Data Structure & Algorithms (DSA) : CCT203

Semester - III, B.Tech. CSE (Cyber Security)

UNIT II

GC Code: dyb4gii

Course Objectives

1. CO1: To impart to students the basic concepts of data structures and algorithms.
2. CO2: To familiarize students on different searching and sorting techniques.
3. CO3: To prepare students to use linear (stacks, queues, linked lists) and nonlinear (trees, graphs) data structures.
4. CO4: To enable students to devise algorithms for solving real-world problems.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Text Books & Reference Books

Text Books:

1. Ellis Horowitz, Sartaj Sahni & Susan Anderson-Freed, Fundamentals of Data Structures in C, Second Edition, Universities Press, 2008.
2. Mark Allen Weiss; Data Structures and Algorithm Analysis in C; Second Edition; Pearson Education; 2002.
3. G.A.V. Pai; Data Structures and Algorithms: Concepts, Techniques and Application; First Edition; McGraw Hill; 2008.

Reference Books:

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein; Introduction to Algorithms; Third Edition; PHI Learning; 2009.
2. Ellis Horowitz, Sartaj Sahni and Sanguthevar Rajasekaran; Fundamentals of Computer Algorithms; Second Edition; Universities Press; 2008.
3. A. K. Sharma; Data Structures using C, Second Edition, Pearson Education, 2013.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Course Outcomes

On completion of the course the student will be able to

1. Recognize different ADTs and their operations and specify their complexities.
2. Design and realize linear data structures (stacks, queues, linked lists) and analyze their computation complexity.
3. Devise different sorting (comparison based, divide-and-conquer, distributive, and tree-based) and searching (linear, binary) methods and analyze their time and space requirements.
4. Design traversal and path finding algorithms for Trees and Graphs.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

UNIT II Stacks and Queues

- **Stack ADT:**
  - Introduction and its representation.
  - Allowable operations
  - Algorithms and their complexity analysis
  - Applications of stacks – expression conversion and evaluation (algorithmic analysis)
  - Multiple stacks.
- **Queue ADT:**
  - Allowable operations
  - Algorithms and their complexity analysis for simple queue and circular queue
  - Introduction to double-ended queues and priority queues.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Introduction

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Introduction

- A stack is a linear data structure which the elements in a stack are added and removed only from one end, which is called the TOP.
- A stack is called a LIFO (Last-In-First-Out) data structure, as the element that was inserted last is the first one to be taken out.

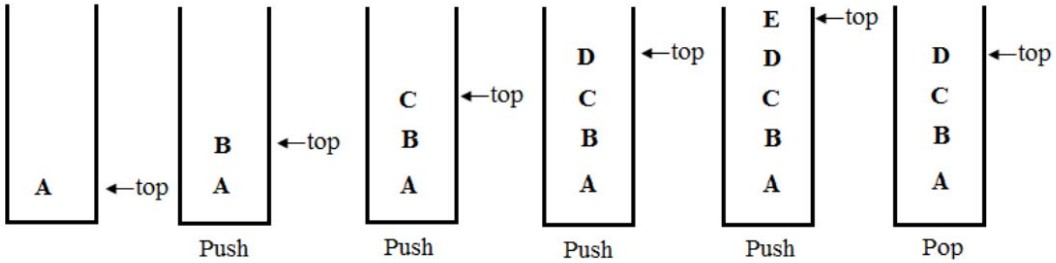
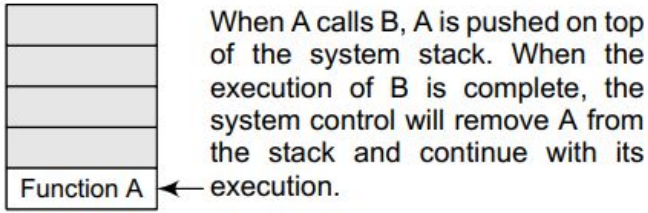


Figure: Inserting and deleting elements in a stack

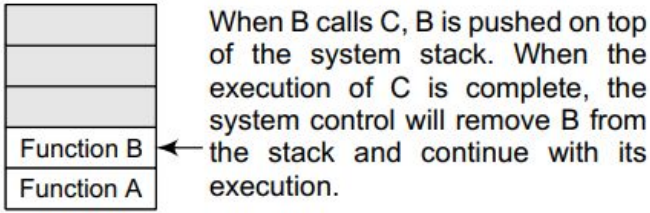
Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Introduction

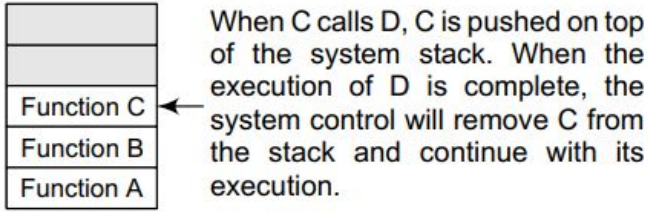
- Why Stack?
- The answer is in **function calls**.
- Consider an example, where we are executing function A. In the course of its execution, function A calls another function B. Function B in turn calls another function C, which calls function D.



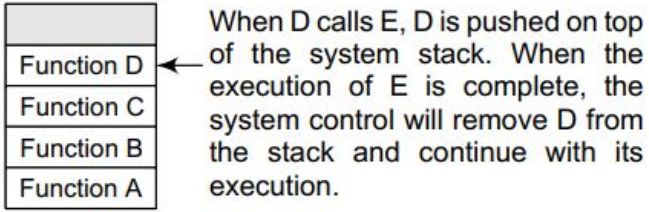
When A calls B, A is pushed on top of the system stack. When the execution of B is complete, the system control will remove A from the stack and continue with its execution.



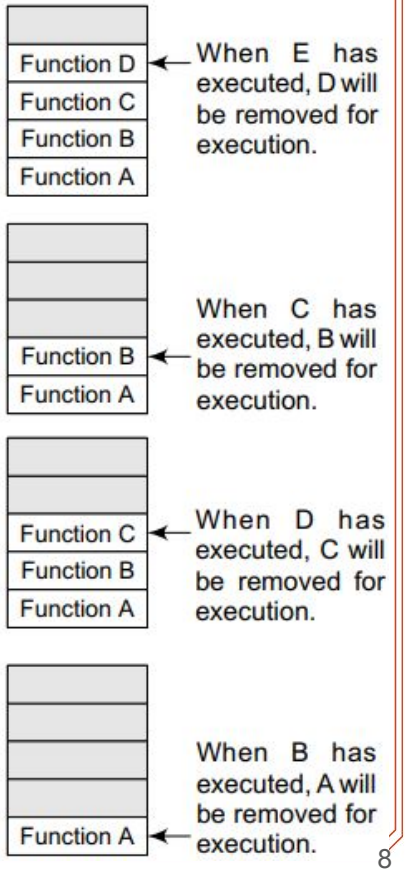
When B calls C, B is pushed on top of the system stack. When the execution of C is complete, the system control will remove B from the stack and continue with its execution.



When C calls D, C is pushed on top of the system stack. When the execution of D is complete, the system control will remove C from the stack and continue with its execution.

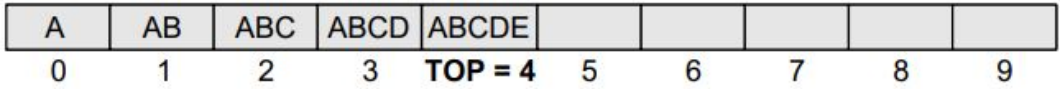


When D calls E, D is pushed on top of the system stack. When the execution of E is complete, the system control will remove D from the stack and continue with its execution.



Array representation of Stacks

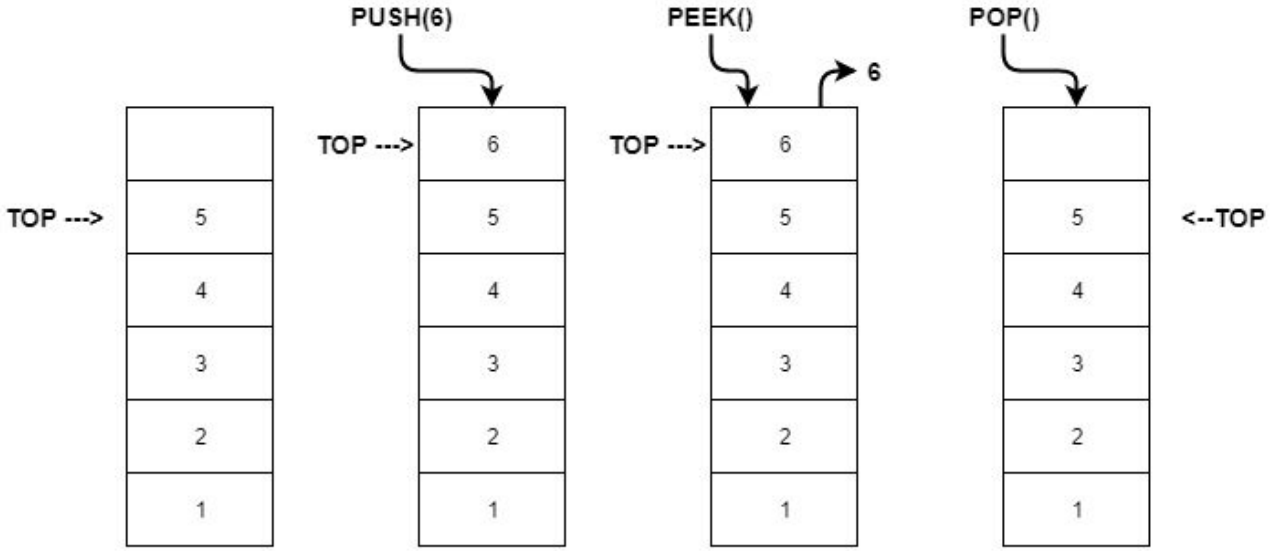
- Stacks can be represented as a linear array or linked list.
- Every stack has a variable called TOP associated with it, which is used to store the address of the topmost element of the stack.
- It is this position where the element will be added to or deleted from.
- There is another variable called MAX, which is used to store the maximum number of elements that the stack can hold.
- If TOP = NULL, then it indicates that the stack is empty and if TOP = MAX–1, then the stack is full.



- The stack in figure shows that TOP = 4, so insertions and deletions will be done at this position. In the above stack, five more elements can still be stored.

Role of TOP pointer in Stack

- TOP pointer allows only one element at a time, be it its insertion or deletion.
- TOP is the only pointer using which insertion or deletion is done.
- To print the topmost element also, TOP pointer is used.



