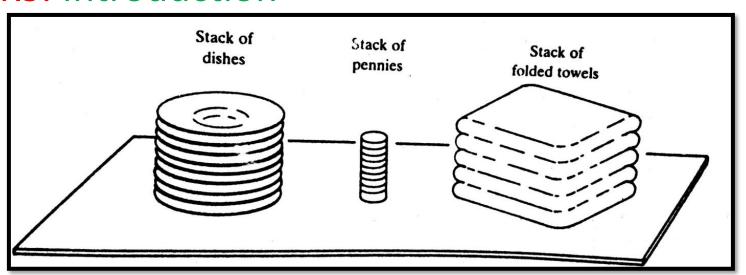
(golamrashed@ru.ac.bd)



Department of Computer Science and Engineering (CSE) Varendra University, Rajshahi, Bangladesh

STACKS: Introduction

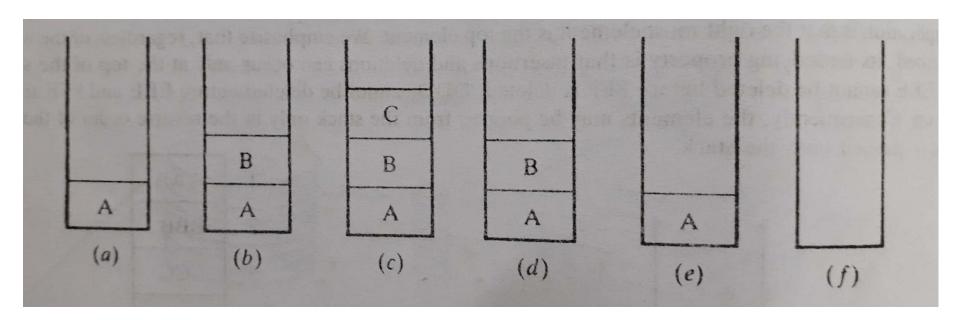




- A stack is a linear structure in which items may be added or removed only at one ends
- Stacks are also called Last-in-First-out (LIFO) list.
- Other names:
 - Piles
 - Push-down lists.
- ➤ Although the stack may be a very restricted type of data structure, it has many important applications in computer science. 2

STACKS: Dealing Postponed Decisions

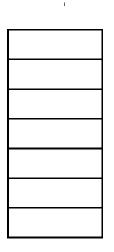
Stacks are frequently used to indicate the order of the processing of data when certain steps of the processing must be postponed until other conditions are fulfilled.



STACKS: Special Terminology



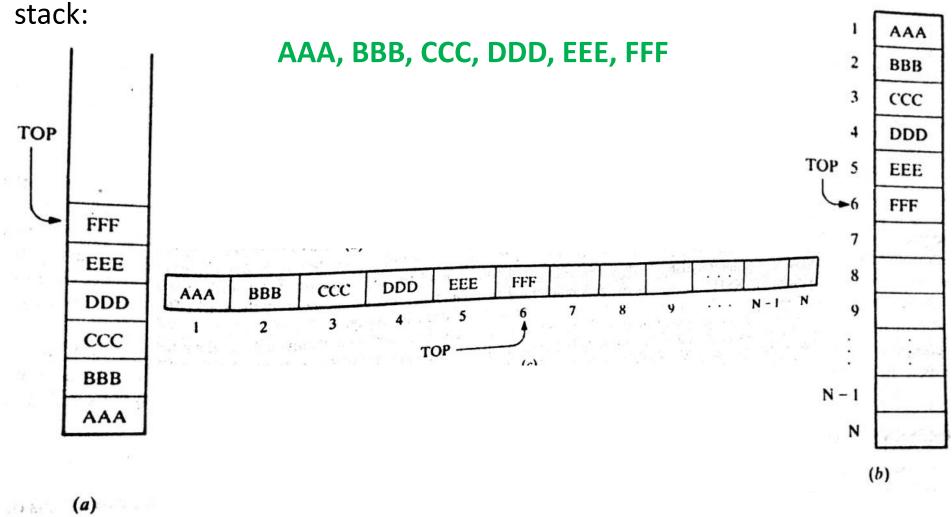
Special terminology is used for two basic operations associated with stacks:



- (a) "Push"-used to insert an element into a stack
- (b) "Pop"-used to delete an element from a stack.

STACKS: Push Example

Suppose the following 6 elements are pushed, in order, onto an empty stack:



STACKS: Array Representation

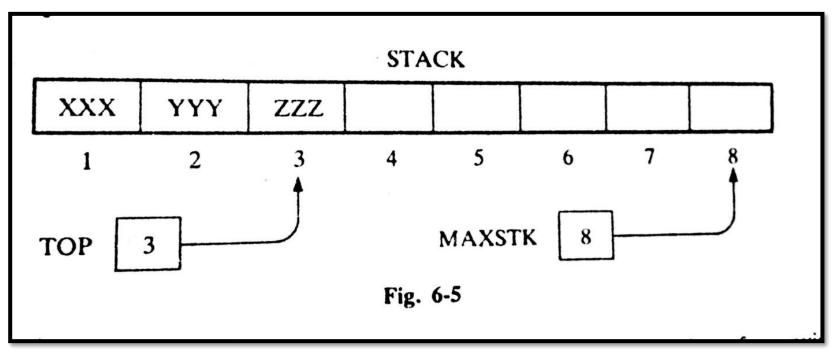


- Stack may be represented in the computer in various ways,
 - ✓ usually by means of a one-way list, or
 - ✓ A linear array
- Usually Stacks will be maintained by......
 - A linear array, STACK, (main lists)
 - A pointer variable, TOP, (contains the locations of the top element of the stack)
 - A variable MAXSTK, (gives the maximum number of elements that can be held by the stack)

The condition TOP=0 or TOP=NULL of a STACK indicates.....?

STACKS: Array Representation Example





Present Status: TOP=3;

MAXSTK=8;

There is room for 5 more items in the stack

Future Implementation:

Pushing an item onto a stack Popping an item from a stack

STACKS: Pushing Algorithm



PUSH (STACK, TOP, MAXSTK, ITEM)

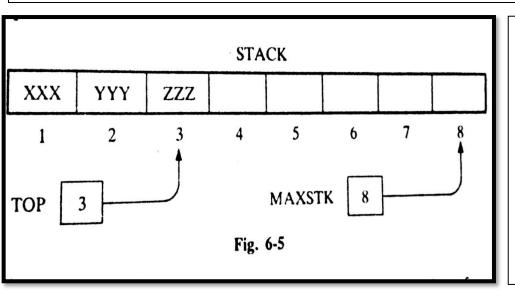
Step 1. [Stack already filled?]

If **TOP=MAXSTK**, then print: **OVERFLOW**, and Return.

Step 2. Set TOP=TOP+1. [Increase TOP by 1.]

