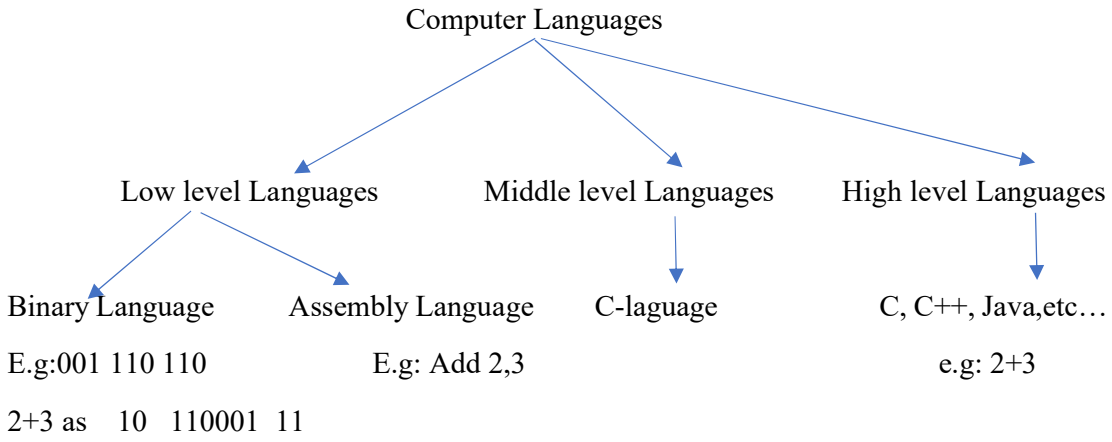
Subject: Data Structures

Introduction to Data Structures

What is Data Structure?

Data structure is a way of organizing all data items that considers not only the elements stored but also their relationship to each other.

Computer Languages



Traslators:

Interpreter: Translates High level language to low level

It takes more time to execute-----disadvantage

It takes very less memory------- advantage

Compiler:

Translates High level language to low level

It takes less time to execute-----Advantage

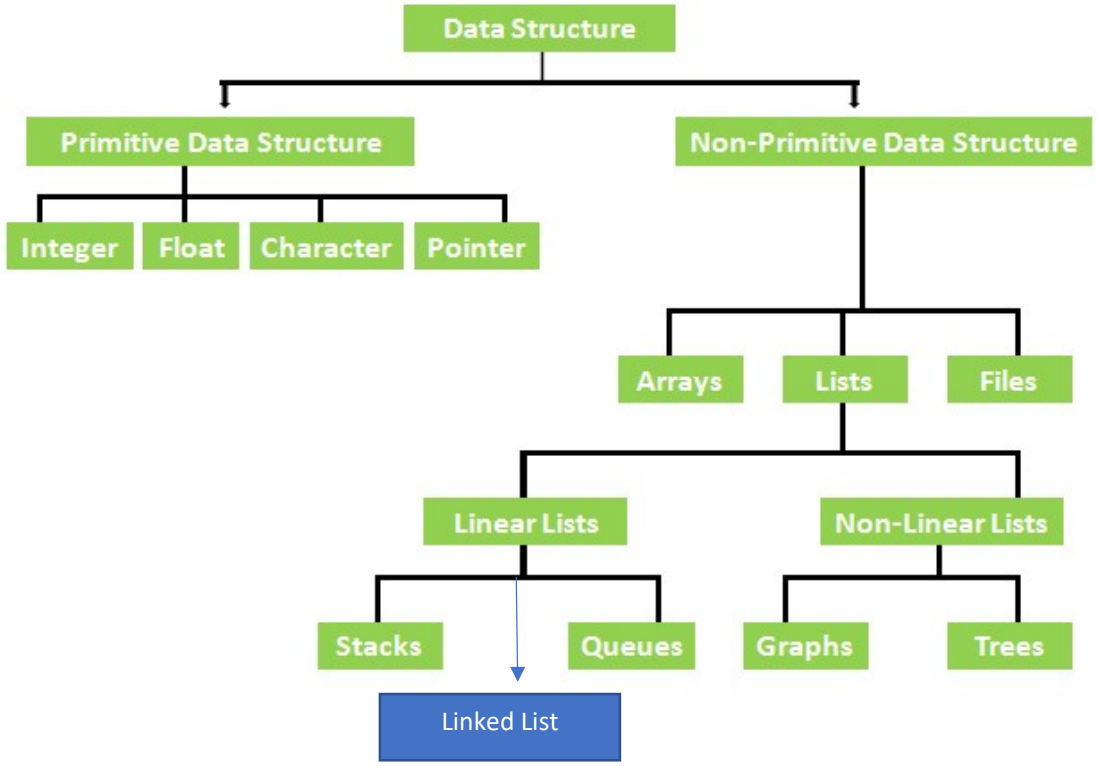
It takes more memory------- Disadvantage

Importance of Data Structure:

Program = Data Structure+ Algorithm

Subject: Data Structures

Classification of Data Structures: (19.08.2020, 8.00 AM to 9.00AM)



Difference between Primitive and Non-primitive Data Structures?

