DATA STRUCTURES

classmate
Date
Page 1

1. Data structure refers to the process of placing data in an organize manner on primary memory.

2. Data Bases means the process of storing the data in an organize manner on secondary memory.

-> Data structures:

1. Hold the data
2. Imposes certain restriction
3. Users Benefits from restrictions

-> Types:

1. Foundational Datastructures:

   * Arrays
   * Linked Lists

2. Derived Data structures:

   * stacks
   * Queue
   * Tree
   * Graph
   * set
   * Dictionary
   * Map

Scanned by CamScanner

-> STACK:

1. Restrictions imposed by the stack:

   * Last-In First out [LIFO]
   * First-In Last out [FILO]

* PUSH: process of putting the data into the stack
* POP: process of removing the data from the stack

-> QUEUE:

1. Restrictions imposed by the Queue:

   * Last-In Last out [LILO]
   * First-In First out [FIFO]

-> int p; // p is an integer.

-> int p[5]; // p is an Array of 5 integers

-> int *p; // p is a pointer to an integer.

-> int **p; // p is a pointer to pointer to an integer

-> int *p[5]; // p is an Array of 5 pointers to integer

-> int (*p)[5]; // p is a pointer to an Array of 5 integers

-> int *(*p)[5]; // p is a pointer to an Array of 5 pointess to integer.

Scanned by CamScanner

=> **Pointer:** mechanism gives us an alternative way to access the contents of memory location.

* Data can be accessed in two ways:

1. Direct method (Using the variable Name)
2. Indirect method (Using pointers)

=> **'*' in C language is used for 3 purposes:**

1. Used to declare a pointer variable
2. Used for multiplication purpose.
3. Used to penetrate or Dereference a pointer.

=> **Unstructured style:**

```
#include<stdio.h>
void main()
{
 int a,b,c;
 clrscr();
 printf("Hey there!");
 getch();
}
```

=> **Structured style:**

```
#include<stdio.h>

void greet(); // Function prototype
void main()
{
```

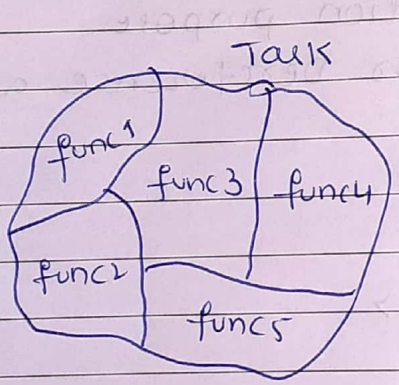
Scanned by CamScanner

```
clrscr();
greet(); // Function call or Function Signature
getch();
}

void greet()
{
   printf("Hey there!");
}
```

→ Function Body

→ When to use structured programming style:



- *. sub-tasks
- *. function
- *. procedure
- *. module
- *. routine
- *. sub-routine

→ Synonyms.

* complex & huge task.
* Division of labour also called as, dividing the task among different teams or members.
* Structured programming is also called as Procedure Oreinted programming language.

→ When to use Unstructured style:

(func) → Task.

* Simple & small task.
* No division of labour is involved, a single fellow can write a function & complete the task.
* Also called as Unstructured programming style.

Scanned by CamScanner

⇒ **STACK:** is a linear datastructure, in which insertion and deletion of elements is performed from only one end called top of the stacks. It works on the principle of LIFO or FILO.

* **Applications / Uses of STACK:**

1. To convert infix expression to postfix or prefix form.
2. To evaluate a postfix or infix expression.
3. Balancing of paranthesis.
4. palindrome
5. Reverse a string.
6. sort
7. Recursion.
8. convert decimal to Binary.

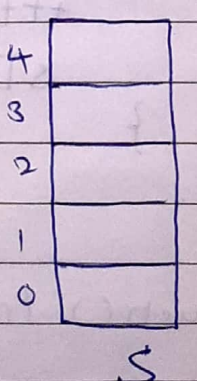
⇒ **Arrays Vs STACK:**

Arrays	Stacks
1. can hold data	1. can holds data.
2. Random insertion	2. orderly insertion.
3. Random deletion.	3. LIFO / FILO deletion.