Step 3. Set **STACK**[**TOP**]:=**ITEM.** [Insert ITEM in new TOP position]

Step 4. Return



- 1. Since TOP=3, go to Step 2
- 2. TOP=3+1=4
- 3. STACK [TOP]=STACK[4]= www
- 4. Return

STACKS: Popping Algorithm



POP (STACK, TOP, ITEM)

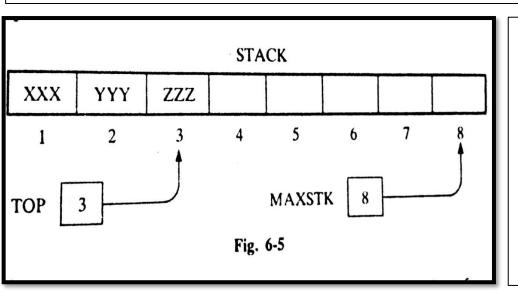
Step 1. [Stack has an item to be removed?]

If **TOP=0**, then print: **Underflow**, and Return.

Step 2. Set **ITEM**= **STACK**[**TOP**] [Assign TOP element to ITEM.]

Step 3. Set **TOP**:= **TOP-1.** [Decreases TOP by 1]

Step 4. Return



- 1. Since TOP=3, go to Step 2
- 2. ITEM=ZZZ.
- 3. TOP=3-1=2.
- 4. Return.

STACKS:



PUSH

- 1. Since TOP=3, go to Step 2
- 2. TOP=3+1=4
- 3. STACK [TOP]=STACK[4]= www
- 4. Return

POP

- 1. Since TOP=3, go to Step 2
- 2. ITEM=ZZZ.
- 3. TOP=3-1=2.
- 4. Return.

- ✓ TOP is changed before the insertion in PUSH, whereas
- ✓ TOP is changed after the deletion in POP

STACKS: Arithmetic Expressions;

- ✓ Let Q be an arithmetic expression involving constants and operation. Let Q be an arithmetic expression involving constants and operation.
 - Q: $2 \uparrow 3+5*2 \uparrow 2-12/6$, we have to find the value of Q
- ✓ Q may have different levels of precedence in its binary operations.
- ✓ We assume the following three levels of precedence for the usual FIVE binary operations.
 - ➤ Highest: Exponential (个)
 - Next Highest: Multiplication (*) and division(/)
 - Lowest: Addition (+) and Subtraction(-)
- \checkmark Thus, we obtain after exponentiations (8+5*4-12/6)
- \checkmark After, multiplication and division (8+20-2)
- \checkmark After. Addition and subtraction , 26

STACKS: Polish Notation



For most common arithmetic operations:

Polish notation refers to the notation in which the operator symbol is placed before its two operands.

Instant Exercise: (Infix expression to polish notation)

$$(A+B)*C = ?$$

= *[A+B]C=? = A+[*BC]=?
= *+ABC = +A*BC

STACKS: Polish Notation



- ✓ The fundamental property of polish notation is that the order in which the operations are to be performed is completely determined by
 - ✓ The positions of the operators, and
 - ✓ Operands in the expression.
- ✓ One never needs parentheses when writing expressions in polish notation.

STACKS: Reverse Polish Notation



Reverse Polish Notation refers to the analogous notation in which the operator symbol is placed after its two operands:

- ✓ This notation is frequently called postfix notation
- ✓One never needs parentheses to determine the order of the operations in any arithmetic expression in *postfix* notation

STACKS: Evaluation of Postfix Expression



P: 5, 6, 2, +, *, 12, 4, /, -

Symbol Scanned	STACK
(1) 5	5
(2) 6	5, 6
(3) 2	5, 6, 2
(4) +	5, 8
(5) *	40
(6) 12	40, 12
(7) 4	40, 12, 4
(8) /	40, 3
(9) -	37
(10))	

STACKS:

Algorithm to find the VALUE of an arithmetic expression PE 210 written in postfix notation.

- 1. Add a right parenthesis "(" at the end of P.
- 2. Scan P from left to right and repeat Steps 3 and 4 for each of P until the ")" is encountered.
- 3. If an operand is encountered, put it on STACK
- 4. If an operator \mathbb{H} is encountered, then:
 - a) Remove the two top elements of the STACK, where A is the top element and B is the next-to-top element
 - b) Evaluate B **₩** A
 - c) Place the result of (b) back on STACK.

[End of If structure]

[End of Step 2 loop]

5. Set VALUE equal to the TOP element on STACK.

16

STACKS: Transforming Q into P expression



Q: A + (B * C- (D / E \uparrow F) * G) * H = (?) P expression

Symbol Scanned	Stack	Expression P
(1) A	(A
(2) +	(+	Α
(3) ((+ (Α
(4) B	(+ (АВ
(5) *	(+ (*	A B
(6) C	(+ (*	ABC
(7) -	(+ (-	A B C *
(8) ((+ (- (A B C *
(9) D	(+ (- (A B C * D
(10) /	(+(-(/	A B C * D
(11) E	(+ (- (/	ABC*DE
(12) 个	(+(-(/↑	ABC*DE
(13) F	(+ (- (/↑	ABC*DEF
(14))	(+ (-	ABC*DEF个/
(15) *	(+ (- *	ABC*DEF个 /

STACKS: Transforming Q into P expression



Q: A + (B * C- (D / E \uparrow F) * G) * H = (?) P expression

Symbol Scanned	Stack	Expression P
(7) -	(+ (-	A B C *
(8) ((+ (- (A B C *
(9) D	(+ (- (A B C * D
(10) /	(+ (- (/	A B C * D
(11) E	(+ (- (/	ABC*DE
(12) 个	(+(-(/↑	ABC*DE
(13) F	(+(-(/个	ABC*DEF
(14))	(+ (-	ABC*DEF个 /
(15) *	(+ (- *	ABC*DEF个 / 18
(16) G	(+ (- *	A B C * D E F 个 / G
(17))	(+	A B C * D E F 个 / G * -
(18) *	(+ *	A B C * D E F 个 / G * -
(19) H	(+ *	A B C * D E F 个 / G * - H
(20))		A B C * D E F 个 / G * - H * +

Instant TEST



Which Expression?

Postfix or Prefix



Equivalent infix expression?