Linear Data Structure: in this all the data items are stored in linear order/sequential

Non-linear Data Structure: in this all the data items are stored in non-linear order/hierarchical manner.

Stack: It is a Linear Data Structure is a LIFO / FILO data structure- last in first out data structure

In real life: CD storage container, file storing, Idly container, etc…

In Computer Science: Function call management.

Queue: It is a Linear Data Structure, is an FIFO data structure-First in First OUT

In real life: queue system in temples, train reservations

Computer science : Customer call management and execution of program statements.

Prepared By: Suresh Kumar K

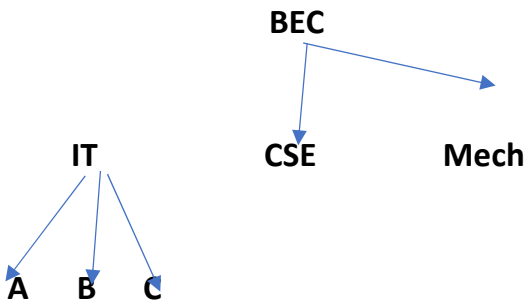
```
main()
{
int a,b,c;
a=10;
b=5;
c=a+b;
}
```

Linked List : it is a linear data structure in which the linear order is maintained by Pointers.

3	12	4	13	6	14	8	15	9	16

Store 5 elements:

Linked list will support in these cases.



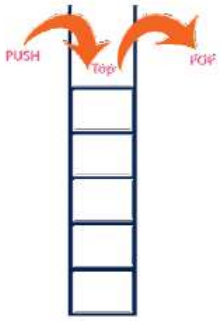
```
main()
{
                Add();
                Sub();
                Mul();
}
  Void add()
{
       Div();

}
Void div()
{
Sdfds----------after this
}
Stack:
2000
1000
```

Stack ADT: Stack is a linear data structure in which the insertion and deletion operations are performed at only one end which is known as Top of the stack.

(Or)

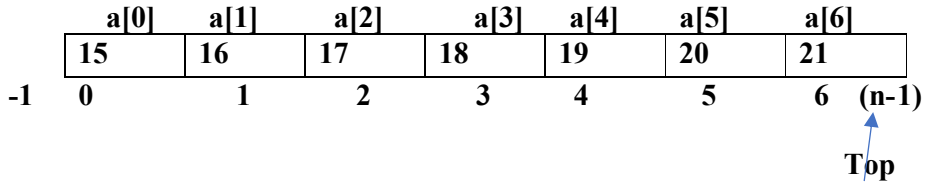
Stack is a collection of similar data items in which both insertion and deletion takes place at one end. That end is called as Top of stack. Which works with basic principle named LIFO (Last In First Out) or FILO (First In Last Out)

Representation of Stack:



Stack

Subject: Data Structures



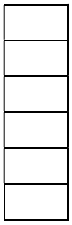
Stack

Basic Operations Performed on Stack:

1. **Push():** To insert an element on to the stack is called as Push operation.
2. **Pop():** To delete an element from stack is called as POP operation.
3. **Display():** To display elements in the stack.

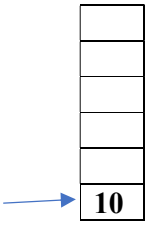
Stack Underflow (Stack empty): When top of the stack points to -1, then stack is underflow.
Stack Overflow(Stack full): When top of the stack reaches to "size-1"(n-1), then stack is overflow.

I want to insert 10,45,12,16,35 and 50.

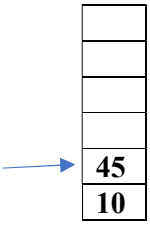


Top=-1

Insert 10 on to the stack



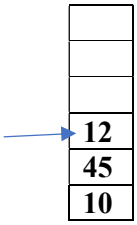
Insert 45



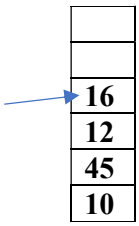
Prepared By: Suresh Kumar K

Subject: Data Structures

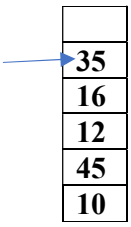
Insert 12



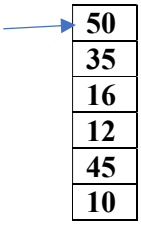
after inserting 16



after inserting 35

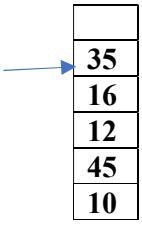


after inserting 50



I want to insert 75, top=size-1, stack is overflow

First deleting element is always 50 here
After deleting 50



Second deleted element is always 35

Prepared By: Suresh Kumar K

Subject: Data Structures

Stack Applications:

There are so many applications related to computer science, some of them are

1. Function call management
2. Conversion of Infix to postfix expressions
3. Evaluation of postfix expressions
4. Pattern matching etc…

What is expression?