⇒ **STACK creation:**

* **push method : [Pseudocode]:**

```
void push()
{
    if (stack is full)
        p/f "PUSH Not possible";
```



Scanned by CamScanner

```
else
 {
    Ask for the item to be pushed.
    collect the item
    increment top
    s[top] = item;
 }
}
```

* Conditions:

1. stack empty : if(top == -1)
2. stack full : if(top == SIZE - 1)

* push - method : [c code snippet]

```
void push()
{
  int item;
  if (top == SIZE - 1)
     printf ("PUSH operation NOT possible.");
  else
  {
     printf("\n Enter the item to be pushed : ");
     scanf ("%d", &item);
      ++top;
      s[top] = item;
  }
}
```

* push() insures orderly insertion.

Scanned by CamScanner

* Pop method : [pseudocode]

```
void pop()
{
   if (stack is empty)
      printf "POP Not possible";
   else
   {
      pop the element at top of stack;
      decrement top
   }
}
```

* conditions :

1. stack empty : if (top == -1)

* Pop method : [c code snippet] :

```
void pop()
{
   if (top == -1)
      printf ("POP Not possible");
   else
   {
      printf ("Element popped is %d", s[top]);
      --top;
   }
}
```

Scanned by CamScanner

* Display method : [pseudocode]

```
void display()
{
  if (stack is empty)
     printf (" Display not possible");
  else
     print all the elements present in the stack.
}
```

* C code Snippet :

```
void display()
{
   int i;
   if (top == -1)
      printf (" Display Not possible");
   else
   {
      for(i=0; i<=top; i++)
         printf (" %d \n", s[i]);
   }
}
```

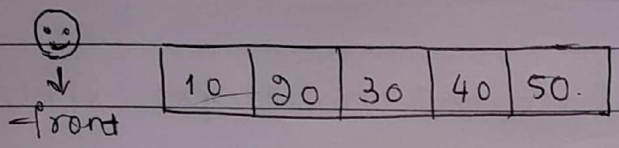
-> Implementing stack Using Local variables.

1. Make the global variables as local to main().
2. Pass the parameters from main to the other functions
3. The other functions must collect the incoming data
4. If any changes are made to the incoming data then the return statement must be used
5. Main has to collect the info back from the function

Scanned by CamScanner

Implementing STACKS in OOJ:

Queue: First-In First-out. or
Last-In Last-out



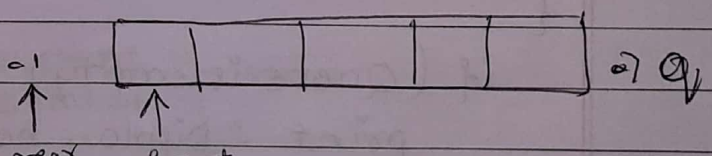
Application/Uses of Queues:

Arrays	Queues
* can holds data	can hold data
* Random Insertion	permits orderly insertion
* Random Deletion	permits LILO/FIFO deletion.

* Insertion : at rear end
Deletion : at front end.

{ rear = -1
front = 0 }

Insert() : [pseudocode]




```
void insert()
{
   if (Queue is full)
      print " Insertion Not possible";
   else {
      Ask for the item to be inserted.
      collect the item
      increment rear
      place item at q[rear]
} }
```

Scanned by CamScanner

=> conditions:

1. Queue full: rear == size - 1;
2. Queue Empty: front > rear.

=> delete(): [pseudocode]

```
void delet()
{
  if (Queue is Empty)
     print "Deletion not possible";
  else
  {
     delete the element at the front of Queue
     make the next element as the new front
        (By incrementing front)
  }
}
```

=> display(): [pseudocode]

```
void display()
{
  if (Queue is empty)
     print "Display not possible";
  else
     print all elements from front to rear;
}
```

Scanned by CamScanner

→ Applications of STACKS:

* prefix expression: Invented in 1924 by a poland, initially this notation is called as polish notation.

ex: +ab.

* postfix expression: Invented in 1954, also called as Reverse polish notation.

ex: ab+.

