STES's

NBN SINHGAD SCHOOL OF ENGINEERING, AMBEGAON (BK), PUNE

Department of Computer Engineering

Sinhgad Institutes

LABORATORY MANUAL

(2020-21)

DATA STRUCTURES LABORATORY

SE-COMPUTER ENGINEERING

SEMESTER-I

Subject Code: 210246

TEACHING SCHEME	EXAMINATION SCHEME
Lectures: 3 Hrs/Week	End-Sem: 70 Marks
Practical: 4 Hrs/Week	Mid-Sem: 30 Marks
	Practical: 50 Marks
	Term Work: 25 Marks

-: **Name of Faculty** :-

Ms. N. K. Kadale

Asst. Professor

Department of Computer Engineering,

NBN Sinhgad School of Engineering, Ambeagon (Bk), Pune

GROUP A : ASSIGNMENTS

Assignment No.:	01 (GROUP A)
Title:	Store marks of students using array in python
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 01 (Group A)

Title : Store marks of students using array in python.

Objectives :
a) To understand the concept of data structure
b) To understand linear and static data structure
c) To understand array data structure
d) To understand array as an ADT

Problem Statement : Write a Python program to store marks scored in subject "Fundamental of Data Structure" by N students in the class. Write functions to compute following:
a)The average score of class
b)Highest score and lowest score of class
c)Count of students who were absent for the test
d)Display mark with highest frequency

Outcomes:
- Understanding the need of data structure
- Understanding the static data structure, array.
- Understanding the linear data structure, array.

Software Requirements: Open Source Python, Programming tool like Jupyter Notebook, Pycharm, Spyder (64 bit)

Theory : Data structure is:
- A representation of data and the operations allowed on that data or
- A way in which sets of data are organized in a particular system or
- A computer interpretable format used for storing, accessing, transferring, and archiving data.

Data structures are categorized into two types

(a) Linear : Elements in linear data structure form a sequence. A linear data structure traverses the data elements sequentially, in which only one data element can directly be reached. Eg: Arrays, Stacks, Queues, Linked Lists.

(b) Non–Linear : If the elements of data structure do not form a sequence or a linear list then data structure is called as Non–Linear data structure. eg. Trees , Graphs.

ARRAY:

Array is defined as a collection of elements of same data type.
Hence, it is also called as homogeneous collection of data items.

An array:
- is an indexed sequence of components.
- component has a fixed, unique index.
- typically, occupies sequential storage locations.
- length is determined when the array is created, and cannot be changed (static data structure).
- has direct access to each element by specifying its position.

Position of element in an array defines index or subscript of particular element. Array

index always starts with zero. If the size of array is n then index ranges from 0 to n–1. Array index must be integer or integer expression. There can be array of integer, floating point or character values. Character array is called as String.

Syntax: type arrayName [arraySize];

Index	0	1	2	3	4
Value	60	20	85	35	75

e.g.

- int data[5]; data[0]= 60; data[1]=20; data[2]=85; data[3]=35; data[4]=75; or
- int data[5]={60, 20, 85, 35, 75}; or
- int data[]= {60, 20, 85, 35, 75};

in above example array is declared and initialized too.

Array initialization can only be used in a declaration.

An array is an Abstract Data Type

- The array type has a set of values
  - The values are all the possible arrays
- The array type has a set of operations that can be applied uniformly to each of these values
  - The only operation is indexing
  - Alternatively, this can be viewed as two operations:
    - inspecting an indexed location
    - storing into an indexed location

It's abstract because the implementation is hidden: all access is via the defined operations

Applications:

1. Arrays are used to implement other data structures, such as heaps, hash tables, deques, queues, stacks, strings, and VLists.
2. Arrays are used to implement mathematical vectors and matrices, as well as other kinds of rectangular tables.
3. Many databases, small and large, consist of (or include) one-dimensional arrays whose elements are records.
4. One or more large arrays are sometimes used to emulate in-program dynamic memory allocation, particularly memory pool allocation.
5. Arrays can be used to determine partial or complete control flow in programs, as a compact alternative to (otherwise repetitive) multiple IF statements.