It is a combination of both operands and operators which gives you a specific value/result. There are different notations to represent an expression. They are

1. Infix expression: It is an arithmetic expression in which an operator is placed in between the operands.
e.g: a+b or 2+3

2. Prefix expression (Polish Notation): It is an arithmetic expression in which an operator is placed before the operands.
e.g: +ab, or +23

3. Post-fix expression (Reverse Polish Notation): It is an arithmetic expression in which an operator is placed after the operands.
e.g: ab+ or 23+

Algorithm for Infix to Postfix Conversion:

1. Scan the infix expression from left to right.
2. If the scanned character is an operand, then place it into postfix expression (output it).
3. Else,

…..3.1 If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), then push it on to the stack.

…..3.2 Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack. (If you encounter parenthesis while popping then stop there and push the scanned operator in the stack.)

4. If the scanned character is an '(', push it to the stack.
5. If the scanned character is an ')', pop all the symbols from stack and output it until a '(' is encountered, and discard both the parenthesis.
6. Repeat steps 2-6 until infix expression is scanned.
7. if the scanned symbol is null character then Pop all the symbols from stack and placed it into postfix expression (output it).

Prepared By: Suresh Kumar K

E.g: Convert the infix expression a+b*c/d-e into postfix expression using stack.

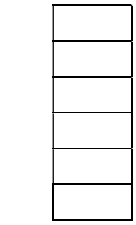
Solution:

Step1: Scan the given infix expression from left to right. i.e.,

Given expression: **a + b * c / d - e**

Step2: Current scanned symbol is an operand i.e., ***a*** then placed it into postfix expression

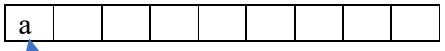
Post fix expression



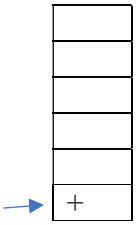
Top -1

Stack

Step3: Current scanned symbol is an operator i.e., + , here stack is empty then push it on to the stack

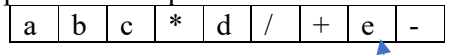


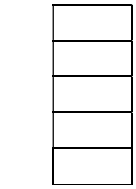
Post fix expression



Stack

Step4: Current scanned symbol is an operand i.e., b then placed it into post fix





Step5: current scanned symbol is an operator i.e., * here scanned symbol is greater than stack top symbol i.e., * > + then push it on to the stack.
Step6: current scanned symbol is an operand i.e., c then placed it into postfix
Step7: Current scanned symbol is an operator i.e., / then pop * and place it into postfix and then push / on to the stack.
Step8: Current symbol is an operand i.e., d then placed in postfix
Step9: Current scanned symbol is an operator i.e., - then pop all symbols from stack until it is greater.
Step10: Current scanned symbol is an operand i.e., e then place it into postfix.
Step11: current scanned symbol is NULL then pop all symbols from stack then place it into postfix.

Therefore postfix expression is abc*d/+e-

E.g: Convert a ^ b – c / d * e + f – g / h – i to postfix expression using stack

Postfix expression is: **a b ^ c d / e * - f + g h / - i -**

Problem No:5 Convert the following infix expression to postfix notation using stack.

a + (b * c - (d / e ^ f) * g) * h

solution: a b c * d e f ^ / g * - h * +

Problem No:6 Convert the following infix expression to postfix notation using stack.

((AX + (B * C Y)) / (D - E))

Problem No:7 Convert the following infix expression to postfix notation using stack.

((H * ((((A + ((B + C) * D)) * F) * G) * E)) + J)

Problem No:8 Convert the following infix expression to postfix notation using stack.

(A + B) * C + D / (E + F * G) – H

Solution: **A B + C * D E F G * + / + H -**

Subject: Data Structures

Evaluation of Postfix expression:

1. Read all the symbols one by one from left to right in the given Postfix Expression
2. If the reading symbol is an operand, then push it on to the Stack.
3. If the reading symbol is an operator (+ , - , * , / etc.,), then perform TWO pop operations and store the two popped operands in two different variables (operand1 and operand2). Then perform reading symbol operation using operand2 and operand1 and push result back on to the Stack.
4. Finally! perform a pop operation and display the popped value as final result when the reading symbol is NULL.

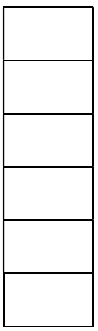
Example: Evaluate the following postfix expression

5, 3 ,+ ,8 , 2 , -, *

Solution:

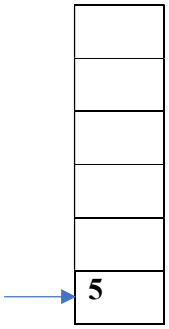
Step 1: Given Postfix expression is 5 3 + 8 2 - *

Step2: Read the expression from left to right i.e., 5, 3 ,+ ,8 , 2 , -, *



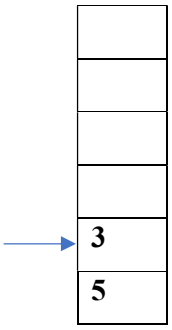
Top=-1

Step3: Current reading symbol is an operand i.e., 5 then push it on to the stack



Step4: 5, 3 ,+ ,8 , 2 , -, *

Current scanned symbol is 3 i.e., an operand then push it on to the stack

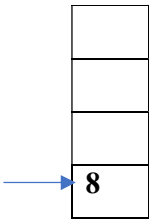


Stack

Step5: : 5, 3 ,+ ,8 , 2 , -, *

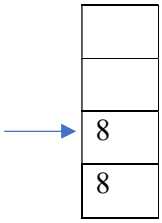
If current scanned symbol is an operator i.e., + then pop two operands from stack and perform corresponding operation on those two operands.