Allowable Operations on a Stack

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Allowable Operations on a Stack

A stack supports three basic operations: **push, pop,** and **peek.**

- **Push:** It refers to inserting an element in the stack, which is inserted at the top of the stack.
- **Pop:** It refers to Deleting an Element from the stack. i.e. the only element at the top of the stack.
- **Peek:** To print the element pointed by the TOP.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Array Implementation of Stack

- int stack[10]; // 10 is a pre-defined capacity of the stack
- int top = -1; // default value for the top is -1, denotes stack is empty.
- int capacity; //max size of the stack. int n;

Stack Initialization:

```
1.  void initStack(int c){
2.          capacity = c;
3.          top=-1;
4.  }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

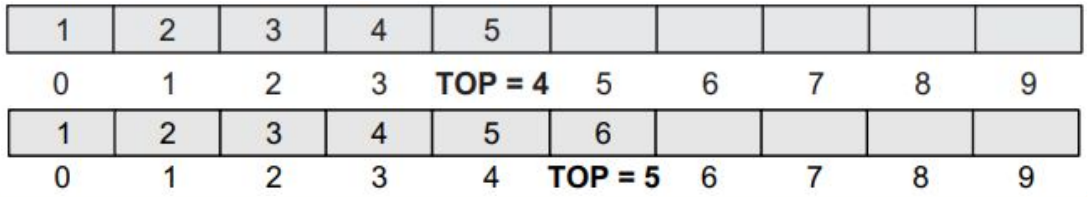
Push Operation

- It is used to insert an element into the stack.
- The new element is added at the topmost position of the stack.
- Before push, check if TOP=MAX–1, (stack is full)
- If stack that is already full, an OVERFLOW message is printed

Code Snippet:

```
1. void push(int st[], int val){
2. if(top == MAX-1){
3. printf("\n STACK OVERFLOW");
4. }
5. else{
6. top++;
7. st[top] = val;
8. } }
```

```
Step 1: IF TOP = MAX-1
            PRINT "OVERFLOW"
            Goto Step 4
        [END OF IF]
Step 2: SET TOP = TOP + 1
Step 3: SET STACK[TOP] = VALUE
Step 4: END
```



Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Pop Operation

- The pop operation is used to delete the topmost element from the stack.
- Before pop(), check if TOP=NULL (if the stack is empty)
- If a stack that is already empty, an UNDERFLOW message is printed.

Code Snippet:

```
1.  int pop(int st[]){
2.  int val;
3.  if(top == -1){
4.  printf("\n STACK UNDERFLOW");
5.  return -1;
6.  }
7.  else{
8.  val = st[top];
9.  top--;
10. return val;
11. } }
```

```
Step 1: IF TOP = NULL
            PRINT "UNDERFLOW"
            Goto Step 4
        [END OF IF]
Step 2: SET VAL = STACK[TOP]
Step 3: SET TOP = TOP - 1
Step 4: END
```

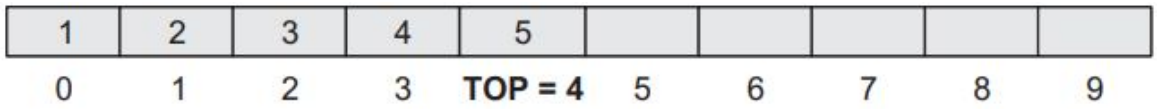
Peek Operation

- Peek is an operation that returns the value of the topmost element of the stack without deleting it from the stack
- However, the Peek operation first checks if the stack is empty, i.e., if TOP = NULL, then an appropriate message is printed, else the value is returned.

Code Snippet:

```
1. int peek(int st[]){
2. if(top == -1){
3. printf("\n STACK IS EMPTY");
4. return -1;
5. }
6. else
7. return (st[top]);
8. }
```

```
Step 1: IF TOP = NULL
            PRINT "STACK IS EMPTY"
            Goto Step 3
Step 2: RETURN STACK[TOP]
Step 3: END
```



Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Array Implementation of Stack

IsEmpty() operation:

Code Snippet:

```
1.  bool isEmpty()
2.  {
3.      if ( top == -1 )
4.          return True
5.      else
6.          return False
7.  }
```

IsFull() operation:

Code Snippet:

```
1.  bool IsFull()
2.  {
3.  if (top == MAX-1)
4.  return True;
5.  else
6.  return False
7.  }
```

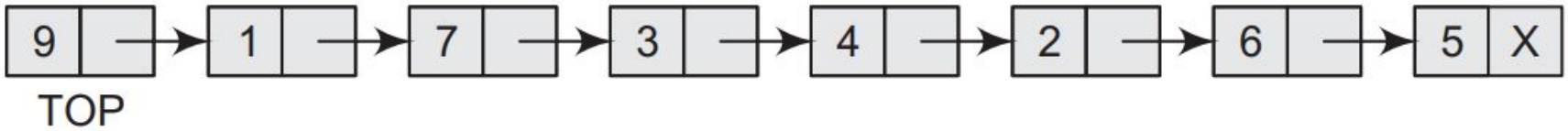

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Menu Driven program in C implementing all the stack operations

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

LINKED REPRESENTATION OF STACKs

- **The drawback** : the array must be declared to have some fixed size
- If the size of stack is not known in advance, linked representation, is used
- In a linked stack, every node has two parts
  one that stores **data**
  another that stores the **address of the next node.**
- The **START pointer** of the linked list is used as **TOP**
- All insertions and deletions are done at the **node pointed by TOP.**
- If **TOP = NULL**, then it indicates that the stack is empty.

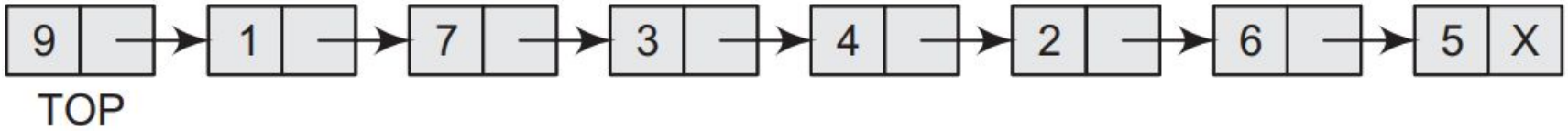


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

LINKED REPRESENTATION OF STACKs

- **Code Snippet:**

```
1. struct stack{
2. int data;
3. struct stack *next;
4. };
5. struct stack *top = NULL;
```



Operations on Linked Stack:

- A linked stack supports all the three stack operations, that is, **push, pop, and peek**

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

OPERATIONS ON A LINKED STACK

- **Push Operation:**

```
1.  struct stack *push(struct stack *top, int val){
2.  struct stack *ptr;
3.  ptr = (struct stack*)malloc(sizeof(struct stack));
4.  ptr -> data = val;
5.  if(top == NULL){
6.   ptr -> next = NULL;
7.   top = ptr;
8.  }
9.  else{
10.  ptr -> next = top;
11.  top = ptr;
12. }
13. return top;
14. }
```

```
Step 1: Allocate memory for the new
        node and name it as NEW_NODE
Step 2: SET NEW_NODE -> DATA = VAL
Step 3: IF TOP = NULL
            SET NEW_NODE -> NEXT = NULL
            SET TOP = NEW_NODE
        ELSE
            SET NEW_NODE -> NEXT = TOP
            SET TOP = NEW_NODE
        [END OF IF]
Step 4: END
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

OPERATIONS ON A LINKED STACK

- **Pop Operation:**

```
1.  struct stack *pop(struct stack *top){
2.  struct stack *ptr;
3.  ptr = top;
4.  if(top == NULL)
5.  printf("\n STACK UNDERFLOW");
6.  else {
7.   top = top -> next;
8.   printf("\n The value being deleted is: %d", ptr -> data);
9.   free(ptr);
10.  }
11.  return top;
12.  }
```

```
Step 1: IF TOP = NULL
            PRINT "UNDERFLOW"
            Goto Step 5
        [END OF IF]
Step 2: SET PTR = TOP
Step 3: SET TOP = TOP -> NEXT
Step 4: FREE PTR
Step 5: END
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

OPERATIONS ON A LINKED STACK

- **Peek Operation:**

```
1. int peek(struct stack *top){
2. if(top==NULL)
3. return -1;
4. else
5. return top ->data;
6. }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

OPERATIONS ON A LINKED STACK

- **Traverse Operation:**

```
1.  struct stack *display(struct stack *top) {
2.  struct stack *ptr;
3.  ptr = top;
4.  if(top == NULL)
5.  printf("\n STACK IS EMPTY");
6.  else {
7.  while(ptr != NULL)  {
8.   printf("\n %d", ptr -> data);
9.   ptr = ptr -> next;
10.   }
11.  }
12.  return top;
13.  }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Write a program to implement a linked stack and perform all the linked stack operations

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

OPERATIONS ON A LINKED STACK

RCOEM
Shri Ramdeobaba College of Engineering and Management, Nagpur

```
1.  struct stack
2.  {
3.  int data;
4.  struct stack *next;
5.  };
6.  struct stack *top = NULL;
7.  struct stack *push(struct stack *, int);
8.  struct stack *display(struct stack *);
9.  struct stack *pop(struct stack *);
10. int peek(struct stack *);
11. int main(int argc, char *argv[]) {
12. int val, option;
13. do
14. {
15.  printf("\n *****MAIN MENU*****");
16.  printf("\n 1. PUSH");
17.  printf("\n 2. POP");
18.  printf("\n 3. PEEK");
19.  printf("\n 4. DISPLAY");
20.  printf("\n 5. EXIT");
21.  printf("\n Enter your option: ");
22.  scanf("%d", &option);
```

```
1.  switch(option) {
2.   case 1:
3.   printf("\n Enter the number to be pushed on stack: ");
4.   scanf("%d", &val);
5.   top = push(top, val);
6.   break;
7.   case 2:
8.   top = pop(top);
9.   break;
10.  case 3:
11.  val = peek(top);
12.  if (val != -1)
13.  printf("\n The value at the top of stack is: %d", val);
14.  else
15.  printf("\n STACK IS EMPTY");
16.  break;
17.  case 4:
18.  top = display(top);
19.  break;
20.   }
21. }while(option != 5);
22. return 0;
23. }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Multiple stacks

- While implementing a stack using an array, the size of the array must be known in advance.
- If the stack is allocated less space, then OVERFLOW
- Solution to OVERFLOW = frequent modification and reallocation of more space for the array
- If allocate a large amount of space for the stack, then sheer wastage of memory.
- Solution to wastage = To have **multiple stacks** or to have **more than one stack** in the same array of sufficient size.
- Fig shows, **an array STACK[n]** is used to represent two stacks, Stack A and Stack B.
- The value of n is such that the combined size of both the stacks will never exceed n.
- an array STACK[n] is used to represent two stacks, Stack A and Stack B.
- The value of n is such that the combined size of both the stacks will never exceed n.
- Stack A = grow from left to right, Stack B = grow from right to left at the same time

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Multiple stacks

- Fig shows, **an array STACK[n]** is used to represent two stacks, Stack A and Stack B.

- Extending this concept to multiple stacks, a stack can also be used to represent n number of stacks in the same array.
- That is, if we have a STACK[n], then each stack I will be allocated an equal amount of space bounded by indices b[i] and e[i]

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Write a C program to implement multiple stacks.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Applications of stacks

- Reversing a array
- Parentheses checker
- Function Call & Recursion
- Expression conversion and evaluation (Polish Notation)
  - Conversion of an infix expression into a postfix expression
  - Conversion of an infix expression into a prefix expression
  - Evaluation of a postfix expression
  - Evaluation of a prefix expression

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Evaluation of Arithmetic Expressions

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Polish Notations

Infix Notation:

- The operator is placed in between the operands.
- Ex: A+B; here, plus operator is placed between the two operands A and B

Postfix Notation:

- The operator is placed after the operands.
- Ex: A+B Infix notation is same as AB+ in postfix notation

Prefix Notation:

- The operator is placed before the operands.
- Ex: A+B Infix notation is same as +AB in prefix notation

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Conversion of an infix to a postfix expression

Note:

1. The order of evaluation of a postfix expression is always from left to right.
2. Operators with same precedence are performed from left-to-right

- Convert the following infix expressions into postfix expressions

1. (A–B) * (C+D)

a. [AB–] * [CD+]

b. AB–CD+*

2. (A + B) / (C + D) – (D * E)

a. [AB+] / [CD+] – [DE*]

b. [AB+CD+/] – [DE*]

c. AB+CD+/DE*–

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Conversion of an infix to a postfix expression

- Convert the following infix expressions into prefix expressions.