Instant TEST



Infix	Prefix	Postfix
A+B * C+D	++A*BCD	ABC*+D+
(A+B) * (C+D)	*+AB+CD	AB+CD+*
A*B + C*D	+*AB*CD	AB*CD*+
A+B+C+D	++AB+CD	AB+CD++

Instant TEST

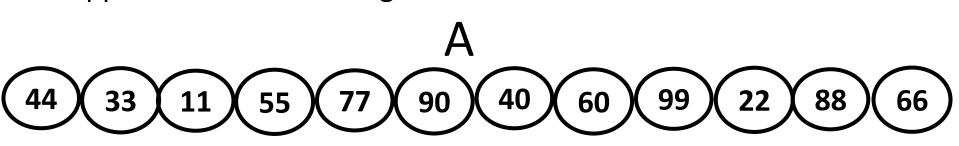


Infix Expression	Prefix Expression	Postfix Expression
1-4^3+7*(9^1/5)-2	-1+^43-*7/^9152	143^-791^5/*+2-
A+B-(C*D^E)*X+Y	+A-B+**C^DEXY	AB+CDE^*X*-Y+

QUICKSORT: An Application of STACKS

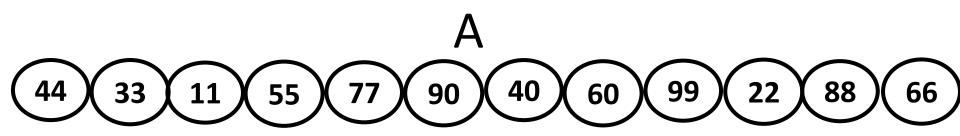


- What is Sorting?
 - ✓ The operation of rearranging the elements of a list so that they are in some logical order.
 - Numerically ordered (When list contain numerical data)
 - Alphabetically ordered (When list contains character data)
- Quicksort is an algorithm of the DIVIDE-AND-CONQUER type.
 - The problem of sorting a set is reduced to the problem of sorting two smaller sets.
- We illustrate this "Reduced Step" by means of a specific example
- Suppose A is the following list of 12 numbers.



Reduction Step of the Quicksort Algorithm

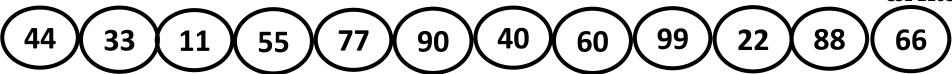


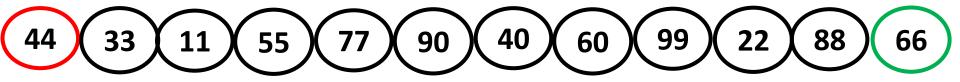


- The reduction step of the quicksort algorithm finds the final position of one of the numbers.
- In this illustration, we use the first number 44.

STACKS: Quicksort/ Reduction Step



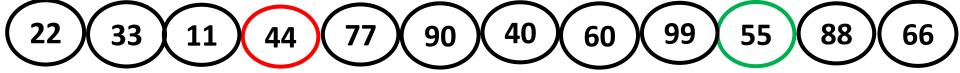




- Beginning with the last number, 66, scan the list from right to left till less than 44.
- ✓ Interchange 44 and 22



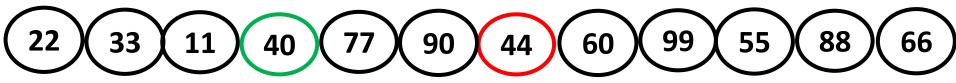
- ✓ Beginning with, 22, scan left to right till greater than 44
- ✓ Interchange 44 and 55



- ✓ Beginning with, 55, scan the list from right to left till less than 44.
 - ✓ Interchange 44 and 40

STACKS: Quicksort/ Reduction Step

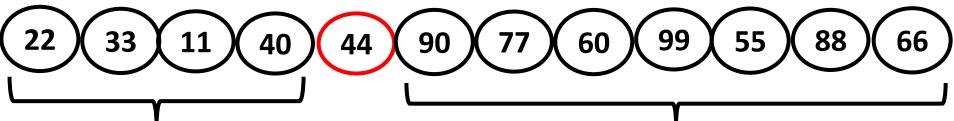




- ✓ Beginning with, 40, scan left to right till greater than 44
- ✓ Interchange 44 and 77



- ✓ Beginning with, 77, scan the list from right to left till less than 44.
- ✓ Do not meet such a number before meeting 44.



First sublist

Second sublist

- √ Thus, 44 is correctly placed in its final position.
- ✓ The task of sorting the original list A has now been reduced to the task of sorting each of the above sublists.

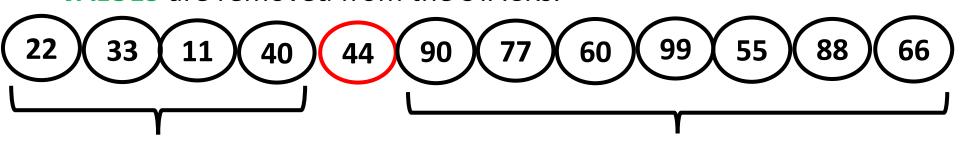
STACKS: Quicksort/ Reduction Step

- CSE 2103
- The reduction step is repeated with each sublist containing 2^{section}
 more elements.
- Since we can process one sublist at a time, we must be able to keep track of some sublist for future processing.
- This is accomplished by using two STACKS

LOWER and UPPER

to temporarily hold such sub-lists.

- The addresses of the first and last elements of each sublist, called its "BOUNDARY VALUES", are pushed onto the STACKs LOWER and UPPER, respectively.
- The reduction steps is applied to a sublist only after its BOUNDARY VALUES are removed from the STACKs.

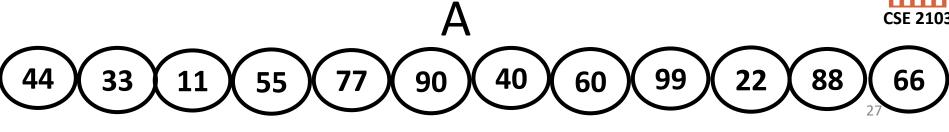


First sublist

Second sublist

STACKS: Illustration of the way the STACKS LOWER and UPPER are used





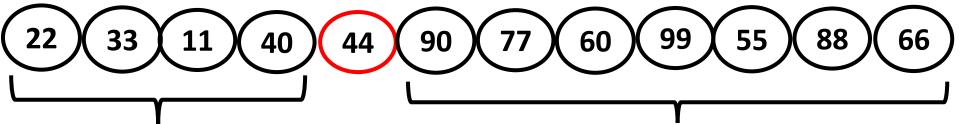
N=12 elements,
Thus,
Boundary values are ()?
1 and 12.
Now,
1 and 12 should be Stacked
LOWER:1 and UPPER:12

- In order to apply the REDUCTION STEP, the algorithm first removes the top values 1 and 12.
- After removing the top values 1 and 12 from the STACK, leaving LOWER: (Empty) and UPPER: (Empty)
- Then Applies the REDUCTION STEP to the corresponding list A[1], A[2],...,A[12].

STACKS: Illustration of the way the STACKS LOWER and UPPER are used



- After executing REDUCTION STEP to the list A[1] to A[12]
 - Finally places the first element 44, in A[5].