Operand1=3
Operand2 = 5
Operation= 5+3 (operand2 + operand1) =8 then push the result back in to the stack

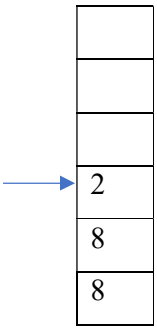


Stack

Step6: Current scanned symbol is an operand i.e., 8 then push it on to the stack

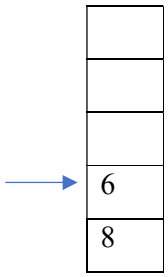


Step7: Current scanned symbol is an operand i.e., 2 then push it on to the stack



Step8: : Current scanned symbol is an operator i.e., - then pop two operands from stack and perform corresponding operation i.e.,

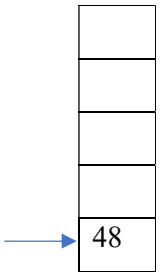
Operand1= 2
Operand2= 8 then Operation = 8 – 2 = 6 push result back in to the stack.



Stack

Step9: Current scanned symbol is an operator i.e., * then pop two operands from stack and perform corresponding operation i.e.,

Operand1= 6
Operand2= 8 then Operation= 8 * 6= 48



Step10: current scanned symbol is NULL then pop the value from stack and display as result

Evaluation of : 5, 3 ,+ ,8 , 2 , -, * is 48.

Subject: Data Structures

Table Representation:

Given Postfix expression: 5, 3 ,+ ,8 , 2 , -, *

Current Scanned Symbol	Stack	Operation
5	5	Push
3	3 5	Push
+	8	Op1=3 Op2=5 5+3=8
8	8 8	push
2	2 8 8	push
-	6 8	Op1=2 Op2=8 8-2=6
*	48	Op1=6 Op2=8 8*6=48
NULL	Empty	Result=48

The final result is 48

Problem N0:2 Evaluate the expression 7+3-8/2*5^3+2
Postfix notation for given expression is **7 3 + 8 2 / 5 3 ^ * - 2 +**

Current scanned symbol	Stack	Action/operation
7	7	push
3	3 7	push
+	10	Op1=3 Op2=7 7+3=10
8	8 10	push

Prepared By: Suresh Kumar K

2	**2** **8** **10**	**Push**
/	**4** **10**	**Op1=2** **Op2=8** **8/2=4**
5	**5** **4** **10**	**push**
3	**3** **5** **4** **10**	**push**
^	**125** **4** **10**	**Op1=3** **Op2=5** **5^3= 125**
*	**500** **10**	**Op1=125** **Op2=4** **4*125= 500**
-	**-490**	**Op1=500** **Op2=10** **10-500=-490**
2	**2** **-490**	**push**
+	**-488**	**Op1=2** **Op2=-490** **-490+2= -488**
NULL	**empty**	**Result= -488**

Result of the given expression is -488

Problem No: 3 Evaluate **((10 * (6 / ((9 + 3) * 11))) + 17) + 5**

Problem No: 4 Evaluate **6 2 3 + - 3 8 2 / + * 2 $ 3 + using Stack.**

Problem No: 5 Evaluate **7 2 3 * 5 + 8 4 2 / - * - using Stack.**

Problem No: 6 Evaluate **3 6 + 1 + 2 5 * 4 + * 8 7 + * using Stack.**

Subject: Data Structures

Program No:2 Write a C-program to implement Evaluation of postfix expression.

Out put:1. Enter any valid postfix notation: 23+
The result is 5
2. Enter any valid postfix notation: 234+*
The result is 14

```
char  str[10];

printf("\n Enter any valid postfix expression");
scanf("%s", str);
```

23+

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]				str[9]
2	3	+	\0	\0	\0	\0	\0	\0	\0
0	1	2	3	4	5	6	7	8	9

Step1: read the symbols from left to right in a given expression.

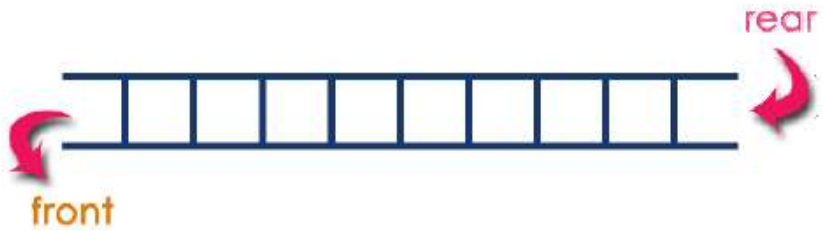
Queue:

Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends.