1. **(A + B) * C**
   a. (+AB)*C
   b. *+ABC

2. **(A–B) * (C+D)**
   a. [–AB] * [+CD]
   b. *–AB+CD

3. **(A + B) / (C + D) – (D * E)**
   a. [+AB] / [+CD] – [*DE]
   b. [/+AB+CD] – [*DE]
   c. –/+AB+CD*DE

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm to convert an infix to postfix notation


```
Step 1: Add ")" to the end of the infix expression
Step 2: Push "(" on to the stack
Step 3: Repeat until each character in the infix notation is scanned
        IF a "(" is encountered, push it on the stack
        IF an operand (whether a digit or a character) is encountered, add it to the
        postfix expression.
        IF a ")" is encountered, then
          a. Repeatedly pop from stack and add it to the postfix expression until a
             "(" is encountered.
          b. Discard the "(". That is, remove the "(" from stack and do not
             add it to the postfix expression
        IF an operator O is encountered, then
          a. Repeatedly pop from stack and add each operator (popped from the stack) to the
             postfix expression which has the same precedence or a higher precedence than O
          b. Push the operator O to the stack
        [END OF IF]
Step 4: Repeatedly pop from the stack and add it to the postfix expression until the stack is empty
Step 5: EXIT
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Write a program to convert an infix expression into its equivalent postfix notation

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Example

Convert the following infix expression into postfix expression using the algorithm.

A – (B / C + (D % E * F) / G)* H

1) A+B*C-D => ABC*+D+
2) (A+B) * (C+D) => AB+CD+*
3)

Infix Character Scanned	Stack	Postfix Expression
	(	
A	(	A
–	(–	A
(	(– (	A
B	(– (	A B
/	(– (/	A B
C	(– (/	A B C
+	(– (+	A B C /
(	(– (+ (	A B C /
D	(– (+ (	A B C / D
%	(– (+ (%	A B C / D
E	(– (+ (%	A B C / D E
*	(– (+ (% *	A B C / D E
F	(– (+ (% *	A B C / D E F
)	(– (+	A B C / D E F * %
/	(– (+ /	A B C / D E F * %
G	(– (+ /	A B C / D E F * % G
)	(–	A B C / D E F * % G / +
*	(– *	A B C / D E F * % G / +
H	(– *	A B C / D E F * % G / + H
)		A B C / D E F * % G / + H * –

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Evaluation of a Postfix Expression

- Every character of the postfix expression is scanned from left to right.
- If the character encountered is an operand, it is pushed onto the stack.
- However, if an operator is encountered, then the top two values are popped from the stack and the operator is applied on these values. The result is then pushed onto the stack.
- For example:
  Consider the infix notation: 9 – ((3 * 4) + 8) / 4
  Convert it to Postfix notation: 9 3 4 * 8 + 4 / –
  And now evaluate the postfix expression.

Character Scanned	Stack
9	9
3	9, 3
4	9, 3, 4
*	9, 12
8	9, 12, 8
+	9, 20
4	9, 20, 4
/	9, 5
–	4

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm to evaluate Postfix Expression

- For example: evaluate the postfix expression: 9 3 4 * 8 + 4 / –

```
Step 1: Add a ")" at the end of the
        postfix expression
Step 2: Scan every character of the
        postfix expression and repeat
        Steps 3 and 4 until ")"is encountered
Step 3: IF an operand is encountered,
        push it on the stack
        IF an operator O is encountered, then
        a. Pop the top two elements from the
           stack as A and B as A and B
        b. Evaluate B O A, where A is the
           topmost element and B
           is the element below A.
        c. Push the result of evaluation
           on the stack
        [END OF IF]
Step 4: SET RESULT equal to the topmost element
        of the stack
Step 5: EXIT
```

Character Scanned	Stack
9	9
3	9, 3
4	9, 3, 4
*	9, 12
8	9, 12, 8
+	9, 20
4	9, 20, 4
/	9, 5
–	4

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Write a C program to evaluate a postfix expression

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Conversion of an Infix to a Prefix Expression

- There are two algorithms to convert an infix expression into its equivalent prefix expression
- **Algorithm No: 1**

```
Step 1: Scan each character in the infix
        expression. For this, repeat Steps
        2-8 until the end of infix expression
Step 2: Push the operator into the operator stack,
        operand into the operand stack, and
        ignore all the left parentheses until
        a right parenthesis is encountered
Step 3: Pop operand 2 from operand stack
Step 4: Pop operand 1 from operand stack
Step 5: Pop operator from operator stack
Step 6: Concatenate operator and operand 1
Step 7: Concatenate result with operand 2
Step 8: Push result into the operand stack
Step 9: END
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Conversion of an Infix to a Prefix Expression

- There are two algorithms to convert an infix expression into its equivalent prefix expression
- **Algorithm No: 2**

```
Step 1: Reverse the infix string. Note that
        while reversing the string you must
        interchange left and right parentheses.
Step 2: Obtain the postfix expression of the
        infix expression obtained in Step 1.
Step 3: Reverse the postfix expression to get
        the prefix expression
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Ex: Conversion of an Infix to a Prefix Expression

- **For example, given an infix expression**
  (A – B / C) * (A / K – L)
- **Step 1: Reverse the infix string.**
  Note that while reversing the string you must interchange left and right parentheses.
  (L – K / A) * (C / B – A)
- **Step 2: Obtain the corresponding postfix expression of the infix obtained as a result of Step 1.**
  The expression is: (L – K / A) * (C / B – A)
  Therefore, = [L – (K A /)] * [(C B /) – A]
  = [LKA/–] * [CB/A–]
  = L K A / – C B / A – *
- **Step 3: Reverse the postfix expression to get the prefix expression.**
  Therefore, the prefix expression is *** – A / B C – /A K L**

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Write a C program to convert an infix expression to a prefix expression.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Evaluation of a Prefix Expression

- For example, consider the prefix expression
  + – 9 2 7 * 8 / 4 12

Character scanned	Operand stack
12	12
4	12, 4
/	3
8	3, 8
*	24
7	24, 7
2	24, 7, 2
-	24, 5
+	29

```
Step 1: Accept the prefix expression
Step 2: Repeat until all the characters
        in the prefix expression have
        been scanned
        (a) Scan the prefix expression
            from right, one character at a
            time.
        (b) If the scanned character is an
            operand, push it on the
            operand stack.
        (c) If the scanned character is an
            operator, then
              (i) Pop two values from the
                  operand stack
             (ii) Apply the operator on
                  the popped operands
            (iii) Push the result on the
                  operand stack
Step 3: END
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Write a C program to evaluate a prefix expression.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Queue

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Queue ADT

- Allowable operations
- Algorithms and their complexity analysis for **simple queue** and **circular queue**
- Introduction to **double-ended queues** and **priority queues**.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Queue ADT

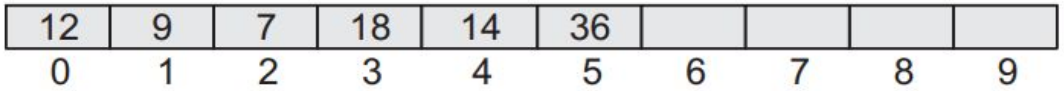
Analogies:

- **People moving on an escalator.**
  The people who got on the escalator first will be the first one to step out of it.
- **People standing outside the ticketing window of a cinema hall.**
  The first person in the line will get the ticket first and thus will be the first one to move out of it.
- **Luggage kept on conveyor belts.**
  The bag which was placed first will be the first to come out at the other end.
- **Cars lined at a toll bridge.**
  The first car to reach the bridge will be the first to leave.
- **People waiting for a bus.**
  The first person standing in the line will be the first one to get into the bus.
- **Moral: element at the first position is served first.**

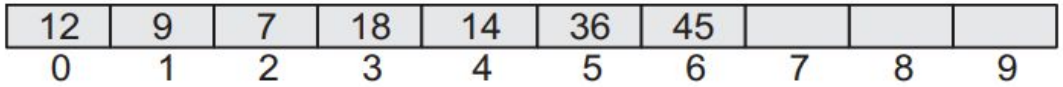
Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Queue ADT

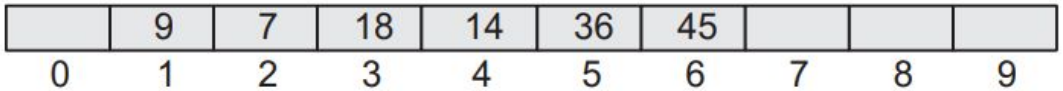
- A queue is a FIFO (First-In, First-Out) data structure
- Here, the element that is inserted first is the first one to be taken out.
- The elements in a queue are added at one end called the **REAR** and removed from the other end called the **FRONT**
- Queues can be implemented by using **arrays** and **linked lists**
- **Fig: a) Queue (Front = 0 and Rear = 5)**



- **Fig b) insertion (Rear = 6)**



- **Fig c) deletion (Front = 1)**



Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Queue ADT

Overflow Condition: REAR = MAX – 1

- If we try to insert an element into a queue that is already full.
- When **REAR = MAX – 1,** where MAX is the size of the queue (because the index starts from 0)

Underflow Condition:

- If we try to delete an element from a queue that is already empty.
- **If FRONT = –1 and REAR = –1**, it means there is no element in the queue.

Define a Queue Using Array: (Global)

- #define MAX 10
- int queue[MAX];
- int front = -1, rear = -1;

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm to insert an element in a queue

Steps:

1. Check for the overflow condition.
2. Check if the queue is empty.
   If yes,then set FRONT and REAR to zero
   (the new value can be stored at the 0th location.)
   Else, (queue already has some values)
   then REAR is incremented so that it points to the next location in the array.
3. The value is stored in the queue
   at the location pointed by REAR.

```
Step 1: IF REAR = MAX-1
            Write OVERFLOW
            Goto step 4
        [END OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = NUM
Step 4: EXIT
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm to insert an element in a queue

Code Snippet:

```
1.  void insert() {
2.  int num;
3.  printf("\n Enter the number to be inserted");
4.  scanf("%d", &num);
5.  if(rear == MAX-1)
    printf("\n OVERFLOW");
6.  else if(front == -1 && rear == -1)
    front = rear = 0;
7.  else
    rear++;
8.  queue[rear] = num;
9.  }
```

```
Step 1: IF REAR = MAX-1
            Write OVERFLOW
            Goto step 4
        [END OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = NUM
Step 4: EXIT
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm to delete an element from the queue

Steps:

1. Check for underflow condition.
   An underflow occurs
   if FRONT = –1 or FRONT > REAR.
2. Else, if queue has some values,
   then FRONT is incremented so that
   it now points to the next value in the queue

```
Step 1: IF FRONT = -1 OR FRONT > REAR
            Write UNDERFLOW
        ELSE
            SET VAL = QUEUE[FRONT]
            SET FRONT = FRONT + 1
        [END OF IF]
Step 2: EXIT
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm to delete an element from the queue

Code Snippet:

```
1.  int delete(){
2.  int val;
3.  if(front == -1 || front>rear){
    printf("\n UNDERFLOW");
    return -1;
    }
4.  else{
    val = queue[front];
    front++;
    if(front > rear)
        front = rear = -1;
        return val;
        }
    }
```

```
Step 1: IF FRONT = -1 OR FRONT > REAR
            Write UNDERFLOW
        ELSE
            SET VAL = QUEUE[FRONT]
            SET FRONT = FRONT + 1
        [END OF IF]
Step 2: EXIT
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Codes for Peek and Display

Peek Code Snippet:

```
1.  int peek() {
2.  if(front==-1 || front>rear) {
3.  printf("\n QUEUE IS EMPTY");
4.  return -1;
5.  }
6.  else {
7.  return queue[front];
8.  }
9.  }
```

Display Code Snippet:

```
1.  void display() {
2.  int i;
3.  printf("\n");
4.  if(front == -1 || front > rear)
5.  printf("\n QUEUE IS EMPTY");
6.  else {
7.  for(i = front;i <= rear;i++)
8.  printf("\t %d", queue[i]);
9.  }
10. }
```