→ Steps for evaluating an prefix expression:

1. Reverse the prefix expression.
2. If you encounter, an operand → push
3. If you encounter, an operater → pop the top of stack & make it operand 1.
   → pop the next top of stack & make it operand 2
   → perform the operation
   → push the result.

ex: +4 * 2 3

pop: 3 2 * 4 +.

current symbol	stack
3	3
2	2, 3
*	2 * 3 = 6 → 6
4	4, 6
+	4 + 6 = 10 //

Scanned by CamScanner

→ Steps for Evaluating a postfix Expression:

1. No reversal required.
2. If operand → push
3. If operator → pop the top of stack & make it operand2
   → pop the next top of stack & make it operand1
   → perform operators push the result.

ex: 4 2 3 * +

current symbol	stack
4	4
2	2, 4
3	3, 2, 4
*	2 * 3 = 6 → 6, 4
+	4 + 6 = 10 → 10
No symbol	10 //

Scanned by CamScanner

Q1. convert & evalute the infix expression : (8+5)*(6/3).

soln: prefix : *+8 5 /63

1. 3 6 / 5 8 + *

2.

current symbol	stack
	[ ]
3	3
6	6 3
/	6/3 = 2
	2
5	5 2
8	8 5 2.
+	8+5 = 13
	13 2
*	13*2 = 26 //

Scanned by CamScanner

Q2. convert & evaluate the infix expression to postfix.
(8+5)*(6/3)

soln: postfix : 8 5 + 6 3 / *

current symbol	stacks
8	8
5	5 8
+	8+5 = 13 → 13
6	6 13
3	3 6 13
/	6/3 = 2 → 2 13
*	13*2 = 26
	26
NO symbol.	26 //

Scanned by CamScanner

$: power of symbol.

ex: 4 $ 2 = 4^2 = 16.

07 Pseudocode of prefix in JAVA:

```
create a new stack of Integers.
Reverse the prefix exp & traverse the entire length of prefix exp
{
  Consider the character at ith position as the current symbol
  if (the current symbol is a digit)
  {
    Obtain numerical value of the symbol & place it in num
    push num onto the stack.
  }
  else
  {
    pop top of stack & place it in opnd1.
    pop top of stack & place it in opnd2.

    verify (if symbol is)
    {
      case '+': then compute opnd1 + opnd2 and push
                it onto the stack.
      case '-' : then compute opnd1 - opnd2 and push
                it onto the stack.
      case '*':
      case '/':
    }
  }
}
pop the top of stack & place it in ans
```

Scanned by CamScanner

Q7 JAVA program to evaluate a prefix expression:

```
import java.util.Scanner;
import java.util.Stack;

class prefix
{
  String exp;
  int ans;

  public void getInput()
  {
    System.out.println("Enter a prefix Expression");
    Scanner sc = new Scanner(System.in);
     exp = sc.next();
  }

  public void evalPre()
  {
    Stack<Integer> stk = new Stack<Integer>();
    for(int i = exp.length()-1; i>=0; --i)
    {
      char symbol = exp.charAt(i);
      if(Character.isDigit(symbol))
      {
        int num = Character.getNumericValue(symbol);
        stk.push(num);
      }
    else
    { int opnd1 = stk.pop();
      int opnd2 = stk.pop();

     switch(symbol)
     {
```

Scanned by CamScanner

```
case '+': stk.push(opnd1 + opnd2);
          break;

case '-': stk.push(opnd1 - opnd2);
          break;

case '*': stk.push(opnd1 * opnd2);
          break;

case '/': if(opnd2 == 0)
          {
           System.out.println("Division by zero not
            possible");
           System.exit(0);
          }
  else
  {
    stk.push(opnd1/opnd2);
    break;
  }
 }
 }
}
  ans = stk.pop();
}

public void display()
{
  System.out.println("Evaluation of prefix Exp: "+ans");
}
}
```

Scanned by CamScanner

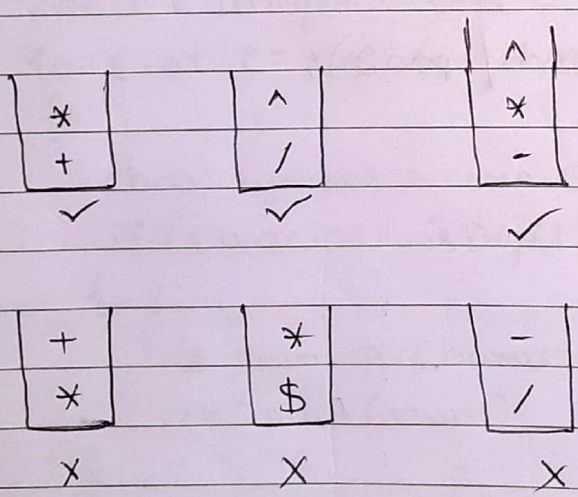
```
class Evalprefix {
    public static void main (String args[])
{
    Prefix x = new Prefix();
    x.getInput();
    x.evalpre();
    x.display();
    }
}
```