First sublist

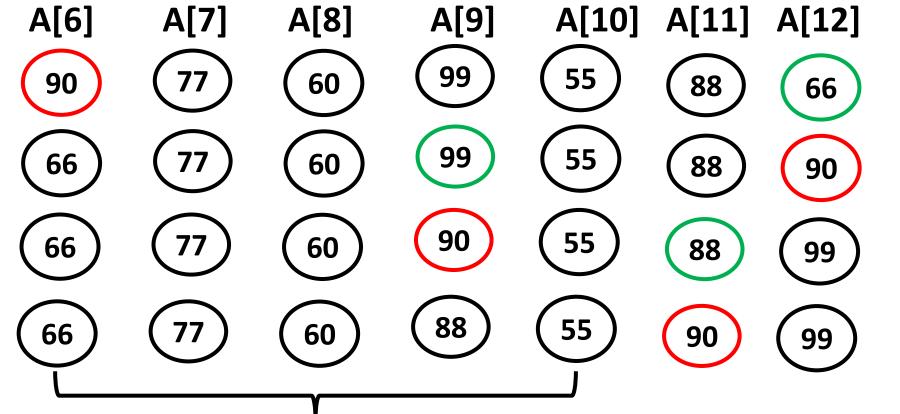
Second sublist

- Accordingly, the algorithm pushes the boundary values
 - 1 and 4 of the first sublist, and
 - 6 and 12 of the second sublist on to the STACK to yield
 LOWER= 1, 6 and UPPER= 4, 12
- In order to apply the REDUCTION STEP again, the algorithm removes the TOP values 6 and 12 from the STACKs, leaving LOWER= 1, and UPPER= 4

28

STACKS: Illustration of the way the STACKS LOWER and UPPER are used





Second sublist

First sublist

- The second sublist has only one element, Accordingly
- The algorithm pushes only the boundary values 6 and 10 of the first sublist on the STACKs to yield
 - LOWER= 1, 6 and UPPER= 4, 10

QUICKSORT

- CSE 2103
- The quick sort is regarded as the best sorting algorithm.
- This is because of its significant advantage in terms of efficiency because it is able to deal well with a huge list of items.
- Because it sorts in place, no additional storage is required as well.
- The slight disadvantage of quick sort is that its worst-case performance is similar to average performances of the bubble, insertion or selections sorts.
- In general, the quick sort produces the most effective and widely used method of sorting a list of any item size.

Recursion: Function

- CSE 2103
- ✓ A function is said to be Recursively defined if the function definition refers to itself.
- ✓ A *Recursive Function* must have the following two properties:
 - There must be certain arguments, called **BASE VALUE**, for which the function does not refer to itself.
 - ➤ Each time the function does refer to itself, the argument of the function must be closer to a **BASE VALUE**.
- ✓ A Recursive Function with two properties is also said to be well-defined.

Recursion: Factorial Function



- ✓ In some problems, it may be natural to define the problem in terms of the problem itself.
- ✓ Recursion is useful for problems that can be represented by a SIMPLER VERSION of the same problem.
- ✓ Example: the factorial function

$$6! = 6 * 5 * 4 * 3 * 2 * 1$$

We could write:

$$6! = 6 * 5!$$

Recursion: Factorial Function



✓ In general, we can express the factorial function as follows:

$$n! = n * (n-1)!$$
Is this correct? Well... almost.

✓ The factorial function is **ONLY DEFINED** for *positive* integers. So we should be a bit more precise:

```
i) n! = 1 (if n is equal to 1)
ii) n! = n * (n-1)! (if n is larger than 1)
```

- ✓ Observe that, this definition of n! is recursive, since it refers to itself when it uses (n-1)!, However,
- \checkmark i) the value of n! is explicitly given when n=0 (BASE VALUE)

value of n which is closer to the BASE VALUE

 \checkmark ii) the value of n! for arbitrary n is defined in terms of a smaller

Recursion: Factorial Function



EXAMPLE: Let's calculate 3! Using the recursive definition.

- (6) 2! = 2 . 1 = 2
- $(7) \qquad 3! = 3 \quad . \quad 2 = 6$
 - ✓ Observe that we back track in the reverse order of the original postponded evaluations.
 - ✓ Recall that this type of postponed processing tends itself to the use of STACKS.

Recursion: Function



Assume the number typed is 3, that is, numb=3.

```
fac(3):
                         No.
3 <= 1 ?
fac(3) = 3 * fac(2)
  fac(2):
                         No.
      fac(2) = 2 * fac(1)
                              i) n! = 1 (if n is equal to 1)
                              ii) n! = n * (n-1)! (if n is larger than 1)
         fac(1):
1 <= 1 ? Yes.
           return 1
                             int fac(int numb) {
      fac(2) = 2 * 1 = 2
                               if(numb <= 1)
      return fac(2)
                                  return 1;
                               else
 fac(3) = 3 * 2 = 6
                                  return numb * fac(numb-1);
 return fac(3)
```

fac(3) has the value 6

Recursion: Function



For certain problems (such as the factorial function), a recursive solution often leads to short and elegant code. Compare the recursive solution with the iterative solution:

Recursive solution

```
int fac(int numb) {
  if(numb<=1)
    return 1;
  else
    return numb*fac(numb-1);
}</pre>
```

Iterative solution

```
int fac(int numb) {
int product=1;
  while(numb>1) {
    product *= numb;
    numb--;
    }
  return product;
}
```

Recursion: Overhead

CSE 2103

■Space:

- Every invocation of a function call requires:
 - space for parameters and local variables
 - space for return address
- ■Thus, a recursive algorithm needs space proportional to the number of nested calls to the same function.

■Time:

- Calling a function involves
 - •allocating, and later releasing, local memory
 - copying values into the local memory for the parameters
 - branching to/returning from the function
- •All contribute to the time overhead.

Recursion: Cost



- ✓ You have to pay a price for recursion:
 - Calling a function CONSUMES MORE TIME AND MEMORY than adjusting a loop counter.
 - ➤ High performance applications (graphic action games, simulations of nuclear explosions) hardly ever use recursion.
- ✓ In **LESS DEMANDING APPLICATIONS** recursion is an attractive alternative for iteration.

Recursion: Precautions



You must always make sure that the recursion bottoms out:

- ➤ A recursive function must contain at least one non-recursive branch.
- > The recursive calls must eventually lead to a non-recursive branch.

```
int fac(int numb) {
   if(numb<=1)
     return 1;
   else
     return numb * fac(numb-1);
}</pre>
```

How many pairs of rabbits can be produced from single pair in a year's time?

Assumptions:

- Each pair of rabbits produces a new pair of offspring every month;
- each new pair becomes fertile at the age of one month;
- none of the rabbits dies in that year.

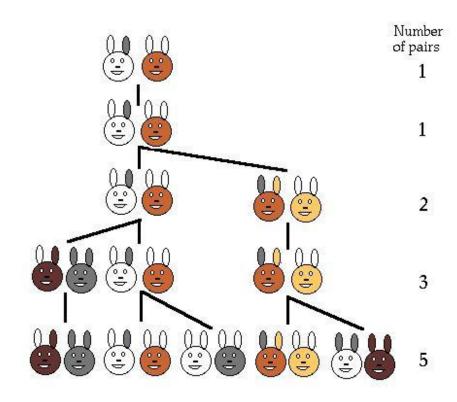
Example:

- After 1 month there will be 2 pairs of rabbits;
- after 2 months, there will be 3 pairs;
- after 3 months, there will be 5 pairs (since the following month the original pair and the pair born during the first month will both produce a new pair and there will be 5 in all).