Write a C program to implement a linear queue

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Write a program to implement a linear queue

RCOEM
Shri Ramdeobaba College of Engineering and Management, Nagpur

```
1.  #include <stdio.h>  #include <conio.h>
2.  #define MAX 10
3.  int queue[MAX], int front = -1, rear = -1;
4.  void insert(void);
5.  int delete_element(void);
6.  int peek(void);
7.  void display(void);
8.  int main() {
9.  int option, val;
10. do {
11. printf("\n\n ***** MAIN MENU
    *****");
12. printf("\n 1. Insert an element \n 2. Delete
    an element \n 3. Peek \n 4. Display the
    queue \n 5. EXIT");
13. printf("\n Enter your option : ");
14. scanf("%d", &option);
```

```
1.  switch(option)  {
2.  case 1:       insert(); break;
3.  case 2: val = delete_element();
4.  if (val != -1)
5.  printf("\n The number deleted is : %d", val);
6.  break;
7.  case 3: val = peek();
8.  if (val != -1)
9.  printf("\n The first value in queue is : %d", val);
10. break;
11. case 4: display(); break;
12.  }
13. }while(option != 5);
14. getch();
15. return 0;
16. }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

TYPES OF QUEUES

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Types of Queues

A queue data structure can be classified into the following types:

1. Circular Queue
2. Deque
3. Priority Queue
4. Multiple Queue

Linear Queue:

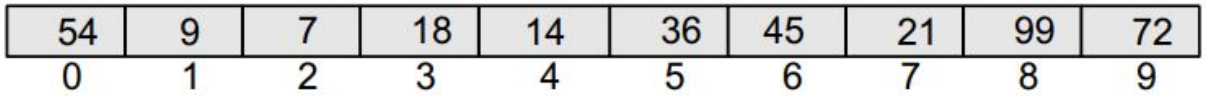
- Insertions can be done only at one end called the REAR
- Deletions are always done from the other end called the FRONT
- Here, Front = 0, Rear = 9. No element can be inserted as Queue is FULL.

54	9	7	18	14	36	45	21	99	72
0	1	2	3	4	5	6	7	8	9

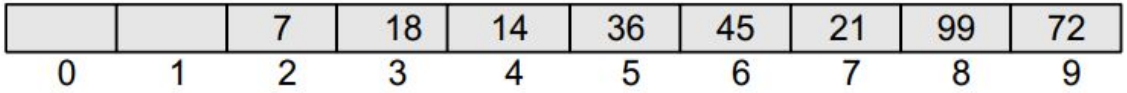
Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Linear Queue

- Here, Front = 0, Rear = 9. No element can be inserted as Queue is FULL.



- Consider, Front = 2 and Rear = 9 (After two successive deletion)



- What is the condition of Queue FULL???
  REAR = MAX - 1, i.e. Condition True.
- But there are two space available for new elements to be inserted.
- **Hence, this is a major drawback of a linear queue.**
- What are the solution?

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Solution to the drawback of Linear Queue

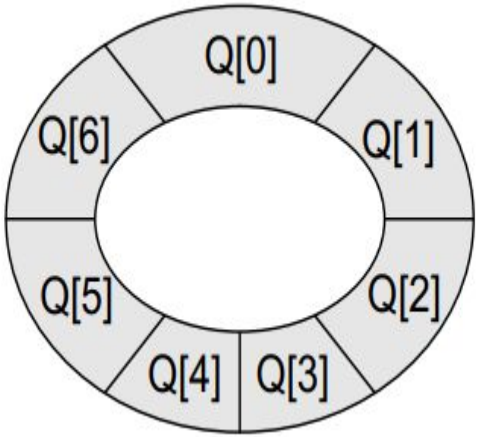
		7	18	14	36	45	21	99	72
0	1	2	3	4	5	6	7	8	9

Solution 1:

- Shift the elements to the left so that the vacant space can be occupied and utilized efficiently
- It can be time-consuming, if queue is very large.

Solution 2:

- Use a circular queue
- In the circular queue, the first index comes right after the last index.
- The circular queue will be full only
  when front = 0 and rear = Max – 1
- Fig on the right is an example of Circular queue.



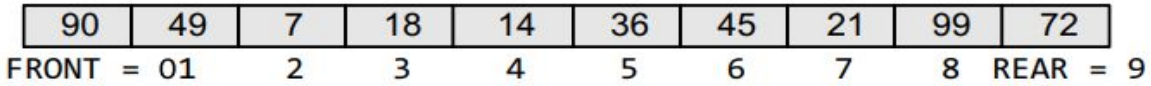
Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Circular Queue

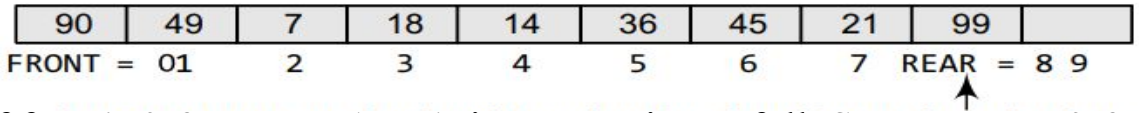
- A circular queue is implemented in the same manner as a linear queue is implemented.
- The only difference will be in the code that performs insertion and deletion operations.

For insertion, we now have to check for the following three conditions:

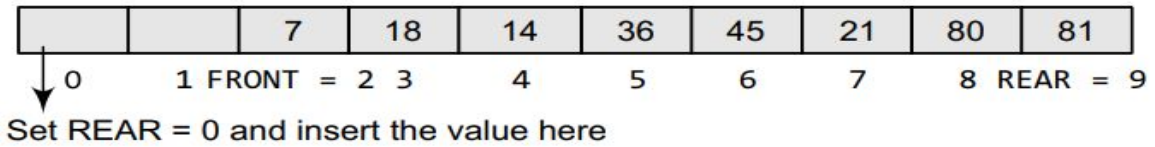
- If front = 0 & rear = MAX – 1, the circular queue is full.



- If rear != MAX – 1, then rear will be incremented and the value will be inserted.



- If front!=0 & rear=MAX–1, i.e. queue is not full. So, set rear = 0 & insert the new element there



Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm to insert an element in a circular queue

Steps:

1. Check for the overflow condition.
2. We make two checks.
   First to see if the queue is empty
   And Second to see if the REAR end has already reached the maximum capacity while there are certain free locations before the FRONT end.
3. The value is stored in the queue at the location pointed by REAR

```
Step 1: IF FRONT = 0 and Rear = MAX - 1
            Write "OVERFLOW”
            Goto step 4
        [End OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE IF REAR = MAX - 1 and FRONT != 0
            SET REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = VAL
Step 4: EXIT
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm to insert an element in a circular queue

Code Snippet:

```
1.  void insert() {
2.  int num;
3.  printf("\n Enter the number to be inserted in the
    queue : ");
4.  scanf("%d", &num);
5.  if(front==0 && rear==MAX–1)
6.   printf("\n OVERFLOW");
7.  else if(front==–1 && rear==–1) {
8.   front=rear=0;
9.   queue[rear]=num;
10. }
11. else if(rear==MAX–1 && front!=0) {
12. rear=0;
13. queue[rear]=num;
14. }
15. else {
16. rear++;
17. queue[rear]=num;
18. } }
```

```
Step 1: IF FRONT = 0 and Rear = MAX - 1
            Write "OVERFLOW”
            Goto step 4
        [End OF IF]
Step 2: IF FRONT = -1 and REAR = -1
            SET FRONT = REAR = 0
        ELSE IF REAR = MAX - 1 and FRONT != 0
            SET REAR = 0
        ELSE
            SET REAR = REAR + 1
        [END OF IF]
Step 3: SET QUEUE[REAR] = VAL
Step 4: EXIT
```

Algorithm to delete an element in a circular queue

Steps:

1. Check for the underflow condition.
2. The value of the queue at the location pointed by FRONT is stored in VAL.
3. We make two checks.
   First see, if the queue has become empty after deletion and
   second see, if FRONT has reached the maximum capacity of the queue.
   The value of FRONT is then updated based on the outcome of these checks

```
Step 1: IF FRONT = -1
            Write "UNDERFLOW"
            Goto Step 4
        [END of IF]
Step 2: SET VAL = QUEUE[FRONT]
Step 3: IF FRONT = REAR
            SET FRONT = REAR = -1
        ELSE
            IF FRONT = MAX -1
                  SET FRONT = 0
            ELSE
                  SET FRONT = FRONT + 1
            [END of IF]
        [END OF IF]
Step 4: EXIT
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm to delete an element in a circular queue

Code Snippet:

```
1.  int delete_element() {
2.  int val;
3.  if(front==–1 && rear==–1)  {
4.   printf("\n UNDERFLOW");
5.   return –1;
6.   }
7.  val = queue[front];
8.  if(front==rear)
9.   front=rear=–1;
10. else {
11.  if(front==MAX–1)
12. front=0;
13.  else
14.  front++;
15. }
16. return val;
17. }
```

```
Step 1: IF FRONT = -1
            Write "UNDERFLOW"
            Goto Step 4
        [END of IF]
Step 2: SET VAL = QUEUE[FRONT]
Step 3: IF FRONT = REAR
            SET FRONT = REAR = -1
        ELSE
            IF FRONT = MAX -1
                  SET FRONT = 0
            ELSE
                  SET FRONT = FRONT + 1
            [END of IF]
        [END OF IF]
Step 4: EXIT
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm to peek and display in a circular queue

Peek Code Snippet:

```
1.  int peek(){
2.  if(front==–1 && rear==–1) {
3.   printf("\n QUEUE IS EMPTY");
4.   return –1;
5.  }
6.  else{
7.   return queue[front];
8.  }
9.  }
```