Algorithm: **Algorithm to accept 10 student's marks:**

1. Start
2. Read the marks of 10 students and stored in array (-1 for absent student)
3. Calculate and print the average of 10 student's marks
4. Comparing all student's marks print highest marks
5. Comparing all student's marks print lowest marks
6. Comparing all student's marks print mark scored by most of the students
7. Comparing -1 in an array, print absent students
8. Stop

Algorithm for highest mark in an array:

```
max=-1
for i= 1:n;
        if a[i] >max and a[i] is not absent
            max=a[i]
end
```

Algorithm for lowest mark in an array:

```
min=101
for i= 1:n;
        if a[i] <min and a[i] is not absent
            min=a[i]
end
```

Algorithm for mark scored by most of the students:

```
fcount=0
for i= 1:n;
        count=0
        for j= 1:n;
                ifa[i]==a[j] then
                        count++;
        if fcount < count then
                fcount=count
                most=a[j]
        end
end
```

Algorithm for calculating absent students:

```
count=0
for i= 1:n;
        if a[i] == -1 then
            count++
end
```

Test Cases: Input: Enter 10 students marks: 40 45 50 41 60 45 70 71 45 73
Output: The average score of class: 54
Highest score of class: 73
Lowest score of class: 40
Marks scored by most of the students: 45
Absent for the test:

Input: Enter 10 students marks: 55 -1 50 65 50 35 60 50 52 45
Output: The average score of class: 51.33
Highest score of class: 35
Lowest score of class: 55
Marks scored by most of the students: 50
Absent for the test: 1

Conclusion : Thus, we implemented linear-static data structure (array) by storing marks of students in an array.

Questions :

1) What is data structure?
2) What is linear data structure?
3) What is static data structure?
4) What is an array?
5) Explain array as an ADT.
6) What are the applications of an array?
7) What is multidimensional array?

Assignment No.:	02 (GROUP A)
Title:	To perform various set operations in python.
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 02 (Group A)

Title : To perform various set operations in python.

Objectives :
a) To understand the concept of array.
b) To understand various set operation using array and function.
c) To understand use of array data structure for set operations.

Problem Statement : In second year computer engineering class, group A student's play cricket, group B students play badminton and group C students play football. Write a Python program using functions to compute following: -

a)List of students who play both cricket and badminton
b)List of students who play either cricket or badminton but not both
c)Number of students who play neither cricket nor badminton
d)Number of students who play cricket and football but not badminton.

Outcomes:
- Understanding the concept of array.
- Understanding the use of functions for set operations.
- Understanding the use of array data structure for set operations.

Software Requirements: Open Source Python, Programming tool like Jupyter Notebook, Pycharm, Spyder (64 bit)

Theory:

Set:

Definition: A set is any collection of definite, distinguishable objects, and we call these objects the elements of the set.

To represents a set, array data structure is used.

Array:

Array is a data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations.

In single dimensional array we can represent all set which is required to perform various set operations.

Operations on set:

Definition: Let A and B be subsets of a set U.

1. **Union of A and B**: $A \cup B = \{x \in U \mid x \in A \text{ or } x \in B\}$
   Union of the sets *A* and *B*, denoted $A \cup B$, is the set of all objects that are a member of *A*, or *B*, or both. The union of {1, 2, 3} and {2, 3, 4} is the set {1, 2, 3, 4}.
   We can represents the above set in terms of programming using array data structure

as follows:
A[ ]={1,2,3} and B[ ]={2, 3, 4}
Suppose C is set for storing union result.
So therefore union is C [ ] = {1, 2, 3, 4}.