CSE 2103

Population Growth in Nature



- ✓ Leonardo Pisano (Leonardo Fibonacci, son of Bonaccio) proposed the (Fibonacci) sequence in 1202 in *The Book of the Abacus*.
- ✓ Fibonacci numbers are believed to model nature to a certain extent, such as Kepler's observation of leaves and flowers in 1611.



Direct Computation Method

Fibonacci numbers:

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ... where each number is the sum of the preceding two.
```

Recursive definition:

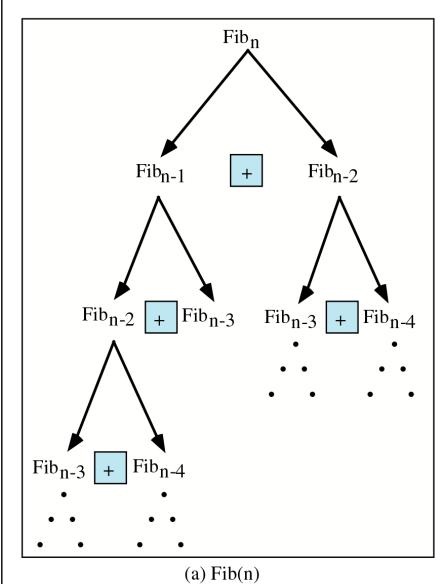
```
F(0) = 0;
F(1) = 1;
F(number) = F(number-1) + F(number-2);
```

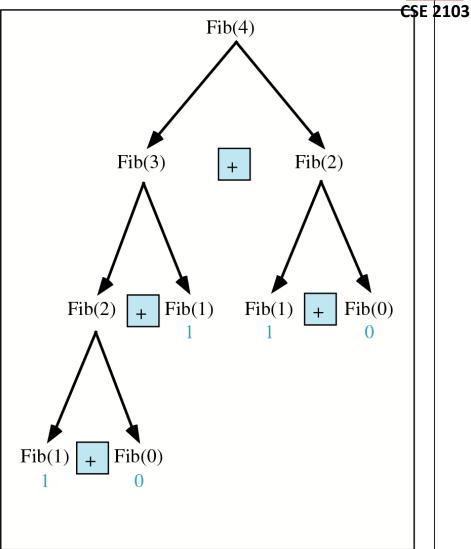


```
// Calculate Fibonacci numbers using recursive function.
// A very inefficient way, but illustrates recursion well
int fib(int number)
                                    Recursive definition:
                                    F(0) = 0;
       if (number == 0) return 0; | F(1) = 1;
       if (number == 1) return 1; F(number) = F(number-1) + F(number-2);
       return (fib(number-1) + fib(number-2));
int main() { // driver function
       int inp number;
       printf("Please enter an integer: ");
       scanf("%d"&inp number);
       cout << "The Fibonacci number for "<< inp_number</pre>
               << " is "<< fib(inp_number)<<endl;</pre>
  return 0:
```

Recursion: Fibbonacci







(b) Fib(4)

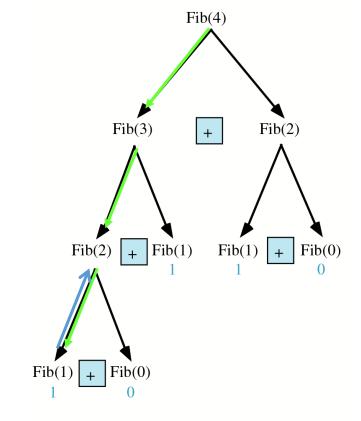
Recursion: Trace a Fibonacci Number



Assume the input number is 4, that is, num=4:

```
fib(4):
   4 == 0 ? No; 4 == 1? No.
   fib(4) = fib(3) + fib(2)
   fib(3):
       3 == 0 ? No; 3 == 1? No.
       fib(3) = fib(2) + fib(1)
       fib(2):
           2 == 0? \text{ No}; 2==1? \text{ No}.
           fib(2) = fib(1) + fib(0)
          fib(1):
              1 == 0 ? No 1 == 1? Yes.
               fib(1) = 1
                      return fib(1);
```

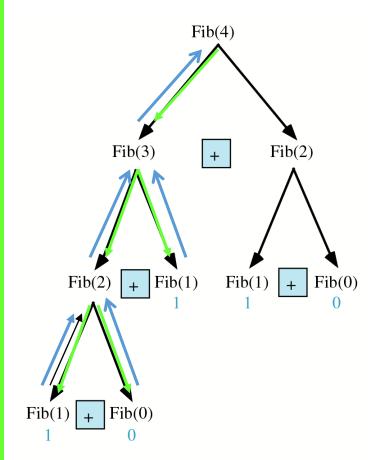
```
int fib(int num)
{
    if (num == 0) return 0;
    if (num == 1) return 1;
    return
        (fib(num-1)+fib(num-2));
}
```



Recursion: Trace a Fibonacci Number



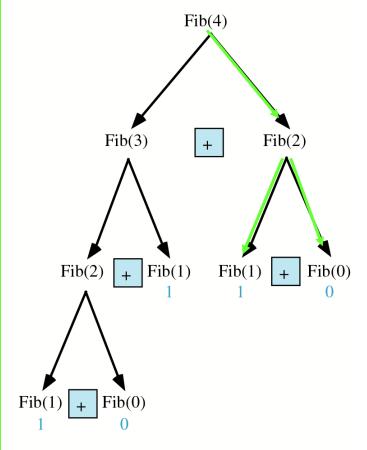
```
fib(0):
            0 == 0 ? Yes.
            fib(0) = 0;
            return fib(0);
  fib(2) = 1 + 0 = 1;
  return fib(2);
fib(3) = 1 + fib(1)
   fib(1):
    1 == 0 ? No; 1 == 1? Yes
    fib(1) = 1;
    return fib(1);
fib(3) = 1 + 1 = 2;
return fib(3)
```



Recursion:



```
fib(2):
  2 == 0 ? No; 2 == 1? No.
  fib(2) = fib(1) + fib(0)
  fib(1):
     1 == 0 ? No; 1 == 1? Yes.
      fib(1) = 1;
      return fib(1);
   fib(0):
      0 == 0? Yes.
     fib(0) = 0;
      return fib(0);
   fib(2) = 1 + 0 = 1;
   return fib(2);
fib(4) = fib(3) + fib(2)
       = 2 + 1 = 3;
return fib(4);
```



Recursion: Divide-and-Conquer Algorithms



- ✓ Consider a problem P associated with a set S.
- ✓ Suppose A is an algorithm which partitions S into smaller sets such that the solution of the problem P for S is reduced to the solution of P for one or more of the smaller sets.
- ✓ Then A is called a divide-and-conquer Algorithm.
- ✓ Examples:

The Quicksort Algorithm

-use reduction step to find the location of a single element and to reduce the problem of sorting the entire set to the problem of sorting smaller sets

The Binary Search Algorithm

-divides the given sorted set into two halves so that the problem of searching for an item in the entire set is reduced to the problem of searching for the item in one of the two halves.

Recursion: Divide-and-Conquer Algorithms

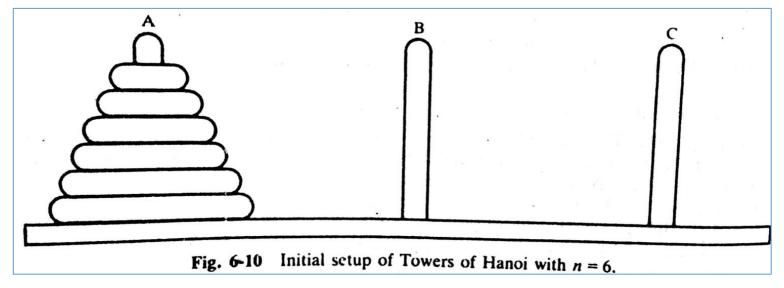


- ✓ A divide-and-conquer algorithm A may be viewed as a recursive procedure. But Why?
 - The divide-and-conquer algorithm A may be viewed as calling itself when it is applied to the smaller sets.
 - The base criteria for these algorithms are usually the oneelement sets.
 - For example, with a sorting algorithm, a one-element set is automatically sorted, and
 - > with a searching algorithm, one-element set requires only a single comparison.

TOWER of HANOI Problem

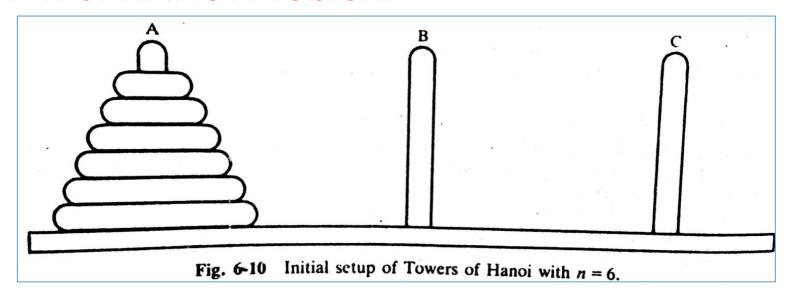






TOWER of HANOI Problem



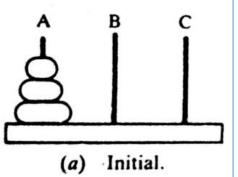


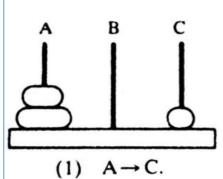
- ✓ Only one disc could be moved at a time
- ✓ A larger disc must never be stacked above a smaller one
- ✓ One and only one extra needle could be used for intermediate storage of discs

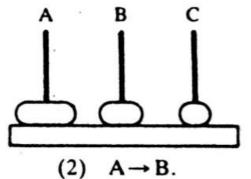
TOWER of HANOI Problem Solution

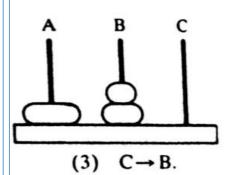


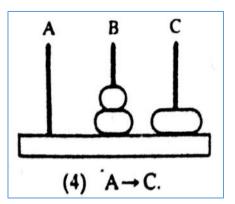
The solution to the Tower of Hanoi problem for n=3 appears

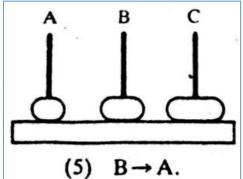


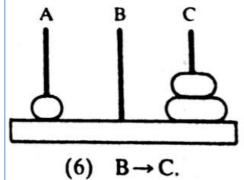


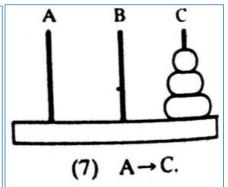












$$A \rightarrow C$$
, $A \rightarrow B$, $C \rightarrow B$, $A \rightarrow C$, $B \rightarrow A$,

B→C,

A→C,

TOWER of HANOI Problem Solution



The **SEPARATE SOLUTION** to the Tower of Hanoi problem

Solution: $A \rightarrow B$, $A \rightarrow C$, $B \rightarrow C$

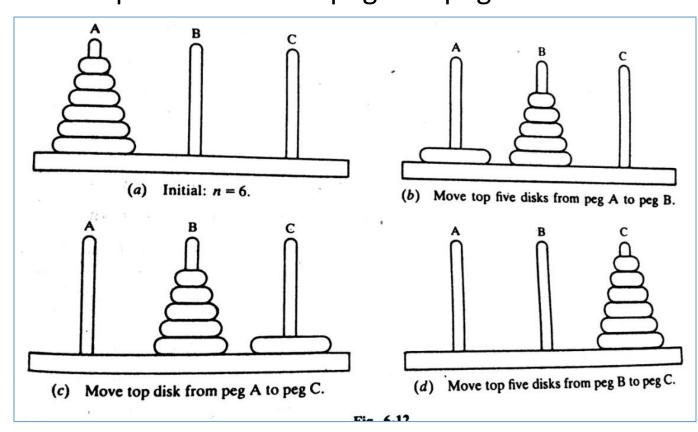
n=2

Solution: $A \rightarrow C$, $A \rightarrow B$, $C \rightarrow B$, $A \rightarrow C$, $B \rightarrow A$, $B \rightarrow C$, $A \rightarrow C$

- ✓ General Solution to the Tower of Hanoi for any # of Disk is PREFERRED.
- ✓ Which can be done in RECURSIVE way.

Solution of TOWER of HANOI in RECURSIVE WAY

- ✓ The solution to the Tower of Hanoi problem for n>1 disks may be reduced to the following sub-problems:
 - ➤ Move the top n-1 disks from peg A to peg B.
 - \triangleright Move the top disk from A to peg C: $A \rightarrow C$
 - ➤ Move the top n-1 disks from peg B to peg C



Solution of TOWER of HANOI in RECURSIVE WAY



The general notation of the solution for any # of disk:

TOWER(N, BEG, AUX, END)

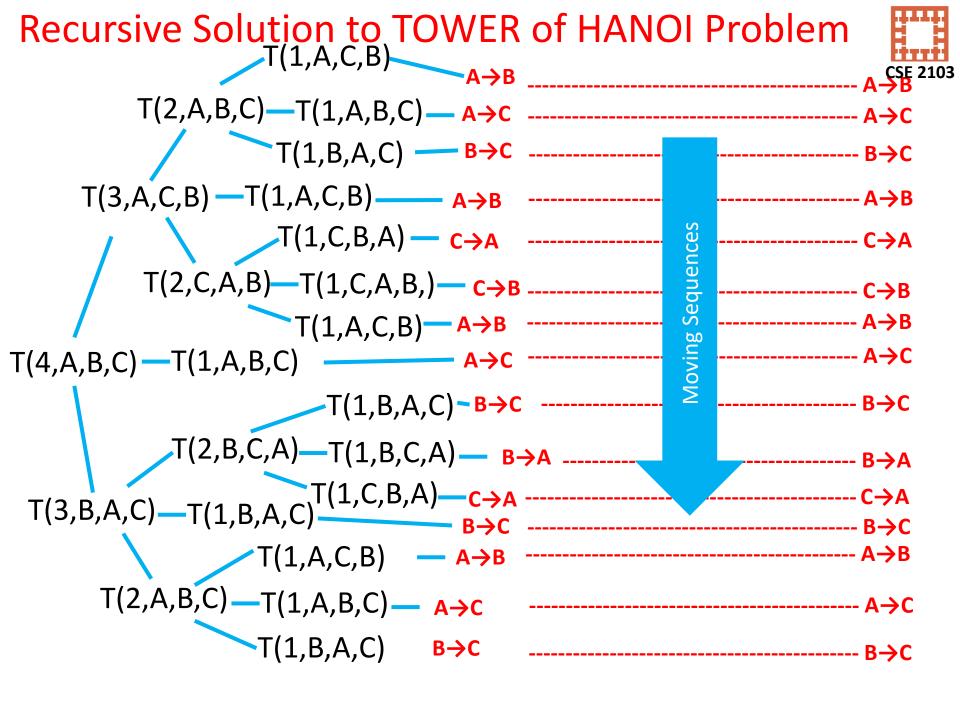
When **n=1** then

TOWER(1, BEG, AUX, END)

 $BEG \rightarrow END$, But

When **n> 1** then then solution may be reduced to the solution of the following three sub-problem:

- (1) TOWER (N-1, BEG, END, AUX)
- (2) TOWER (1, BEG, AUX, END)
- (3) TOWER (N-1, AUX, BEG, END)



Formal TOWER of HANOI Problem Solving Procedure



TOWER (N, BEG, AUX, END)