Display Code Snippet:

```
1.  void display() {
2.  int i;
3.  printf("\n");
4.  if (front ==–1 && rear= =–1)
5.   printf ("\n QUEUE IS EMPTY");
6.  else{
7.   if(front<rear) {
8.   for(i=front;i<=rear;i++)
9.   printf("\t %d", queue[i]);
10.  }
11.  else {
12.  for(i=front;i<MAX;i++)
13.  printf("\t %d", queue[i]);
14. for(i=0;i<=rear;i++)
15.  printf("\t %d", queue[i]);
16.  }}}
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Write a C program to implement a circular queue

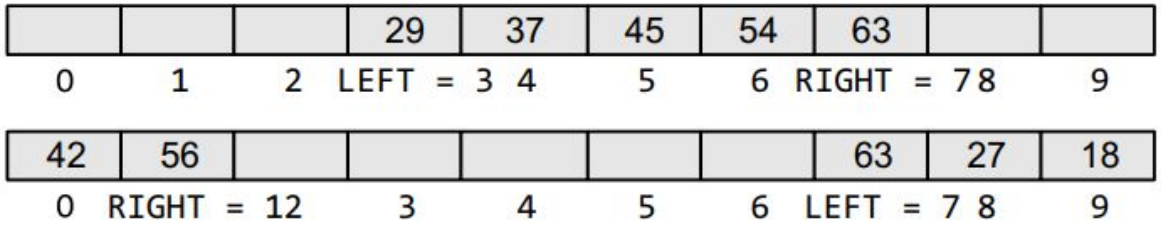
Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Dequeue

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Deque

- A deque is a list in which the elements **can be inserted or deleted at either end**.
- Also called as **Double Ended Queue**
- However, **no element can be added and deleted from the middle**
- It is also known as a **head-tail linked lis**t because elements can be added to or removed from either the front (head) or the back (tail) end.
- A deque is implemented using either **a circular array** or a circular doubly linked list
- Here, two pointers are maintained, **LEFT** and **RIGHT**, which point to either end of the deque
- The elements in a deque extend from the **LEFT** end to the **RIGHT** end and **since it is circular**, **Dequeue[N–1]** is followed by **Dequeue[0]**



Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

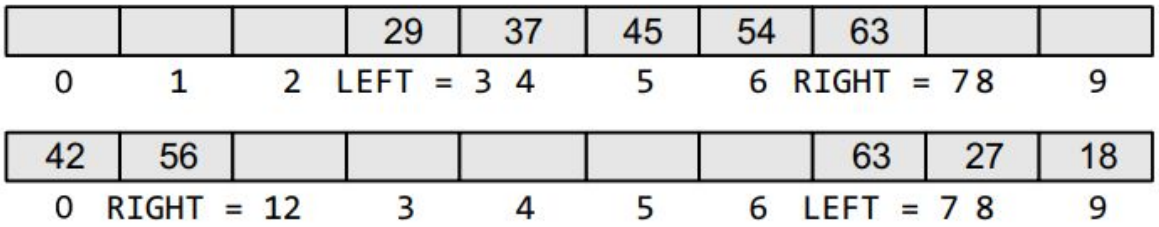
Overflow and Underflow conditions of Deque

A) Overflow condition:

- If the deque is completely full, that means
  Left = 0 && Right = MAX-1

B) Underflow Condition:

- If the deque is completely empty, that means
  Left = -1



Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

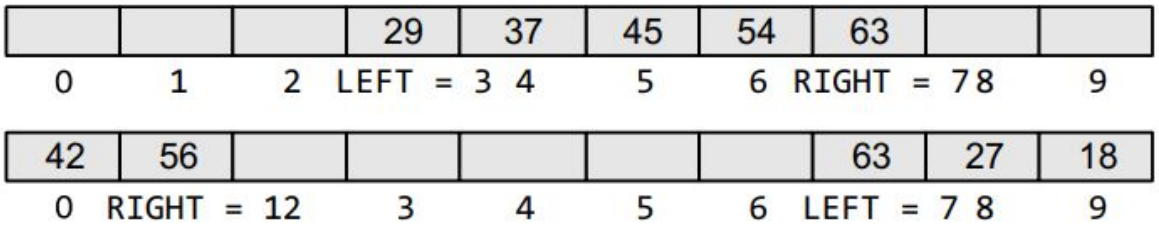
Two variant of Deque

A) Input restricted deque:

- **insertions** can be done only at **one of the ends** (Right)
- deletions can be done from both ends. (Left & Right)

B) Output restricted deque:

- **deletions** can be done only at **one of the ends** (Left)
- insertions can be done on both ends.(Left & Right)



Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Deque Operations

1. **Input Deque: Input Restricted Deque**
   1.Insert at right 2.Delete from left 3.Delete from right
2. **Output Deque: Output Restricted Deque**
   1.Insert at right 2.Insert at left 3.Delete from left
3. **Insert Left:** Insert Element using Left
4. **Insert Right:** Insert Element using Right
5. **Delete Left:** Delete Element using Left
6. **Delete Right:** Delete Element using Right
7. **Display:** Traverse the deque

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm for working of Deque

Steps:

1. Start
2. Ask for choice: Enter 1.Input restricted deque 2.Output restricted deque
3. If choice = 1, ask for choice
   Enter : 1.Insert at right 2.Delete from left 3.Delete from right
   4.Display 5.Quit
4. Else, if choice = 2, ask for choice
   Enter: 1.Insert at right 2.Insert at left 3.Delete from left
   4.Display 5.Quit
5. End of If
6. Stop

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm for Insert Left

1. Take the value to be inserted (val)
2. Check for Overflow condition
   If true, Print "Overflow"
   (left == 0 && right == MAX–1) or
   (left == right+1)
3. Check whether the Deque is initially empty
   If true, Set Left=0, Right=0
4. Check if the deque has space but left=0
   If true, Set, Left=Max-1
5. Else, set, Left = Left - 1
6. deque[left] = val;

If the deque has space from both sides left=1, right=4

	20	30	40	
0	1	2	**Right=3**	4

Overflow: When left == 0 && right == MAX–1

10	20	30	40	50
left=0	1	2	3	**right=4**

Overflow: When left == right+1

10	20	30	40	50
0	**right=1**	**left=2**	3	4

Initially Created & is **Empty**: When left = right= -1

0	1	2	3	4

If the deque has space but left=0

10	20	30		
Left=0	1	**Right=2**	3	4

Code for Insert Left


```
1.  void insert_left() {
2.  int val;
3.  printf("\n Enter the value to be added:");
4.  scanf("%d", &val);
5.  if ((left == 0 && right == MAX–1) ||
    (left == right+1)) {
6.   printf("\n Overflow");
7.   return;
8.  }
9.  if (left == –1) {
    // If queue is initially empty
10. left = 0;
11. right = 0;
12. }
```

```
1.  else {
2.  if(left == 0)
3.   left=MAX–1;
4.   else
5.   left=left–1;
6.  }
7.  deque[left] = val;
8.  }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm for Insert Right

1. Take the value to be inserted (val)
2. Check for Overflow condition
   If true, Print "Overflow"
   (left == 0 && right == MAX–1) or
   (left == right+1)
3. Check whether the Deque is initially empty
   If true, Set Left=0, Right=0
4. Check if the deque has space
   but right = MAX-1
   If true, Set, Right = 0;
5. Else, set, right = right+1;
6. deque[left] = val;

If the deque has space from both sides left=1, right=4

	20	30	40	
0	1	2	**Right=3**	4

Overflow: When left == 0 && right == MAX–1

10	20	30	40	50
left=0	1	2	3	**right=4**

Overflow: When left == right+1

10	20	30	40	50
0	**right=1**	**left=2**	3	4

Initially Created & is **Empty**: When left = right= -1

0	1	2	3	4

If the deque has space but right=MAX-1

		30	40	50
0	1	**Left=2**	3	**Right=4**

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Code for Insert Right


```
1.  void insert_right(){
2.  int val;
3.  printf("\n Enter the value to be added:");
4.  scanf("%d", &val);
5.  if((left == 0 && right == MAX–1) ||
    (left == right+1)) {
6.   printf("\n OVERFLOW");
7.   return;
8.  }
9.  if (left == –1) {
    /* if queue is initially empty */
10. left = 0;
11. right = 0;
12. }
```

```
1.  else {
2.  if(right == MAX–1) /*right is at last
    position of queue */
3.  right = 0;
4.  else
5.   right = right+1;
6.  }
7.  deque[right] = val ;
8.  }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm for Delete Left

1. Check for Underflow condition
   left == -1 , If true, Print “Underflow”
2. Element to be deleted is queue[left]
3. If (left=right), means it has only one element
   If true, Set Left = Right = -1
   Else,
   if (left = Max-1), Set left = 0;
   Else,
   left = left+1;

Underflow: left = = -1, right= -1

0	1	2	3	4

After 3 insertions, queue has elements (consider insert right)

10	**20**	**30**		
left=0	1	**right=2**	3	4

Now after two deletions, queue has only one element

		30		
0	1	**L=R=2**	3	4

If the deque has space from both sides left=1, right=4

	20	**30**	**40**	
0	**left=1**	2	**Right=3**	4

If the deque has space but left=MAX-1

		30	**40**	**50**
0	1	**Right=2**	3	**Left=4**

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Code for Delete Left


```
1.  void delete_left() {
2.  if (left == –1) {
3.   printf("\n UNDERFLOW");
4.   return ;
5.  }
6.  printf("\n The deleted element is : %d",
    deque[left]);
7.  if(left == right){
8.   left = –1;
9.   right = –1;
10. }
11.
```

```
1.  else
2.  {
3.   if(left == MAX–1)
4.  left = 0;
5.   else
6.   left = left+1;
7.  }
8.  }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Algorithm for Delete Right

1. Check for Underflow condition
   left == -1 , If true, Print “Underflow”
2. Element to be deleted is queue[right]
3. If (left=right), means it has only one element
   If true, Set Left = Right = -1
   Else,
   if (right == 0), Set right=MAX–1;
   Else,
   right=right–1;

Underflow: left = = -1, right= -1

0	1	2	3	4

After 4 insertions, queue has elements (consider insert left)

10		**40**	**30**	**20**
right=0	1	**left=2**	3	4

If queue has only one element

		30		
0	1	**L=R=2**	3	4

If the deque has space from both sides left=1, right=4

	20	**30**	**40**	
0	**left=1**	2	**Right=3**	4

If the deque has space but right == 0

10				**50**
right=0	1	2	3	left=4

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Code for Delete Right


```
1.  void delete_right(){
2.  if (left == –1){
3.   printf("\n UNDERFLOW");
4.   return ;
5.  }
6.  printf("\n The element deleted is : %d",
    deque[right]);
7.  if(left == right){
    /*queue has only one element*/
8.   left = –1;
9.   right = –1;
10. }
```

```
1.  else{
2.  if(right == 0)
3.   right=MAX–1;
4.   else
5.   right=right–1;
6.  }
7.  }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Code for display


```
1.  void display() {
2.  int front = left, rear = right;
3.  if(front == –1) {
4.   printf("\n QUEUE IS EMPTY");
5.   return;
6.  }
7.  printf("\n The elements of the queue are : ");
8.  if(front <= rear ){
9.  while(front <= rear) {
10.  printf("%d",deque[front]);
11.  front++;
12.   }
13.  }
```

```
1.  else{
2.  while(front <= MAX–1) {
3.   printf("%d", deque[front]);
4.  front++;
5.   }
6.  front = 0;
7.  while(front <= rear) {
8.   printf("%d",deque[front]);
9.  front++;
10.   }
11.  }
12.  printf("\n");
13.  }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Code for Input Restricted Deque


```
1.  void input_deque(){
2.  int option;
3.  do{
4.  printf("\n INPUT RESTRICTED DEQUE");
5.   printf("\n 1.Insert at right");
6.   printf("\n 2.Delete from left");
7.   printf("\n 3.Delete from right");
8.   printf("\n 4.Display");
9.   printf("\n 5.Quit");
10.  printf("\n Enter your option : ");
11.  scanf("%d",&option);
```

```
1.  switch(option) {
2.   case 1:
3.   insert_right(); break;
4.   case 2:
5.   delete_left(); break;
6.   case 3:
7.   delete_right();break;
8.   case 4:
9.   display();break;
10.   }
11.  }while(option!=5);
12.  }
```