→ Conversion of Expressions:

* PRIORITIES:

1. Exponential ($, ^)
2. Multiplication, Division (*, /)
3. Addition, subtraction (+, -)

Ex:

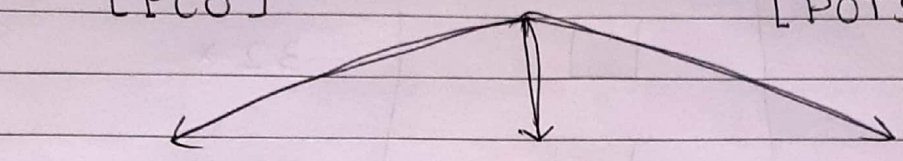


→ Steps for converting an infix Expression to prefix:

1. Reverse the Infix Expression.
2. If operand • send it to prefix expression.
3. If operator, then 3 possibilities exists.

Scanned by CamScanner

priority of current operator [PCO] Vs Priority of top of stack, [POTS].



PCO > POTS → PUSH

PCO == POTS → PUSH

POC < POTS → POP

4. parenthesis

- → (push
- →) push
- → permitted to push any symbol above the brackets
- → pop all symbols between (and).

5. Reverse the final expression.

Q1. Given, infix: 4 + 2 * 3.

Soln: Reverse: 3 * 2 + 4.

current symbol	STACK	Prefix Expression
.		
3		3
*		3
2		3 2

Scanned by CamScanner

+	*	32
		32 *
	+	32 *
4	+	3 2 * 4
no symbol		3 2 * 4 +

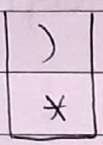
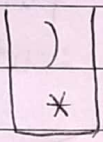
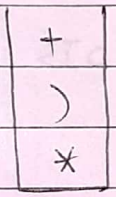
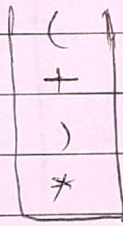
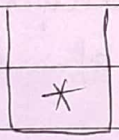
Prefix Expression: + 4 * 2 3

Q2. (8+5) * (6/3)

Soln:) 3 / 6 (*) 5 + 8 (

current symbol	STACK	prefix Expression
)	)	
3	)	3
/	/)	3
6	/)	3 6
(	(/)	3 6

Scanned by CamScanner

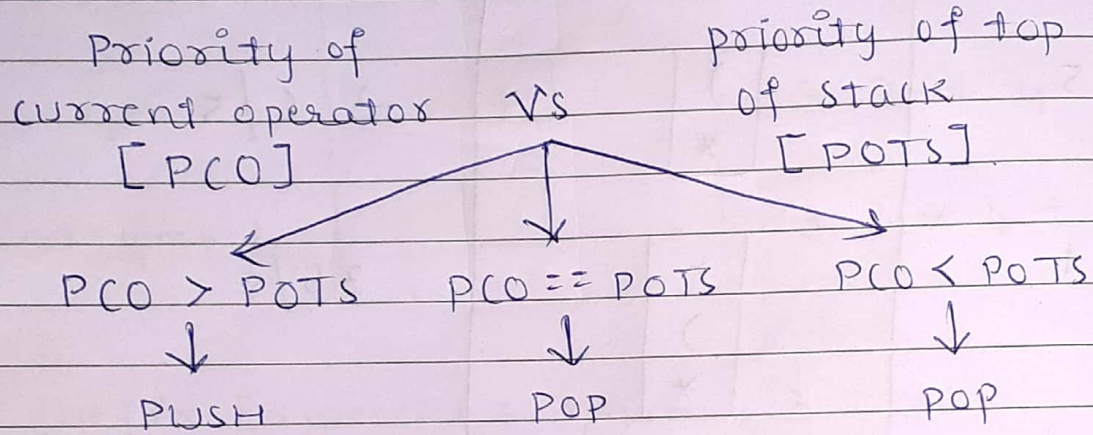
		3 6 /
*		3 6 /
)		3 6 /
5		3 6 / 5
+		3 6 / 5
8		3 6 / 5 8
(		3 6 / 5 8
		3 6 / 5 8 +
no symbol.		3 6 / 5 8 + * .

prefix Expression: * + 8 5 / 6 3 .