```
Step1. If N=1, then:

(a) Write: BEG→END.

(b) Return.
```

[End of If Structure]

```
Step2. [Move N-1 disks from peg BEG to peg AUX.]
Call TOWER (N-1, BEG, END, AUX).
```

Step3. Write: BEG \rightarrow END.

Step4. [Move N-1 disks from peg AUX to peg END.]
Call TOWER (N-1), AUG, BEG, END).

Step5. Return

Implementation of Recursive Pro. By STACKS



Before implementing Recursive procedures.....

Imagine a SUBPROGRAM...

- ✓ It may contain both parameters, and local variables.
- ✓ The parameters are the variables which receive values from objects in the CALLING PROGRAM.
- ✓ It transmit values back to the CALLING PROGRAM.
- ✓ It must also keep track of the return address in the CALLING PROGRAM.
- ✓ The return address is essential, since control must be transferred back to its proper place in the CALLING PROGRAM.
- ✓ Once the subprogram finished executing and control is transferred back to the CALLING PROGRAM.
- ✓ The values of the local variables and the return address are no longer needed.

Implementation of Recursive Pro. By STACKS



Suppose SUBPROGRAM is a recursive program...then

- ✓ Each level of execution of the subprogram may contain different values for the parameters and local variables, and for the return address.
- ✓ Furthermore, <u>if the recursive program does call itself,</u> then these current values must be saved, since they will be used again when the program is reactivated.

Queues



- ✓ A queue is a linear list of elements in which...
 - deletions can take place only at one end, called the front, and
 - insertions can take place only at the other end, called the rear.
- ✓ Queues are also called first-in first-out (FIFO) list
- ✓ This contrasts with stacks, which are LIFO

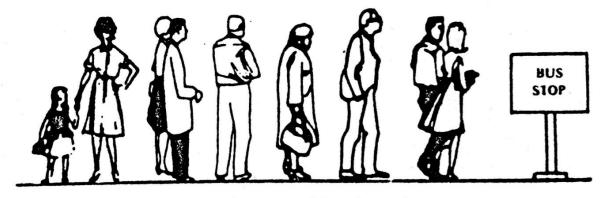


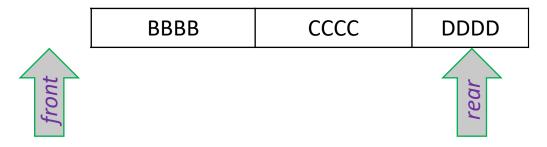
Fig. 6-2 Queue waiting for a bus.

✓ An important example of a queue in computer science occurs in a timesharing system in which programs with the same priority form a queue while waiting to be executed.

Queues: Example

CSE 2103

Consider a Queue with 4 elements



✓ Suppose an element is deleted from the 4 element queue.

Which is we can delete??

AAAA

Then which one is considered to be the front element?

BBBB

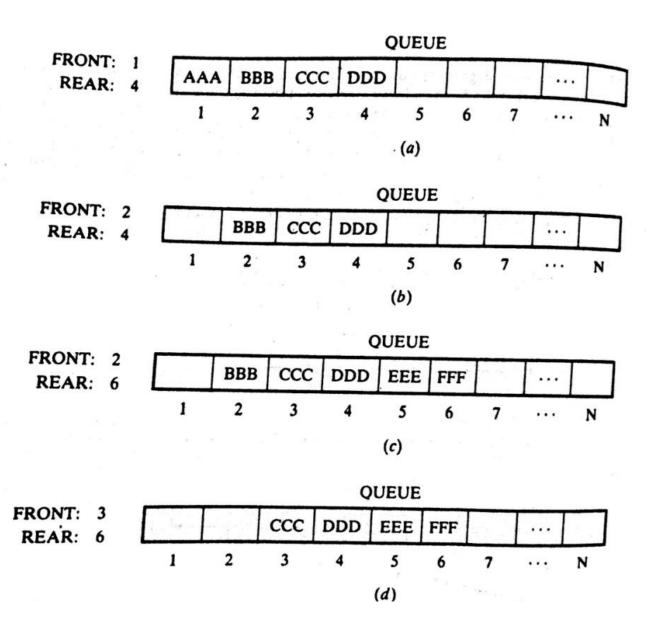
Suppose EEEE is added to the queue

Then which one is considered to be the rear element? EEEE

- CSE 2103
- ✓ Queues may be represented in the computer in various ways.
- ✓ Usually by means of
 - One-way list
 - Linear arrays
- ✓ Each of the queues will be maintained by ...
 - A Linear Array QUEUE
 - Two pointer variables:
 - FRONT (containing the location of the front element of the Queue)
 - REAR (containing the location of the rear element of the Queue)
- ✓ The condition FRONT= NULL will indicate that the Queue is......

 EMPTY.



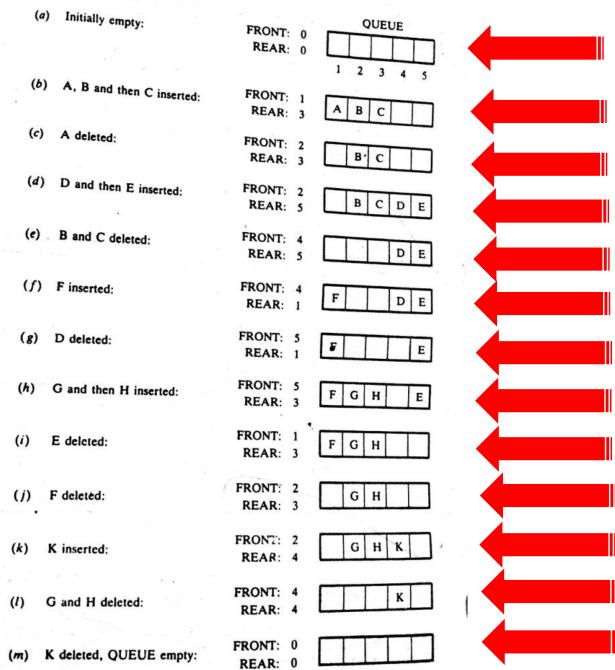


- ✓ After N insertion, the REAR element of the QUEUE will occupy QUEUE [N]
- ✓ This occurs even though the queue itself may not contain many elements
- ✓ Suppose, we want to insert an element **ITEM** into a queue at the time the queue does occupy the last part of the array, that is **REAR=N**.
- ✓ One way to do this is to simply move the entire queue to the beginning of the array changing FRONT and REAR accordingly.
- ✓ Then insert item as above.
- √ This procedure is very expensive
- ✓ The procedure we adopt is to assume that the array QUEUE is Circular. ie, QUEUE[1] comes after QUEUE[N]



- ✓ Similarly, if FRONT=N and an element of Queue is deleted.
- ✓ We reset FRONT=1, instead of increasing FRONT to N+1.





Queues: Insertion Algorithm



QINSERT (QUEUE, N, FRONT, REAR, ITEM)