Code for Output Restricted Deque


```
1.  void input_deque(){
2.  int option;
3.  do{
4.  printf("OUTPUT RESTRICTED DEQUE");
5.   printf("\n 1.Insert at right");
6.   printf("\n 2.Insert at left");
7.   printf("\n 3.Delete from left");
8.   printf("\n 4.Display");
9.   printf("\n 5.Quit");
10.  printf("\n Enter your option : ");
11.  scanf("%d",&option);
```

```
1.  switch(option) {
2.   case 1:
3.   insert_right(); break;
4.   case 2:
5.   insert_left(); break;
6.   case 3:
7.   delete_left(); break;
8.   case 4:
9.   display();  break;
10.   }
11.  }while(option!=5);
12.  }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Write a C program to implement input and output restricted deques.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Example: Deque

Let DEQ[1:6] be a deque implemented as a circular array.

1. Initial DEQ
   LEFT: 3 RIGHT: 5
2. Insert X at the left end and Y at the right end DEQ:
   LEFT: 2 RIGHT: 6
3. Delete twice from the right end DEQ:
   LEFT: 2 RIGHT: 4
4. Insert G, Q and M at the left end DEQ:
   LEFT: 5 RIGHT: 4
5. Insert J at the right end.
   As deque is full (LEFT=RIGHT+1) No insertion
6. Delete twice from the left end DEQ:
   LEFT: 1 RIGHT: 4

DEQ:

[1]	[2]	[3]	[4]	[5]	[6]
		R	T	S	

[1]	[2]	[3]	[4]	[5]	[6]
	X	R	T	S	Y

[1]	[2]	[3]	[4]	[5]	[6]
	X	R	T		

[1]	[2]	[3]	[4]	[5]	[6]
G	X	R	T	M	Q

[1]	[2]	[3]	[4]	[5]	[6]
G	X	R	T	M	Q

[1]	[2]	[3]	[4]	[5]	[6]
G	X	R	T		

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Priority Queues

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

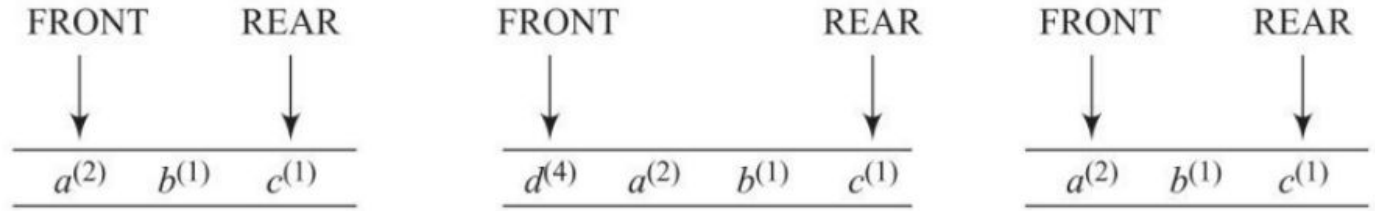
Priority Queues

- A priority queue is a data structure in which **each element is assigned a priority**.
- The priority of the element will be used to **determine the order** in which the elements will be processed.
- The general rules of processing the elements of a priority queue are
  - An element with **higher priority is processed before** an element with a lower priority.
  - Two elements with the **same priority** are processed on a **first-come-first-served (FCFS) basis.**
- It is a modified queue, where when an element has to be removed from the queue, the one with the highest-priority is retrieved first
- Priority queues are widely used in operating systems to **execute the highest priority process first**
- The priority of the process may be set based on the CPU time it requires to get executed completely

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Example of Priority Queues

- **Fig (a):** Let P be a priority queue with three elements a, b, c whose priority factors are 2, 1, 1 respectively. Here, larger the number, higher is the priority accorded to that element.
- **Fig (b):** When a new element d with higher priority viz., 4 is inserted, d joins at the head of the queue superseding the remaining elements
- When elements in the queue have the **same priority**, then the priority queue behaves as an ordinary queue following the principle of **FIFO** amongst such elements.
- But, there are **two ways** to implement Priority Queues.



(a) Initial priority queue (b) Insert $d^{(4)}$ (c) Delete

$x^{(y)}$: x is the element with priority y.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Two ways: Implementation of Priority Queues

- **Using Multiple Queues with different Priorities (Use an unsorted list)**
  Where the insertions are always done at the end of the list.

  OR

- **Using single queue with sorted list according to the priority**
  To store the elements so that when an element has to be taken out, the queue will not have to be searched for the element with the highest priority

- Every time when an element has to be removed from the list, the element with the highest priority will be searched and removed.
- Practically, both these techniques are inefficient and usually a blend of these two approaches is adopted.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Example : Priority Queue

- Let **JOB** be a queue of jobs to be undertaken at a factory shop floor for service by a machine.
- Let high (2), medium (1) and low (0) be the priorities accorded to jobs.
- Let Ji (k) indicate a job Ji to be undertaken with priority k.
- The implementations of a priority queue to keep track of the jobs, using the two methods of implementation discussed above, are illustrated for a sample set of job arrivals (insertions) and job service completion (deletion).
- Opening JOB queue: J1(1) J2 (1) J3 (0)
- Operations on the JOB queue in the chronological order :

  1. J4 (2) arrives 2. J5 (2) arrives
  3. Execute job 4. Execute job 5. Execute job

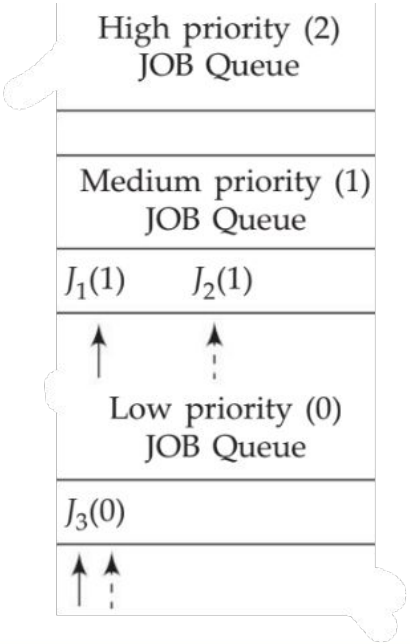
Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

PQueue: Multiple queues vs Single Queue

RCOEM
Shri Ramdeobaba College of Engineering and Management, Nagpur

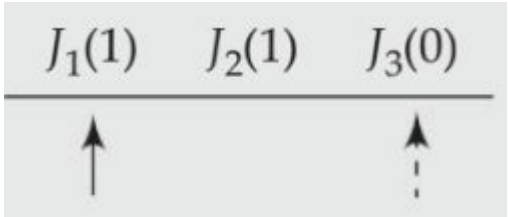
Opening JOB queue: $J_1(1)$ $J_2(1)$ $J_3(0)$

- **Using Multiple Queues**



Opening JOB queue: $J_1(1)$ $J_2(1)$ $J_3(0)$

- **Using Single Queue**

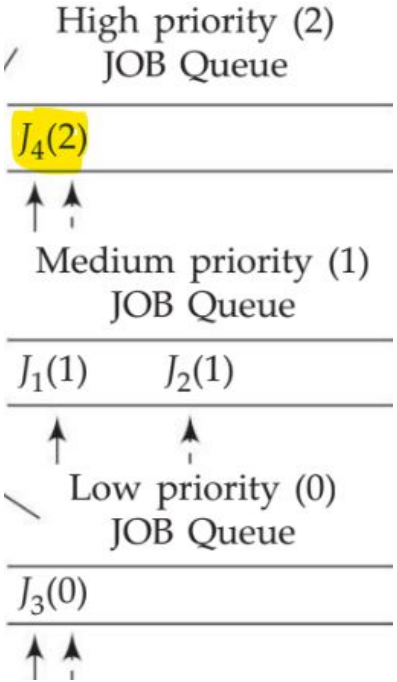


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

PQueue: Multiple queues vs Single Queue

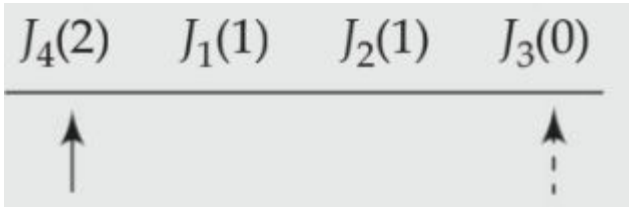
When $J_4(2)$ arrives:

- **Using Multiple Queues: Insert $J_4(2)$**



When $J_4(2)$ arrives:

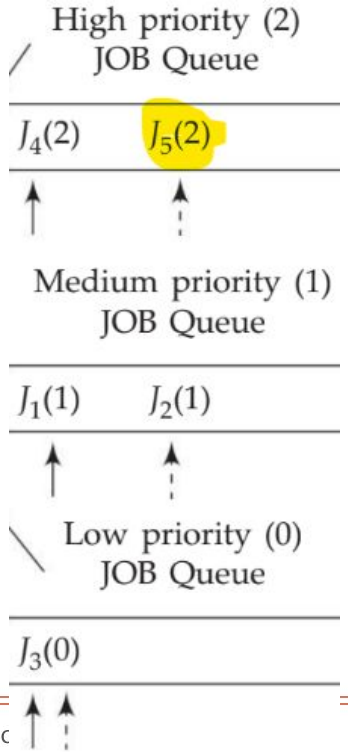
- **Using Single Queue: Insert $J_4(2)$**



PQueue: Multiple queues vs Single Queue

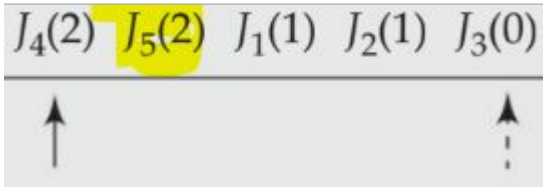
When J_5 (2) arrives:

- **Using Multiple Queues: Insert J_5 (2)**



When $J_5(2)$ arrives:

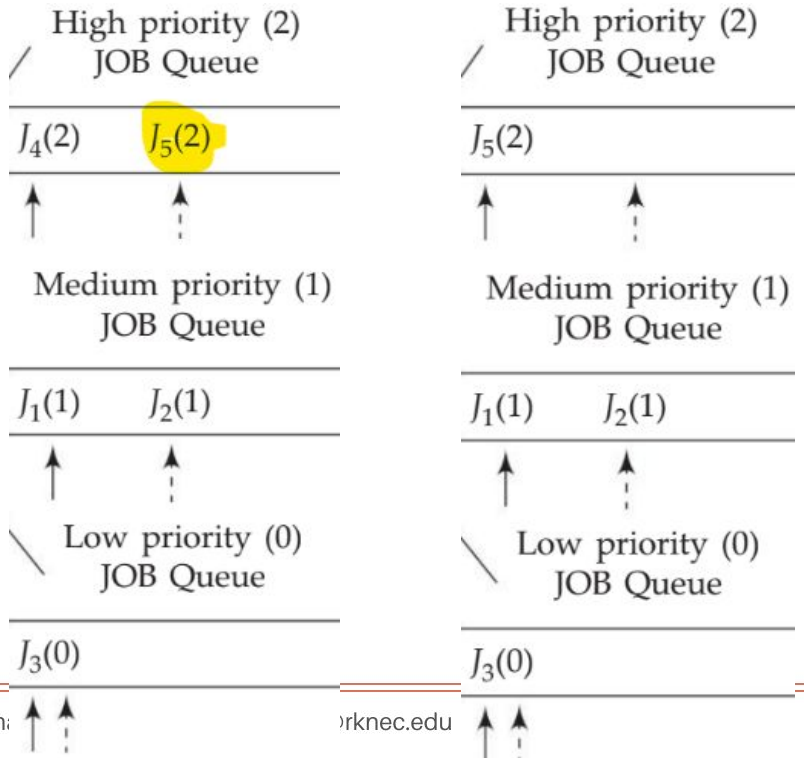
- **Using Single Queue: Insert $J_5(2)$**



PQueue: Multiple queues vs Single Queue

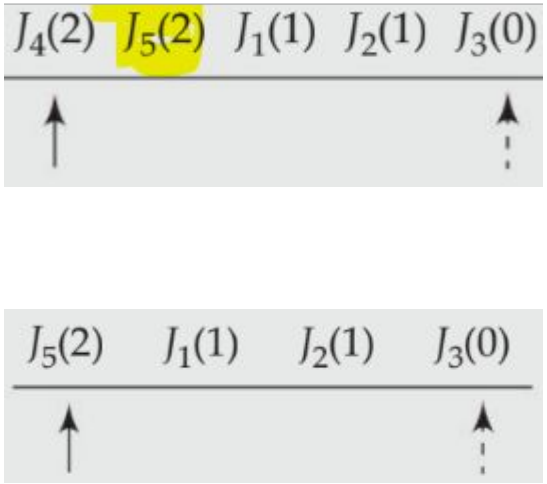
Execute Job:

- **Using Multiple Queues: $J_4(2)$ is deleted**



Execute Job:

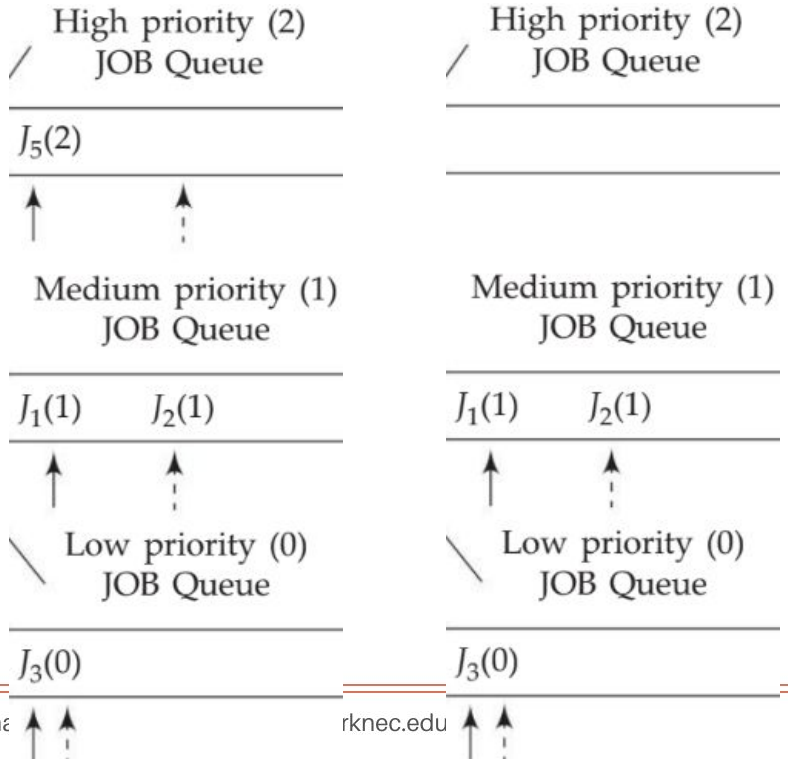
- **Using Single Queue: $J_4(2)$ is deleted**



PQueue: Multiple queues vs Single Queue

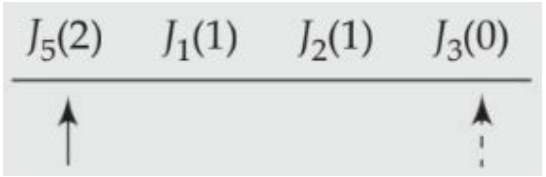
Execute Job:

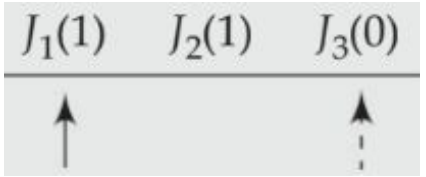
- **Using Multiple Queues: $J_5(2)$ is deleted**



Execute Job:

- **Using Single Queue: $J_5(2)$ is deleted**



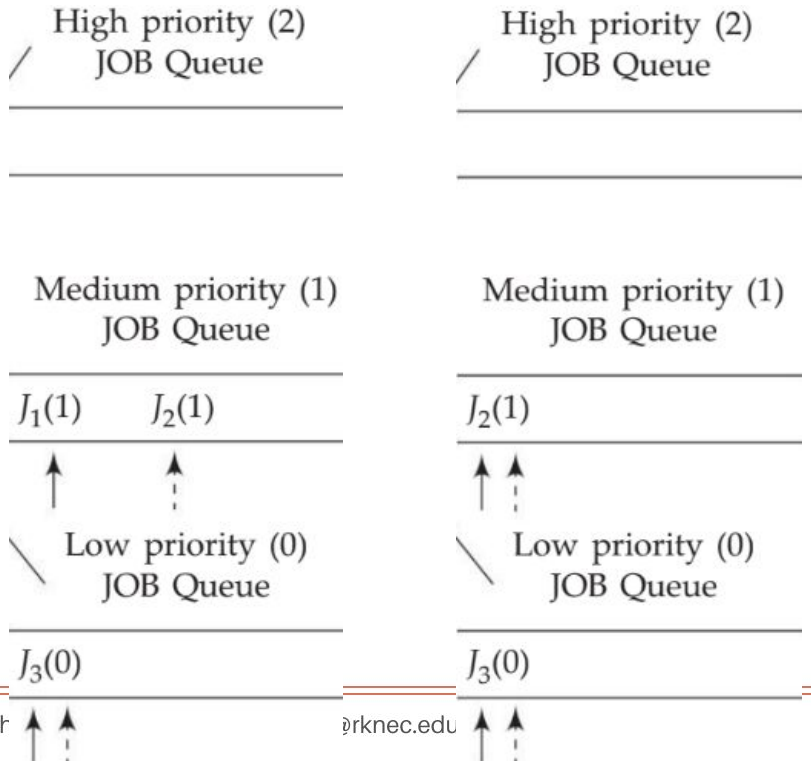


PQueue: Multiple queues vs Single Queue

RCOEM
Shri Ramdeobaba College of Engineering and Management, Nagpur

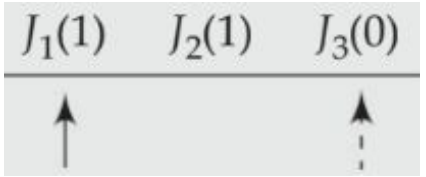
Execute Job:

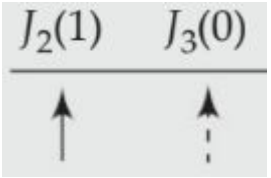
- **Using Multiple Queues: J_1 (1) is deleted**



Execute Job:

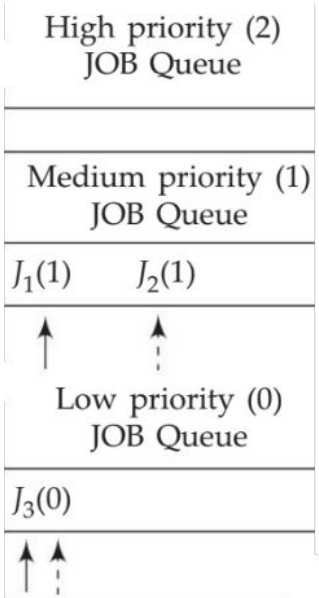
- **Using Single Queue: J_1 (1) is deleted**

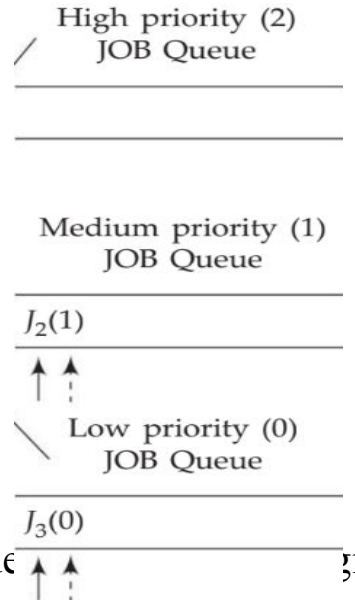


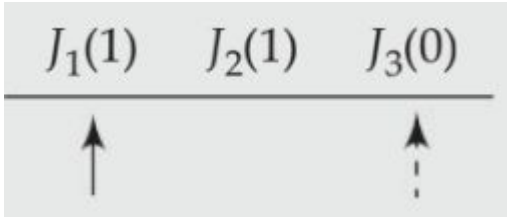


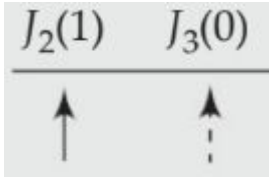
Example : Priority Queue

- Opening JOB queue: J1(1) J2 (1) J3 (0)









- JOB que gical order :

1. J4 (2) arrives 2. J5 (2) arrives 3. Execute job 4. Execute job 5. Execute job

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Compare two ways to design PQueue

RCOEM
Shri Ramdeobaba College of Engineering and Management, Nagpur

Multiple Queues:

- Multiple queues,
  one for each priority.
- **Insertion:** Element e(k) gets inserted in k-priority queue in FIFO mode
- **Deletion:** Element e(k) gets deleted only if all the queues with higher priorities are empty.
- **Time Complexity:**
  For Insertion = O(1)
  For Deletion = Very High
  As it checks all the queues and then decides
- **Space Complexity:** Very High
  As Multiple Queues

Single Queue:

- Single queues, having elements with sorted priorities in descending order.
- **Insertion:** Element e(k) gets inserted according to k-priority
- **Deletion:** Topmost priority element gets deleted.
- **Time Complexity:**
  For Insertion = O(n log n)
  For Deletion = O(1)
  As it deletes the topmost element only.
- **Space Complexity:** Less
  As single queue

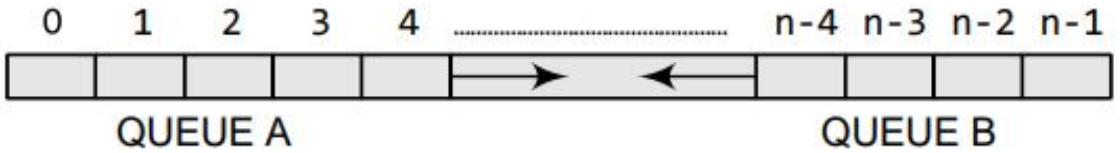
Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Multiple Queues

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Multiple Queues

- When we implement a queue using an array, the size of the array must be known in advance.
- If underflow = sheer waste of memory
- If overflow = modification of code and reallocation of memory
- So, what is the better solution? Answer is **Multiple Queue** or to have **more than one queue** in the **same array** of sufficient size.
- Figure shows a queue[n] which is used to represent two queues, Queue A & Queue B.
- The value of **n** is such that the **combined size of both the queues will never exceed *n***

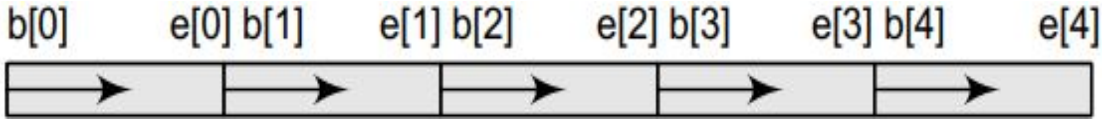


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Multiple Queues

- Figure shows a queue[n] which is used to represent two queues, Queue A & Queue B.
- **Note:** Queue A will grow from left to right & queue B will grow from right to left at the same time
- Extending the concept to multiple queues, a queue can also be used to represent n number of queues in the same array.

- That is, if we have a QUEUE[n], then each queue i will be allocated an equal amount of space bounded by indices b[i] and e[i].



- **Initialize a Multiple Queue:**