In a queue data structure, the insertion operation is performed at a position which is known as '**rear**' and the deletion operation is performed at a position which is known as '**front**'.

In queue data structure, the insertion and deletion operations are performed based on **FIFO** (First In First Out) principle.

Basic representation of Queue:



Basic operations performed on Queue:

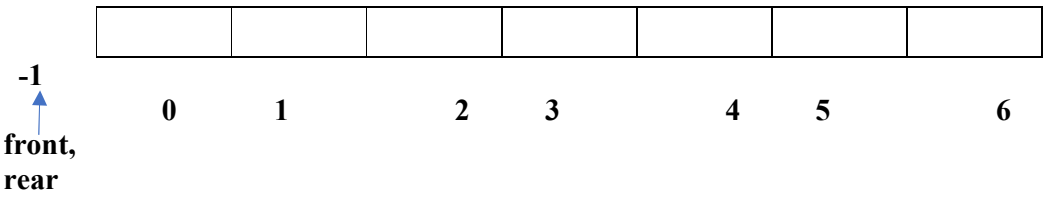
1. Enqueue(): To insert an element into queue is called as Enqueue.

2. Dequeue(): To delete an element from queue is called as Dequeue.

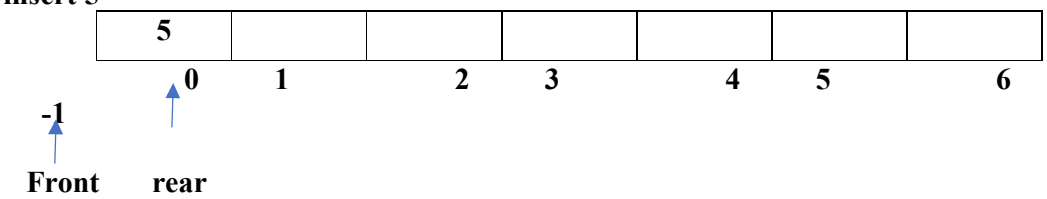
3.Display(): To display the elements in queue.

Insert the following elements 5,6,7,8,9,10,11 into queue

Initial position of queue:

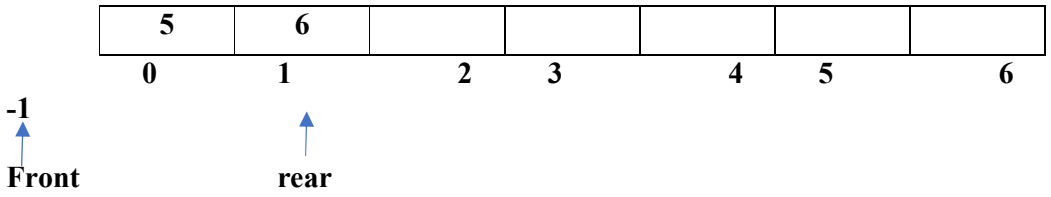


insert 5

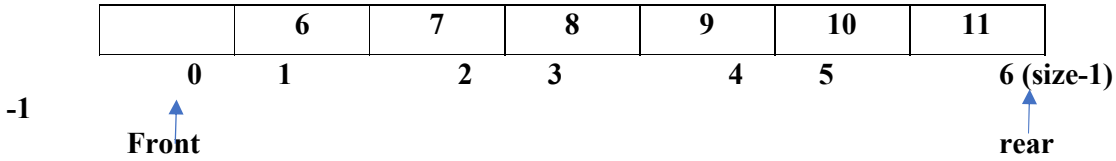


Subject: Data Structures

Insert 6:



Insert 7:



Queue is full: front=-1 and rear=size-1

Queue is empty: front=-1 and rear=-1 or front== rear

First deletion operation: first deleted element is 5

Insertion:

Rear++;
Queue[rear]=element;

Deletion:

Front++;
Queue[front]

Prepared By: Suresh Kumar K

Subject: Data Structures

Enqueue() operation:

The following steps represent how to perform insertion in Queue:

Step 1: Check whether **queue** is **FULL**. (**rear == SIZE-1 && front=-1**)

Step 2: If it is **FULL**, then display **"Queue is FULL!!! Insertion is not possible!!!"** and terminate the function.

Step 3: If it is **NOT FULL**, then increment **rear** value by one (**rear++**) and set **queue[rear]** = **value**.

Dequeue() Operation:

The following steps to delete an element from queue.

Step 1: Check whether **queue** is **EMPTY**. (**front == rear || front==-1 && rear==-1**)

Step 2: If it is **EMPTY**, then display **"Queue is EMPTY!!! Deletion is not possible!!!"** and terminate the function.

Step 3: If it is **NOT EMPTY**, then increment the **front** value by one (**front ++**). Then display **queue[front]** as deleted element. Then check whether both **front** and **rear** are equal (**front == rear**), if it **TRUE**, then set both **front** and **rear** to '**-1**' (**front = rear = -1**).

Display() operation:

The following steps to display the elements of a queue...