```
Step1. [QUEUE already filled?]
       If FRONT=1 and REAR=N, or FRONT=REAR+1, then:
              Write: OVERFLOW, and Return
Step2. [Find new value of REAR.]
       If FRONT:=NULL, then: [Queue initially empty]
              Set FRONT:=1 and REAR:=1
       Else if REAR=N, then:
              Set REAR:=1,
       Else:
              Set REAR:=REAR+1. [End of If structure]
Step3. Set QUEUE[REAR]=ITEM [This inserts new item]
```

Step4. Return

Queues: Deletion Algorithm



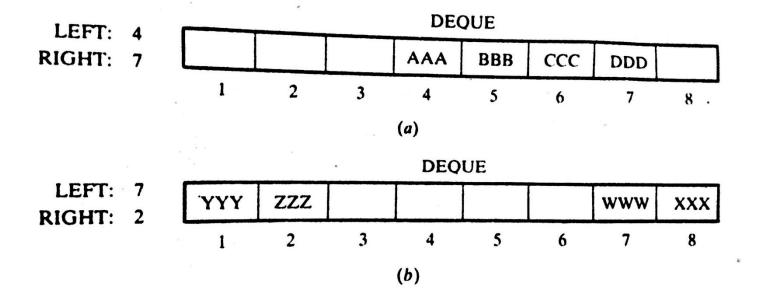
QDELETION (QUEUE, N, FRONT, REAR, ITEM)

```
Step1. [QUEUE already empty?]
       If FRONT=NULL then:
              Write: UNDERFLOW, and Return
Step2. Set ITEM:= QUEUE[FRONT],
Set3. [Find new value of FRONT.]
       If FRONT:=REAR, then: [Queue has only one element to start]
              Set FRONT:=NULL and REAR:=NULL
       Else if FRONT=N, then:
              Set FRONT:=1,
       Else:
              Set FRONT:=FRONT+1.
[End of If structure]
```

Step4. Return

Deques: Definition

- ✓ A deque is a linear list in which elements can be added or removed at either end but not in the middle.
- ✓ Deque can be maintained by......
 - a CIRCULAR array
 - Pointer LEFT-which points left end of the deque
 - > Pointer RIGHT-which points right end of the deque.



Deques: Variation



- ✓ There are TWO variations of a Deque-
 - An Input-restricted deque, and
 - An Output-restricted deque
- ✓ This are intermediate between a Deque and Queue

- ✓ An Input-restricted deque is a deque which allow insertion at only one end of the list but allows deletions at both ends of the list
- ✓ An output-restricted deque is a deque which allows deletions at only one end of the list but allows insertions at both ends of the list.

Priority Queues: Definition



A priority queue is a collection of elements such that each elements has been assigned a priority and such that the order in which elements are deleted and processed comes from the following rules:

- An element of higher priority is processed before any element of lower priority
- Two elements with same priority are processed according to the order in which they were added to the queue.

Priority Queues: One-way List Representation



One way to maintain a priority queue in memory is by means of a oneway list, as follows

- Each node in the list will contain three items of information:
 - > An Information field INFO,
 - A priority number PRN, and
 - A link number LINK
- b) A node X precedes a node Y in the list
 - (1) when X has higher priority then Y
- (2) When both have the same priority but X was added to the list before Y
- Priority numbers will operate in the usual way: the Lower the priority number, the higher the priority.

Priority Queues: Schematic Diagram with 7 elemen



0

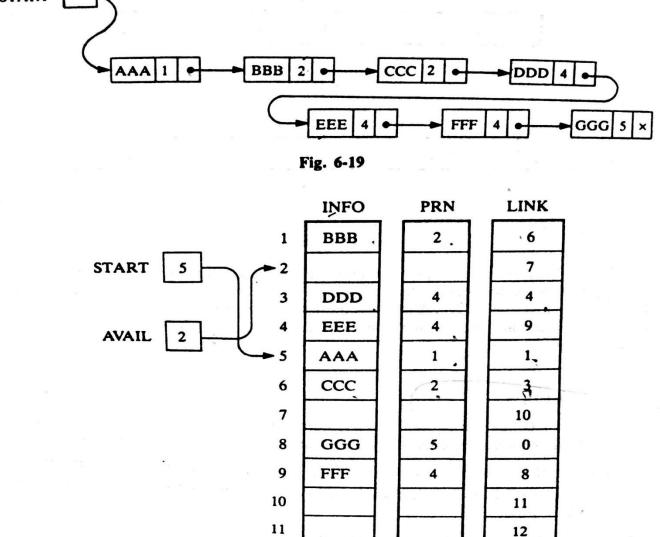


Fig. 6-20

12

Upcoming Presentation from Students:



Implement STACK using one QUEUE

Implements STACK using two QUEUES

Implement QUEUE using one STACK

Implement QUEUE using two STACK

CSE 2103

LOOK at to the SOLVED Problems

6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10

6.11, 6.12

6.14, 6.15, 6.16, 6.17

6.21,