Scanned by CamScanner

-> Steps for converting a infix Expression to postfix:

1. No need to reverse the infix Expression.
2. If operand - send it to postfix Expression.
3. If operator,



4. Parenthesis:
   - -> (: push
   - ->) : push
   - -> permitted to push any symbol above the brackets
   - -> pop all symbols b/w (and)

5. No need to Reverse the final expression

Q1. 4 + 2 * 3

Soln:

Current Symbol	STACK	Postfix Expression
4		4
+	+	4
2	+	42

Scanned by CamScanner

*	* +	4 2 .
3	* +	4 2 3
no symbol		4 2 3 * +

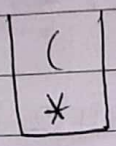
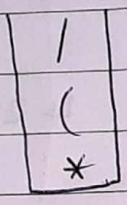
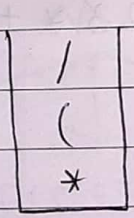
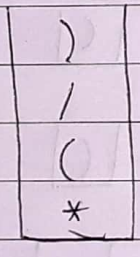
postfix Expression: 4 2 3 * + .

Q2. (8 + 5) * (6 / 3).

Ans:

current symbol	STACK.	postfix Expression.
(	(	
8	(	8 .
+	+ (	8
5	+ (	8 5 .
)	) + (	8 5
		8 5 + .
*	*	8 5 + .
(	(*	

Scanned by CamScanner

6		85+6
/		85+6.
3		85+63.
)		85+63
		85+63/
no symbol		85+63/*.

postfix: 85+63/*.

Scanned by CamScanner

⇒ **Rules to convert prefix to Infix Expression:**

1. push every symbol encountered in the prefix Expression.
2. Search for the pattern

operand 2
operand 1
operator } and replace it with

operand1 operator operand2 } pattern.

3. Treat the new pattern as operand1 if it is immediately present above the operator. otherwise treat it as operand2.

Q1. + 4 * 2 3

Soln:

current symbol	STACK.
+	+
4	4 +
*	* ~~opnd1~~ oper 4 opnd1 + oper
2	2 opnd1 * ~~opnd1~~ oper 4 opnd1 + oper

Scanned by CamScanner

2

2
*
4
+

3

3	opnd2
2	opnd1
*	oper
4	opnd1
+	oper

2 * 3	opnd2	
4	opnd1	
+	oper	4 + 2 * 3

Infix Expression : 4 + 2 * 3

→ Rules to prefix to postfix Expression:

1. push every symbol encountered in the prefix expression
2. search for the pattern

opnd2
opnd1 } pattern & replace it with
oper

opnd1 opnd2 oper } pattern

Scanned by CamScanner

3. Treat the new pattern as opnd1 if it is immediately present above the operator. other treat it as opnd2.

→ Single statement for push operation:

```
void push(int item)
{
    S[++top] = item;
}
```

→ Single statement for pop operation:

```
int
void pop()
{
  return S[top--];
}
```

I. strrev() → i/p: ABCID
o/p: DICBA.

II sentrev() → i/p: ABCID is compulsory
o/p: yrostupmoc si DICBA

III. strpal → i/p: ABCID
o/p: Not a palindrome

i/p: MADAM
o/p: palindrome.

IV. sentwordrev() → i/p: ABCID is compulsory
o/p: DICBA si yrostupmoc.

Scanned by CamScanner

→ Reversal of a String using STACK :

Ask for the input string
collect the input string

As long as (not the end of string)
{
consider the current symbol as item
push the item
proceed
}

As long as (the stack is not empty)
{
pop & place it in the rev array
proceed
}
print the rev array

→
```
void strrev()
{
 int i=0, j=0;
 char str[200], rev[200];
 char item;
 p/f ("Enter string : ");
 s/f ("%s", str);          // gets(str); Sentence Reversal

   while (str[i] != '\0')
   {
     item = str[i];
      push (item);
       ++i;
   }
```

Scanned by CamScanner

```
while(top != -1)
{
  rev[j] = pop();
   ++j;
}
  rev[j] = '\0';
  p/f("%os", rev);
}
```