Step 1: Check whether **queue** is **EMPTY**. (**front == rear**)

Step 2: If it is **EMPTY**, then display **"Queue is EMPTY!!!"** and terminate the function.

Step 3: If it is **NOT EMPTY**, then define an integer variable '**i**' and set '**i = front+1**'.

Step 3: Display '**queue[i]**' value and increment '**i**' value by one (**i++**). Repeat the same until '**i**' value is equal to **rear** (**i <= rear**).

Prepared By: Suresh Kumar K

Subject: Data Structures

Applications of Queue (linear queue):

1. Serving requests on a single shared resource, like a printer, CPU task scheduling etc.
2. In real life scenario, Call Center phone systems uses Queues to hold people calling them in an order, until a service representative is free.
3. Handling of interrupts in real-time systems. The interrupts are handled in the same order as they arrive i.e First come first served.
4. Used to implement Radix sort.

Program No:4 Write a C-program for converting Infix to postfix expression.

```
#include<stdio.h>
#include<conio.h>
#define size 20
int prec(char op);
void infix(char exp[20]);

char stack[size];
char post[20];
int top=-1;
int j=0,i=0,k;
int cr,st;

 exp= a+b*c/d-e
```

a	+	b	*	c	/	d	-	e	\0
exp[0]	exp[1]	exp[2]	3	4	5	6	7	8	exp[9]

exp

```
 void main()
{
 char exp[20];
 clrscr();
 printf("\nEnter infix expression: ");
scanf("%s",exp);
infix(exp);
 getch();
}
void infix(char exp[20])
{
  while(exp[i]!='\0')
 {
  if(exp[i]=='(')
```

```
 {
  top++;
  stack[top]=exp[i];
 }
 else if(exp[i]>='a'&&exp[i]<='z'||exp[i]>='A'&&exp[i]<='Z'||exp[i]>='0'&&exp[i]<='9')
 {
   post[j]=exp[i];
   j++;
 }
 else if ( exp[i]=='+' || exp[i]=='-' || exp[i]=='*' || exp[i]=='/' || exp[i]=='^' )
 {
   cr=prec(exp[i]);
   st=prec(stack[top]);



if(top==-1)
   {
    top++;
    stack[top]=exp[i];
   }
   else
   {
   while(st>=cr)
   {
    post[j]=stack[top];
    j++;
    top--;
    st=prec(stack[top]);
   }
    top++;
    stack[top]=exp[i];
   }
 }
    else if(exp[i]==')')
  {
        while(stack[top]!='(')
     {
        post[j]=stack[top];
        j++;
         top--;
      }
    top--;
   }
        else
    {
```

```
            printf("\nInvalid expression");
        }
        i++;
    }
            for(k=top; k>=0; k--)
        {
            post[j]=stack[k];
            j++;
        }  printf("\n Postfix expression is %s",post);
}     int prec(char op)
{
            if(op=='+'||op=='-')
            return 1;
            else if(op=='*'||op=='/')
            return 2;
            else if(op=='^')
            return 3;
            else
            return 0;
}
```

Sorting Techniques:

Sorting is a technique to rearrange the elements of a list in ascending or descending order, which can be numerical, lexicographical, or any user-defined order.

Sorting is a process through which the data is arranged in ascending or descending order. Sorting can be classified in two types;

Internal Sorting:- This method uses only the primary memory during sorting process. All data items are held in main memory and no secondary memory is required this sorting process. If all the data that is to be sorted can be accommodated at a time in memory is called internal sorting.

There is a limitation for internal sorts; they can only process relatively small lists due to memory constraints. There are 3 types of internal sorts.

(i) SELECTION SORT :- Ex:- Selection sort algorithm, Heap Sort algorithm
(ii) INSERTION SORT :- Ex:- Insertion sort algorithm, Shell Sort algorithm
(iii) EXCHANGE SORT :- Ex:- Bubble Sort Algorithm, Quick sort algorithm

External Sorts:- Sorting large amount of data requires external or secondary memory.
This process uses external memory such as HDD, to store the data which is not fit into the main memory. So, primary memory holds the currently being sorted data only. All external sorts are based on process of merging. Different parts of data are sorted separately and merged together.
Ex:- Merge Sort

Bubble Sort:

34 12 31 25 16 10 78 42

Subject: Data Structures

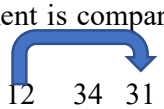
No.of elements=8 i=0, j=i+1 a[i]>a[j] then swap

Step1:

Pass1 : 34 12 31 25 16 10 78 42

34	12	31	25	16	10	78	42
a[0] i	a[1] j	2	3	4	5	6	a[7]

First element is compared with all other elements. i.e., 34 is compared with all other elements. 34>12



12 34 31 25 16 10 78 42

12 is compared with 31, here no swapping required

12 34 31 25 16 10 78 42

12 is compared with 25 , here no swaps required.

12 34 31 25 16 10 78 42

12 is compared with 16, here no swaps required.