```
1. #define MAX 10
2. int QUEUE[MAX];
3. int rearA= –1,frontA= –1, rearB= MAX, frontB = MAX;
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

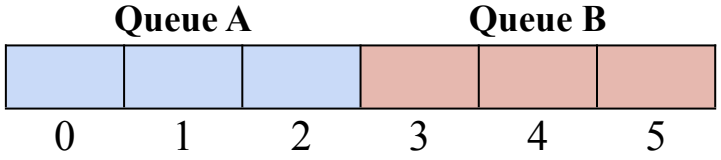
Insert in Multiple Queues: for Queue A

1. If Multiple Queue is Initialized
   FrontA = RearA = -1 & FrontB = RearB = MAX

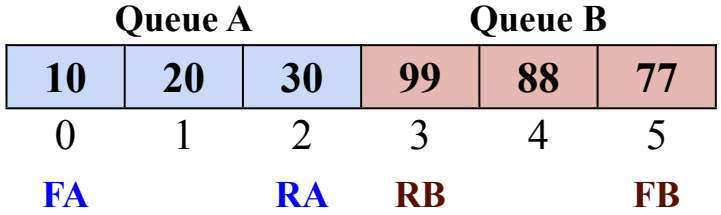
2. Check for Overflow Condition before Inserting
   Check if (rearA = = rearB –1)
   If yes, then printf("\n OVERFLOW");