⇒ palindrome:

```
void palin()
{
  int i=0, j=0;
  char str[200], rev[200], item;

  printf("Enter a string:");
  scanf("%s", str);

  while(str[i] != '\0')
  {
    item = str[i];
    push(item);
     ++i;
  }

  while(top != -1)
  {
    rev[j] = pop();
     ++j;
  }
   rev[j] = '\0';
```

Scanned by CamScanner

```
if (strcmp(str, rev) == 0)
    pf("palindrome");
else
    pf("Not palindrome");
}
```

7) Reversal of individual words in a Sentence:

```
void sentwordrev()
{
  int i=0, j=0;
  char str[200], rev[200], item;

  pf("\nEnter a sentence: ");
  ~~sf("%d", gets(str);~~

  while (str[i] != '\0')          // — (I) —
  {
    while (str[i] != ' ' && str[i] != '\0')  // (II)
    {
      item = str[i];
      push(item);
      ++i;
    }
    while (top != -1)             // — (III) —
    {
      rev[j] = pop();
      ++j
    }
    rev[j] = '\0';
    pf("%s", rev);    ++i; // it reset the loop index
                           variable i to the beginning
                           of the next word.
  }
}
```

Scanned by CamScanner

(I) It works on all the words of A given sentence [1st while loop].

(II) pushing every character of the word into the stack. [2nd while loop]

(III) poping every character of the stacks & finally printing the reversed word.

-> Balanced paranthesis : [simple paranthesis] [pseudocode]

As for the string containing simple paranthesis collect the string & place it in the array str.

```
As long as (not the end of string)
{
  consider the current symbol as item
  if (item is '(')
      push the item
  else
      pop the top of stack.
  proceed
}

if (stack is Empty)
    print "Balanced";
else
    print "Not Balanced";
```

Scanned by CamScanner

```
void simpleParanthesisBalance()
{
   char str[100], item
   int i

   p/f ("\n Enter a string of simple paranthesis : ");
   s/f ("%s", str);
   i=0;

   while (str[i] != '\0')
   {
   if(item == '(')
         push (item);
    else
         pop();
   ++i
   }

   if (top == -1)
      p/f ("Balanced");
   else
      p/f ("Not Balanced");
}
```

-> [Complex Balanced paranthesis]:

```
void complexParanthesisBalance()
{
   char str[100], item;
   int i=0;
   p/f ("Enter a string of paranthesis:");
   s/f ("%s", str);
```

Scanned by CamScanner

```
while(str[i] != '\0')
{
    item = str[i];
    if( item == '(' || item == '[' || item == '{')
        push(item);
    else if(item == ')' || item == ']' || item == '}')
        pop();
    ++i;
}
 if(top == -1)
    pf("Balanced");
 else
    pf("UnBalanced");
}
```

→ Conversion of Decimal Number to Binary form:

```
void decTobin()
{
  int num; int rem;
  pf("Enter the number to be converted :\n");
  sf("%d", &num);

  while(num != 0)
  {
     rem = num % 2;
     num = num/2;
     push(rem);
  }
  display();
}
```

Scanned by CamScanner

→ pseudocode: [Decimal to Binary using stack]:

```
As long as (the num is not equal to zero)
{
    rem = num%2;
    num = num/2;
    push the rem;
}
  display the stack
```

→ Search for the given key element: [pseudocode]

```
if(no data in the data structure)
    print "operation not possible";
else
{
    Ask for the key to be searched collect the key.
    for(every data present in the data structure)
    {
        remove the data & place it in temp
        if(temp matches with the key)
        {
            print "key found"
            Stop
        }
    }
  print "key not found";
}
```

Scanned by CamScanner

*. Search for the given key Using STACK :

```
void searchKey()
{
   int key, i, temp;
   if (top == -1)
      printf ("\n operation not possible ");
   else
   {
      printf ("Enter key to be Searched : ");
      scanf ("%d", &key);

      for (i = top; i >= 0; i--)
      {
         temp = pop();
         if (temp == key)
         {
            printf ("key found");
            getch();
            exit (0);
         }
      }
      printf ("key not found");
   }
}
```