12 34 31 25 16 10 78 42

12 is compared with 10, 12>10 then swap

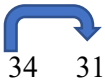
10 34 31 25 16 12 78 42

10 34 31 25 16 12 78 42

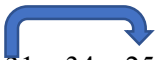
Stop the pass:

Step2:

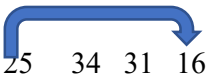
Pass2: in pass2 second is compared with all other elements



10 34 31 25 16 12 78 42



10 31 34 25 16 12 78 42



10 25 34 31 16 12 78 42

10 16 34 31 25 12 78 42

10 12 34 31 25 16 78 42

10 12 34 31 25 16 78 42

End of pass2: 10 12 34 31 25 16 78 42

Prepared By: Suresh Kumar K

Inorder to sort the 8 elements, we need 8 passes.

If n elements are there, here it will take ***n*** passes in bubble sort

Output:

Enter number of elements you want 5

Enter elements: 34 21 56 32 78

Before sorting elements are 34 21 56 32 78

After sorting elements are 21 32 34 56 78

Subject: Data Structures

Insertion sort:

32 25 12 36 10

No.of elemnts= 5

Pass1: 32 25 12 36 10

32>25 then swap

25 32 12 36 10

Pass2:

Sorted Sub list unsorted list

25 32 12 36 10

32>12 then swap

Sorted Sub list unsorted list

25 12 32 36 10

25>12 then swap

12 25 32 36 10

Pass3:

Sorted sub list un sorted list

12 25 32 36 10

32<36 then no swaps required

12 25 32 36 10

Pass4:

12 25 32 36 10

36>10 then swap

Sorted sub list

12 25 32 10 36

12 25 10 32 36

12 10 25 32 36

Prepared By: Suresh Kumar K

Subject: Data Structures

10 12 25 32 36

Sort the following elements using insertion sort.

65 23 12 45 78 43 56 10

I j
0 1

7	3	2	8	6	4	9	5	2	1

Shell Sort:

Shell sort is a highly efficient sorting algorithm and is based on insertion sort algorithm. This algorithm avoids large shifts as in case of insertion sort, if the smaller value is to the far right and has to be moved to the far left.

Interval will be choose from given elements.

Incase if no.of elements is 5 then interval is 5/2

Incase if the no.of elements is ***n*** then interval is ***n/2***

Subject: Data Structures

Progam No:6: Program for implementing Insertion Sort

```
/* C Program to sort an array in ascending order using Insertion Sort */
#include <stdio.h>
void main()
{
   int n, i, j, temp;
   int arr[20];

   printf("Enter number of elements\n");
   scanf("%d", &n);

   printf("Enter %d integers\n", n);
   for (i = 0; i < n; i++)
   {
      scanf("%d", &arr[i]);
   }
   for (i = 1 ; i < n; i++)
   {
            j = i;
         while ( j > 0 && arr[j-1] > arr[j])
         {
            temp     = arr[j];
            arr[j]   = arr[j-1];
            arr[j-1] = temp;
            j--;
         }
   }
   printf("Sorted list in ascending order:\n");
   for (i = 0; i <= n - 1; i++)
   {
      printf("%d\n", arr[i]);
   }
}
```

Subject: Data Structures

Program No:7 Program for implementing Shell Sort.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int a[10], i, j, k,n t;
  printf("Simple Shell Sort Example - Functions and Array\n");
  printf("\n Enter Number of elements you want");
  scanf("%d", &n);
  printf("\nEnter %d Elements for Sorting\n", n);
  for (i = 0; i < n; i++)
    scanf("%d", &a[i]);

  printf("\nBefore sorting Elements are");
  for (i = 0; i < n; i++)
  {
    printf("\t%d", a[i]);
  }

  for (i = n/ 2; i > 0; i--)
{
    for (j = i; j < n; j++)
    {
      for (k = j - i; k >= 0; k = k - i)
        {
                if (a[k + i] >= a[k])
                break;
                else
              {
                 t = a[k];
                a[k] = a[k + i];
                a[k + i] = t;
              }
          }
      }
    }