2. **Intersection of A and B**: $A \cap B = \{x \in U \mid x \in A \text{ and } x \in B\}$
   Intersection of the sets *A* and *B*, denoted $A \cap B$, is the set of all objects that are members of both *A* and *B*. The intersection of {1, 2, 3} and {2, 3, 4} is the set {2, 3} .
   Intersection using array:
   A[ ]={1,2,3} and B[ ]={2, 3, 4}
   Suppose C is set for intersection union result.
   So therefore intersection is C [ ] = {2, 3}.

3. **Difference of B minus A**: $B-A = \{x \in U \mid x \in B \text{ and } x \notin A\}$
   The difference of two sets B and A is the collection of objects in B that are not in A . The difference is written B – A.

4. **Complement of A**: $A^c = \{x \in U \mid x \notin A\}$
   The compliment of a set A is the collection of objects in the universal set that are not in A.

Algorithm: **Algorithm for union of two set:**

Algorithm Union(arr1[ ], arr2[ ])
For union of two arrays, follow the following procedure.
1) Use two index variables i and j, initial values i = 0, j = 0
2) If arr1[i] is smaller than arr2[j] then print arr1[i] and increment i.
3) If arr1[i] is greater than arr2[j] then print arr2[j] and increment j.
4) If both are same then print any of them and increment both i and j.
5) Print remaining elements of the larger array.

Algorithm for Intersection of two set:

Algorithm Intersection(arr1[ ], arr2[ ])
For Intersection of two arrays, print the element only if the element is present in both arrays.
1) Use two index variables i and j, initial values i = 0, j = 0
2) If arr1[i] is smaller than arr2[j] then increment i.
3) If arr1[i] is greater than arr2[j] then increment j.
4) If both are same then print any of them and increment both i and j.

Algorithm for to find the difference of two set:

sort arrays A and B
result will be in C
let a - the first elem of A
let b - the first elem of B

then:
1) while a < b: insert a into C and a = next elem of A
2) while a > b: b = next elem of B
3) if a = b: a = next elem of A and b = next elem of B
4) if b goes to end: insert rest of A into C and stop
5) if a goes to end: stop.

Algorithm to find out complement of two set:

1) If A[i]!=U[i] then save A[i] in C[k] and increment k

   Otherwise check next element from A[i].

2) Stop when i=n, where n is total number of element of A[i].

Test Cases: Various operation of set is tested.
Union operation :
Set A : student who are playing cricket
11 12 13 15
Set B: student who are playing badminton
15 16 17 18
Union set:
11 12 13 15 16 17 18.

Conclusion : Thus, we implemented various set operations using array data structure and function.

Questions :

1) What is array?
2) Explain various operation of set using array.

Assignment No.:	03 (GROUP A)
Title:	Python program that computes the net amount of a bank account based a transaction log from console input.
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 03 (Group A)

Title : Python program that computes the net amount of a bank account based a transaction log from console input.

Objectives :

a) To understand the concept of data structure
b) To understand linear and static data structure
c) To understand array data structure

Problem Statement : Write a Python program that computes the net amount of a bank account based a transaction log from console input. The transaction log format is shown as following: D 100 W 200 (Withdrawal is not allowed if balance is going negative. Write functions for withdraw and deposit) D means deposit while W means withdrawal.

Outcomes:

- Understanding the need of data structure
- Understanding the static data structure, array.
- Understanding the linear data structure, array.

Software Requirements: Open Source Python, Programming tool like Jupyter Notebook, Pycharm, Spyder (64 bit)

Theory:

ARRAY:

Array is defined as a collection of elements of same data type.
Hence, it is also called as homogeneous collection of data items.

An array:

- is an indexed sequence of components.
- component has a fixed, unique index.
- typically, occupies sequential storage locations.
- length is determined when the array is created, and cannot be changed (static data structure).
- has direct access to each element by specifying its position.

Position of element in an array defines index or subscript of particular element. Array index always starts with zero. If the size of array is n then index ranges from 0 to n–1. Array index must be integer or integer expression. There can be array of integer, floating point or character values. Character array is called as String.

Syntax: type arrayName [arraySize];

Index	0	1	2	3	4
Value	60	20	85	35	75

e.g.