Scanned by CamScanner

* Search for the given key Using QUEUE:

```
void searchkey()
{
   int key, i, temp;
   if (front > rear)
      printf (" search not possible ");
   else
   {
      printf (" Enter the key to be searched: ");
      scanf ("%d", &key)

      for ( i = front; i <= rear; i++)
       {
          temp = delet();
          if (temp == key)
          {
            printf (" key found ");
             getch();
             exit(0);
          }
       }
      printf (" key not found ");
   }
}
```

Scanned by CamScanner

=> Search for the multiples of a given number: [STACK]

```
void searchmul()
{
  int num, i, temp;
   if (top==-1)
     printf(" operation not possible");
  else
  {
     printf("Enter number whose multiples are to be
          searched: ");
     scanf("%d", &num);

    for(i=top; i>=0; i--)
     {
        temp = pop();
        if (temp % num == 0)
           printf("%d", temp);
     }
  }
}
```

=> Search for the maximum [Largest] Element in the STACK:

```
void searchMax()
{
  int max=-9999,i,temp;
  if (top==-1)
     printf(" operation not possible.");
  else
  {
```

Scanned by CamScanner

```
for (i=top; i>=0; i--)
 {
    temp = pop();
     if (temp > max)
         max = temp;
 }
 printf ("%d", max);
 }
}
```

Q) Pseudocode for searching maximum Element in a STACK or QUEUE:

```
if (no data in the data structure)
    print " operation not possible ";
else
{
   max = -9999;
   for (every data present in the data structure)
    {
      remove the data & place it in temp
       if (temp > max)
          update the max
    }
  print max
}
```

Scanned by CamScanner

⇒ Reversal of a QUEUE: [Pseudocode]

```
if (no data in the Queue)
   print " Reversal not possible ";
else
{
   for(every data present in the queue)
   {
      delete the data from queue & place it in temp
      push temp into the stack
      increment count
   }

   for(every data present in the stack)
   {
      pop from the stack & place it in temp
      insert temp into the queue
   }
   Display the queue
}
```

⇒ Code for Reversal of a Queue:

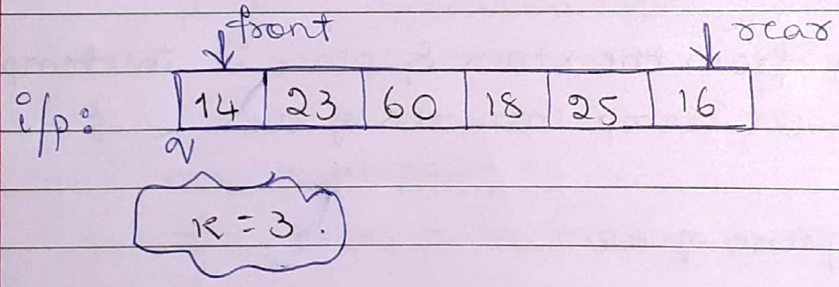
```
void reverseQ()
{     int temp,count=0;
   if(front > rear)
      printf(" Reversal not Possible");
   else
   {
      for (int i=front; i<=rear; i++)
      {
         temp=delet();
```

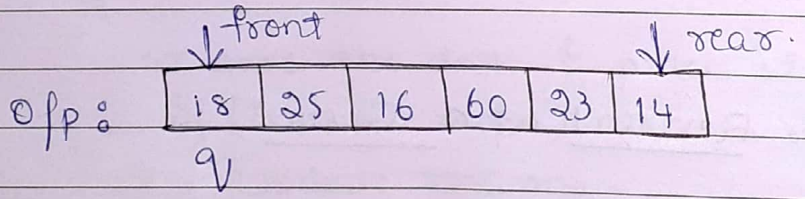
Scanned by CamScanner

```
        push(temp)
        ++count;
    }
    for(int i=0; i<=count-1; i++)
    {
        temp = pop();
        q[++rear] = temp;
    }

    display();
  }
}
```

-> Partial Reverse of a Queue:






```
void partialReverse()
{
    int i, temp, k, count=0;
    if (front>rear)
        printf("Reversal not possible");
    else
    {
```

Scanned by CamScanner

```
printf("Enter the number of items to be Reversed
                                    ");
scanf("%d", &k);

for(i=front; i<=k-1; i++)
{
    temp = delet();
    push(temp);
    ++count;
}

for(i=0; i<=count-1; i++)
{
    temp = pop();
    q[++rear] = temp;
}
display();
}
}
```

Scanned by CamScanner