3. When val (10) is inserted in Queue A
   a) if (FA=RA=-1)
   Set FA=RA= 0 & Set queue[RA] = val
   b) else
   Set RA= RA+1 & Set queue[RA] = val

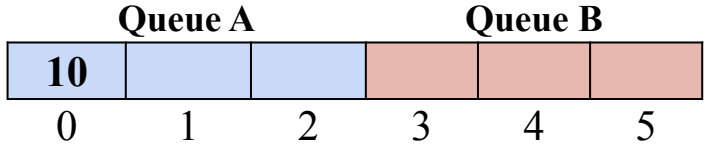
1)



2)

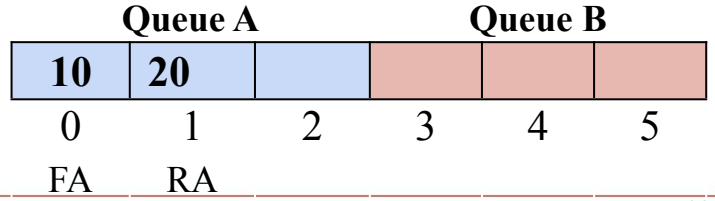


3a)



FA = RA=

3b)



Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Code for Insert & Delete in Multiple Queues: for Queue A


```
1.  void insertA(int val){
2.  if(rearA==rearB –1)
3.   printf("\n OVERFLOW");
4.  else{
5.   if(rearA ==–1 && frontA == –1){ rearA =
    frontA = 0;
6.   QUEUE[rearA] = val;
7.   }
8.   else
9.   QUEUE[++rearA] = val;
10.  } }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Delete in Multiple Queues: for Queue A

1. Check for Underflow Condition before Deleting

   a) if (frontA = –1)

      If yes, then printf("\n UNDERFLOW");

   b) else, val = Queue[FA]; //val=10

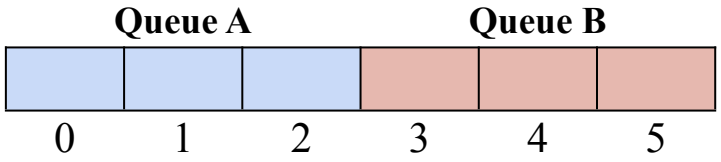
   Set FA = FA + 1 //FA=1

   else,

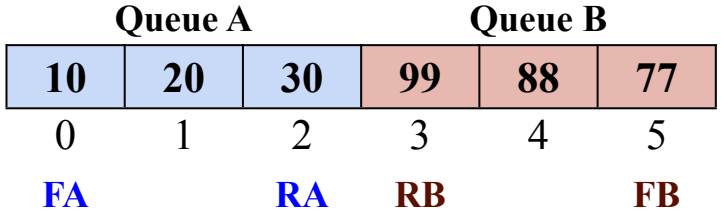
      i) Check if FA > RA

      Set FA = RA = -1

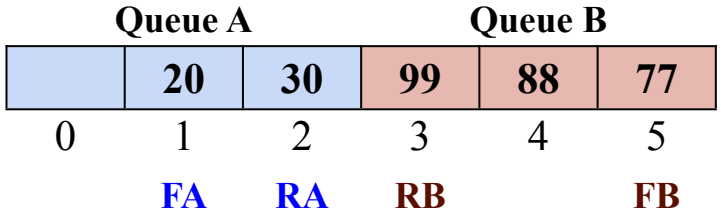
1a)



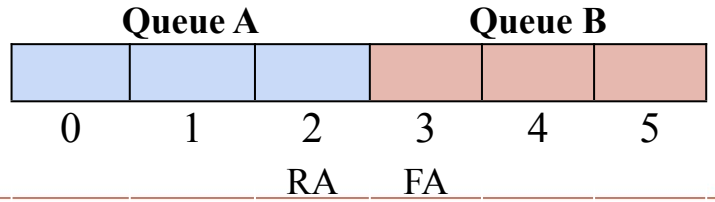
1b)



1b)



1bi)



Code for Insert & Delete in Multiple Queues: for Queue A


```
1.  void insertA(int val){
2.  if(rearA==rearB –1)
3.   printf("\n OVERFLOW");
4.  else{
5.   if(rearA ==–1 && frontA == –1){ rearA =
     frontA = 0;
6.   QUEUE[rearA] = val;
7.   }
8.   else
9.   QUEUE[++rearA] = val;
10.  } }
```

```
1.  int deleteA(){
2.  int val;
3.  if(frontA==–1) {
4.   printf("\n UNDERFLOW");
5.   return –1;
6.  }
7.  else{
8.   val = QUEUE[frontA];
9.   frontA++;
10.  if (frontA>rearA)
11.  frontA=rearA=–1
12.  return val;
13.  } }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Code for Insert & Delete in Multiple Queues: for Queue B


```
1.  void insertB(int val)
2.  {
3.  if(rearA==rearB–1)
4.   printf("\n OVERFLOW");
5.  else
6.  {
7.   if(rearB == MAX && frontB == MAX)
8.   { rearB = frontB = MAX–1;
9.   QUEUE[rearB] = val;
10.  }
11.  else
12.  QUEUE[—rearB] = val;
13. } }
```

```
1.  int deleteB(){
2.  int val;
3.  if(frontB==MAX) {
4.   printf("\n UNDERFLOW");
5.   return –1;
6.  }
7.  else{
8.   val = QUEUE[frontB];
9.   frontB – –;
10.  if (frontB<rearB)
11.  frontB=rearB=MAX;
12.  return val;
13. } }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Code for Display in Multiple Queues:for Queue A & Queue B


```
1.  void display_queueA()
2.  {
3.  int i;
4.  if(frontA==–1)
5.   printf("\n QUEUE A IS EMPTY");
6.  else
7.  {
8.   for(i=frontA;i<=rearA;i++)
9.   printf("\t %d",QUEUE[i]);
10. }
11. }
```

```
1.  void display_queueB()
2.  {
3.  int i;
4.  if(frontB==MAX)
5.   printf("\n QUEUE B IS EMPTY");
6.  else
7.  {
8.   for(i=frontB;i>=rearB;i—)
9.   printf("\t %d",QUEUE[i]);
10. }
11. }
```


Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Write a program to implement multiple queues

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

APPLICATIONs OF QUEUES

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

APPLICATIONs OF QUEUES

- It is used as **waiting lists** for a single shared resource like printer, disk, CPU.
- Queues are used to **transfer data asynchronously** (data not necessarily received at same rate as sent) between two processes (IO buffers), e.g., pipes, file IO, sockets.
- Queues are used as **buffers** on MP3 players and portable CD players, iPod playlist.
- Queues are used in **Playlist for jukebox** to add songs to the end, play from the front of the list.
- Queues are used in operating system for **handling interrupts**. When programming a real-time system that can be interrupted, for example, by a mouse click, it is necessary to process the interrupts immediately, before proceeding with the current job. If the interrupts have to be handled in the order of arrival, then a FIFO queue is the appropriate data structure.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

End of Unit II

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Do It Yourself

Review Questions:

- Study the concept of Recursion
- Study the types of Recursion: (Direct Recursion, Indirect Recursion, Tail Recursion)
- Compare Recursion versus Iteration
- Advantages and disadvantages of using a recursive program.

Programming Questions:

- Write a program to calculate the factorial of a given number using recursion
- Write a program to calculate the GCD of two numbers using recursive functions.
- Write a program to calculate exp(x,y) using recursive functions
- Write a program to print the Fibonacci series using recursion.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Review Questions on Stack

1. What do you understand by the terms: stack overflow and underflow?
2. Differentiate between an array and a stack.
3. How does a stack implemented using a linked list differ from a stack implemented using an array?
4. Differentiate between peek() and pop() functions.
5. Why are parentheses not required in postfix/prefix expressions?
6. Explain how stacks are used in a non-recursive program?
7. What do you understand by a multiple stack? How is it useful?
8. Explain the terms infix expression, prefix expression, and postfix expression. Convert the following infix expressions to their postfix equivalents

   (a) A – B + C (b) A * B + C / D (c) (A – B) + C * D / E – C

   (d) (A * B) + (C / D) – (D + E) (e) ((A – B) + D / ((E + F) * G))

   (f) (A – 2 * (B + C) / D * E) + F (g) 14 / 7 * 3 – 4 + 9 / 2

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Review Questions on Stack

1. Find the infix equivalents of the following postfix equivalents:
   (a) A B + C * D – (b) ABC * + D –
2. Give the infix expression of the following prefix expressions.
   (a) * – + A B C D (b) + – a * B C D
3. Convert the expression given below into its corresponding postfix expression and then evaluate it. Also write a program to evaluate a postfix expression. 10 + ((7 – 5) + 10)/2
4. Write a function that accepts two stacks. Copy the contents of the first stack in the second stack. Note that the order of elements must be preserved.(Hint: use a temporary stack)
5. Draw the stack structure in each case when the following operations are performed on an empty stack. (a) Add A, B, C, D, E, F (b) Delete two letters (c) Add G (d) Add H (e) Delete four letters (f) Add I
6. Differentiate between an iterative function and a recursive function. Which one will you prefer to use and in what circumstances?

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Programming Questions on Stack

1. Write a program to implement a stack using a linked list.
2. Write a program to convert the expression “a+b” into “ab+”.
3. Write a program to convert the expression “a+b” into “+ab”.
4. Write a program to implement a stack that stores names of students in the class.
5. Write a program to input two stacks and compare their contents.
6. Write a program to compute F(x, y), where F(x, y) = F(x-y, y) + 1 if y£x And F(x, y) = 0 if x<y
7. Write a program to compute F(n, r) where F(n, r) can be recursively defined as:
 F(n, r) = F(n–1, r) + F(n–1, r–1)
8. Write a program to compute Lambda(n) for all positive values of n where Lambda(n) can be recursively defined as: Lambda(n) = Lambda(n/2) + 1 if n>1 and Lambda(n) = 0 if n =1
9. Write a program to compute F(M, N) where F(M, N) can be recursively defined as: F(M,N) = 1 if1 M=0 or M≥N≥1 and F(M,N) = F(M–1,N) + F(M–1, N–1), otherwise
10. Write a program to reverse a string using recursion

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Review Questions on Queue

1. What is a priority queue? Give its applications.
2. Explain the concept of a circular queue? How is it better than a linear queue?
3. Why do we use multiple queues?
4. Draw the queue structure in each case when the following operations are performed on an empty queue.
   (a) Add A, B, C, D, E, F
   (b) Delete two letters
   (c) Add G
   (d) Add H
   (e) Delete four letters
   (f) Add I

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu

Programming Questions on Queue

1. Write a program to calculate the number of items in a queue.
2. Write a program to create a linear queue of 10 values.
3. Write a program to create a queue using arrays which permits insertion at both the ends.
4. Write a program to implement a deque with the help of a linked list.
5. Write a program to create a queue which permits insertion at any vacant location at the rear end.
6. Write a program to create a queue using arrays which permits deletion from both the ends.
7. Write a program to create a queue using arrays which permits insertion and deletion at both the ends.
8. Write a program to implement a priority queue.
9. Write a program to create a queue from a stack.
10. Write a program to create a stack from a queue.
11. Write a program to reverse the elements of a queue.
12. Write a program to input two queues and compare their contents.

Dr. Charanjeet Dadiyala | dadiyalacs@rknec.edu