  printf("\n\nSorted Data :");
  for (i = 0; i < n; i++) {
    printf("\t%d", a[i]);
  }
}
```

Prepared By: Suresh Kumar K

Subject: Data Structures

Radix Sort Algorithm

Radix sort is one of the sorting algorithms used to sort a list of integer numbers in order. In radix sort algorithm, a list of integer numbers will be sorted based on the digits of individual numbers. Sorting is performed from **least significant digit to the most significant digit**. Radix sort algorithm requires the number of passes which are equal to the number of digits present in the largest number among the list of numbers. For example, if the largest number is a 3 digit number then that list is sorted with 3 passes.

Step by Step Process

The Radix sort algorithm is performed using the following steps...

- **Step 1 -** Define 10 queues each representing a bucket for each digit from 0 to 9.
- **Step 2 -** Consider the least significant digit of each number in the list which is to be sorted.
- **Step 3 -** Insert each number into their respective queue based on the least significant digit.
- **Step 4 -** Group all the numbers from queue 0 to queue 9 in the order they have inserted into their respective queues.
- **Step 5 -** Repeat from step 3 based on the next least significant digit.
- **Step 6 -** Repeat from step 2 until all the numbers are grouped based on the most significant digit.

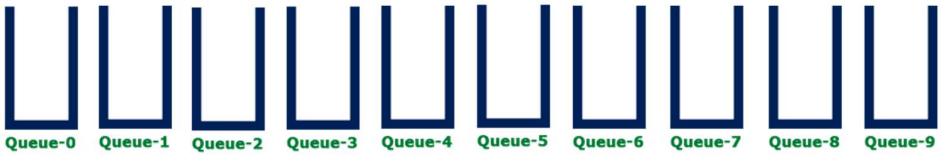
Prepared By: Suresh Kumar K

Subject: Data Structures

Consider the following list of unsorted integer numbers

82, 901, 100, 12, 150, 77, 55 & 23

Step 1 - Define 10 queues each represents a bucket for digits from 0 to 9.



Pass - 1

Step 2 - Insert all the numbers of the list into respective queue based on the Least significant digit (once placed digit) of every number.

8**2**, 90**1**, 10**0**, 1**2**, 15**0**, 7**7**, 5**5** & 2**3**

Group all the numbers from queue-0 to queue-9 inthe order they have inserted & consider the list for next step as input list.

100, 150, 901, 82, 12, 23, 55 & 77

Pass - 2

Step 3 - Insert all the numbers of the list into respective queue based on the next Least significant digit (Tens placed digit) of every number.

1**0**0, 1**5**0, 9**0**1, **8**2, **1**2, **2**3, **5**5 & **7**7

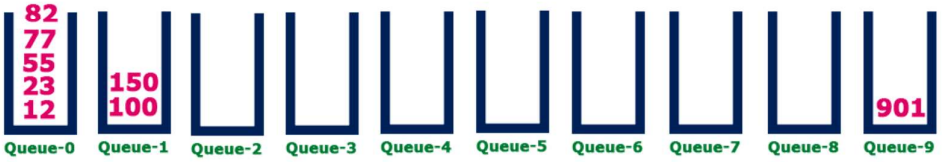
Group all the numbers from queue-0 to queue-9 inthe order they have inserted & consider the list for next step as input list.

100, 901, 12, 23, 150, 55, 77 & 82

Pass - 3

Step 4 - Insert all the numbers of the list into respective queue based on the next Least significant digit (Hundres placed digit) of every number.

100, **9**01, 12, 23, **1**50, 55, 77 & 82



Group all the numbers from queue-0 to queue-9 inthe order they have inserted & consider the list for next step as input list.

12, 23, 55, 77, 82, 100, 150, 901

List got sorted in the increasing order

Subject: Data Structures

Sort the following elements using Radix Sort.

45 234 761 42 100 1234 89 3 76 18 67 90

Sort the following elements using Radix Sort.

-34 45 12 -80 -123 456 67 54 3 10 -12 29

Separate the list into two parts

First part: maintain all negative numbers
Second part: positive numbers

-34 -80 -123 -12 45 12 456 67 54 3 10 29

-12 -34 -80 -123 3 10 12 29 45 54 67 456

(12+3*4-5-(7/8^2)+3-9/4)

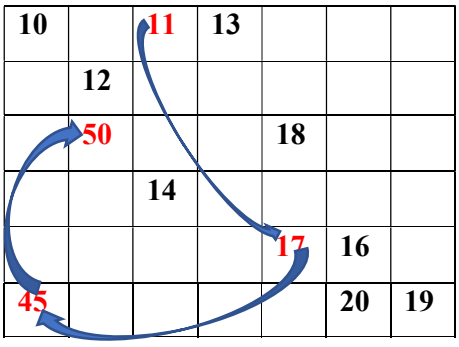
Prepared By: Suresh Kumar K

Subject: Data Structures

Arrays Vs. Linked Lists:

	Array	Linked List
1.	Insertions and deletions are difficult.	Insertions and deletions can be done easily.
2.	It needs movements of elements for insertion and deletion.	It does not need movement of nodes for insertion and deletion.
3.	Space is wasted.	Space is not wasted.
4.	It is a static memory allocation	It is dynamic memory allocation.
5.	It requires less space as only information is stored.	When compare to array,It requires more space as pointers are also stored along with information.
6.	Its size is fixed.	Its size is not fixed.
7.	It cannot be extended or reduced according to requirements.	It can be extended or reduced according to requirements.
8.	Same amount of time is required to access each element.	Different amount of time is required to access each element.
9.	Elements are stored in consecutive memory locations.	Elements may or may not be stored in consecutive memory locations.
10.	If we have to go to a particular element then we can reach there directly.	If we have to go to a particular node then we have to go through all those nodes that come before that node.

Linked List:



Memory= 34

I want to store 34 elements

Linked List : Linked list is a sequence of elements in which each element has a link which points to its next element in the sequence.

Prepared By: Suresh Kumar K

(Or)

Linked list is a linear data structure that contains sequence of elements such that each element maintains a link (pointer) which points to its next element. Each element in a linked list is called as **"Node".**