- int data[5]; data[0]= 60; data[1]=20; data[2]=85; data[3]=35; data[4]=75; or
- int data[5]={60, 20, 85, 35, 75}; or
- int data[]= {60, 20, 85, 35, 75};

in above example array is declared and initialized too.

Array initialization can only be used in a declaration.

Algorithm: **Algorithm to computes the net amount of a bank account based a transaction log :**

1. Start
2. Read type of transaction and amount.
3. If it deposit type of transaction then add it in balance
4. Else if it is withdrawal operation then decrease balance
5. Display total balance
6. Stop.

Test Cases: Suppose the following input is supplied to the program:
D 300
D 300
W 200
D 100
Then, the output should be: 500

Conclusion : Thus, we implemented linear-static data structure (array) by computing the net amount of a bank account based a transaction log

Questions :

1) What is array?
2) Explain declaration and initialization of array.

GROUP B : ASSIGNMENTS

Assignment No.:	01 (GROUP B)
Title:	Implementation of Selection sort and bubble sort in python.
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 01 (Group B)

Title : Implementation of Selection sort and bubble sort in python.

Objectives :
a) To understand the concept of sorting.
b) To understand the concept and use of selection sort.
c) To understand the concept and use of bubble sort.

Problem Statement : Write a Python program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using
a) Selection Sort
b) Bubble sort and display top five scores.

Outcomes:
- Understanding the concept of sorting.
- Understanding the concept and use of selection sort.
- Understanding the concept and use of bubble sort.

Software Requirements: Open Source Python, Programming tool like Jupyter Notebook, Pycharm, Spyder (64bit)

Theory:

Selection Sort:

Selection sort carries out a sequence of passes over the table. At the first pass an entry is selected on some criteria and placed in the correct position in the table. The possible criteria for selecting an element are to pick the smallest or pick the largest. If the smallest is chosen then, for sorting in ascending order, the correct position to put it is at the beginning of the table.

Now that the correct entry is in the first place in the table the process is repeated on the remaining entries. Once this has been repeated *n*-1 times the *n*-1 smallest entries are in the first *n*-1 places which leaves the largest element in the last place. Thus only *n*-1 passes are required.

The pseudocode can be described as follows:

```
procedure selection sort
  list  : array of items
  n     : size of list
  for i = 1 to n - 1
  /* set current element as minimum*/
    min = i
    /* check the element to be minimum */
    for j = i+1 to n
      if list[j] < list[min] then
        min = j;
      end if
    end for
    /* swap the minimum element with the current element*/
    if indexMin != i  then
      swap list[min] and list[i]
    end if
  end for
```

end procedure

For example consider the following example of selection sort being carried out on a sample set of data:

9 2 5 7 4 8 on pass 1 look for smallest in 1st to 6th swap 2nd with first giving
2 9 5 7 4 8 on pass 2 look for smallest in 2nd to 6th swap 5th with second giving
2 4 5 7 9 8 on pass 3 look for smallest in 3rd to 6th swap 3rd with third giving
2 4 5 7 9 8 on pass 4 look for smallest in 4th to 6th swap 4th with fourth giving
2 4 5 7 9 8 on pass 5 look for smallest in 5th to 6th swap 5th with 6th giving
2 4 5 7 8 9 sorted.

Bubble Sort:

Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

The pseudocode can be described as follows:

```
procedure bubbleSort( list : array of items )
  loop = list.count;
  for i = 0 to loop-1 do:
    swapped = false
    for j = 0 to loop-1 do:
      /* compare the adjacent elements */
      if list[j] > list[j+1] then
        /* swap them */
        swap( list[j], list[j+1] )
        swapped = true
      end if
    end for
    /*if no number was swapped that means array is sorted now, break the loop.*/
    if(not swapped) then
      break
    end if
  end for
end procedure return list
```

Example:

First Pass:

(**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.
(1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4
(1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2
(1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(**1 4** 2 5 8) –> (**1 4** 2 5 8)
(1 **4 2** 5 8) –> (1 **2 4** 5 8), Swap since 4 > 2
(1 2 **4 5** 8) –> (1 2 **4 5** 8)

(1 2 4 **5 8**) –> (1 2 4 **5 8**)
Now, the array is already sorted, but algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:
(**1 2** 4 5 8) –> (**1 2** 4 5 8)
(1 **2 4** 5 8) –> (1 **2 4** 5 8)
(1 2 **4 5** 8) –> (1 2 **4 5** 8)
(1 2 4 **5 8**) –> (1 2 4 **5 8**)

After final pass will get the sorted list as : (1 2 4 **5 8**)

Algorithm: **Algorithm for Selection Sort:**
Step 1 – Set MIN to location 0
Step 2 – Search the minimum element in the list
Step 3 – Swap with value at location MIN
Step 4 – Increment MIN to point to next element
Step 5 – Repeat until list is sorted

Algorithm for Bubble Sort:
Step 1 – begin BubbleSort(list)
Step 2 – for all elements of list
Step 3 – if list[i] > list[i+1] then swap(list[i], list[i+1])
Step 4 – display list

Test Cases: **Input** :Unsorted Array: [65 35 45 15 85 5 95 25]
Output :Sorted Array: [5 15 25 35 45 65 85 95]

Conclusion : Thus, we implemented selection sort and bubble sort to sort the given array in ascending order.

Questions :
1) What is sorting?
2) Why we need sorting?
3) Explain best, average, worst case analysis for selection and bubble sort.

Assignment No.:	02 (GROUP B)
Title:	Implementation of insertion sort and shell sort in python.
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 02 (Group B)

Title : Implementation of insertion sort and shell sort in python.

Objectives :
a) To understand the insertion sort.
b) To understand the shell sort.

Problem Statement : Write a Pythonprogram to store second year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using :
a) Insertion sort
b) Shell Sort and display top five scores.

Outcomes:
- Understanding insertion sort.
- Understanding shell sort.

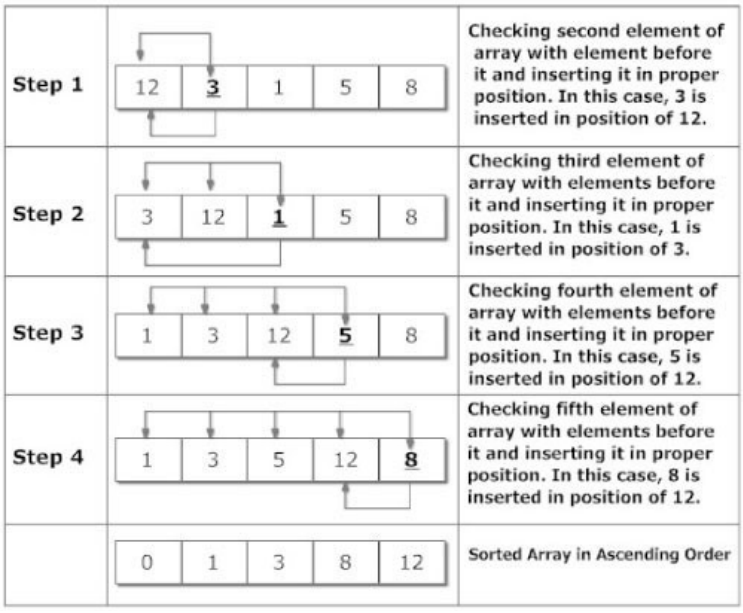
Software Requirements: Open Source Python, Programming tool like Jupyter Notebook, Pycharm, Spyder (64bit)

Theory:

Insertion Sort:

This is a in-place comparison based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. A element which is to be 'inserted' in this sorted sub-list, has to find its appropriate place and insert it there. Hence the name insertion sort.

The array is searched sequentially and unsorted items are moved and inserted into sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n2) where n are no. of items.



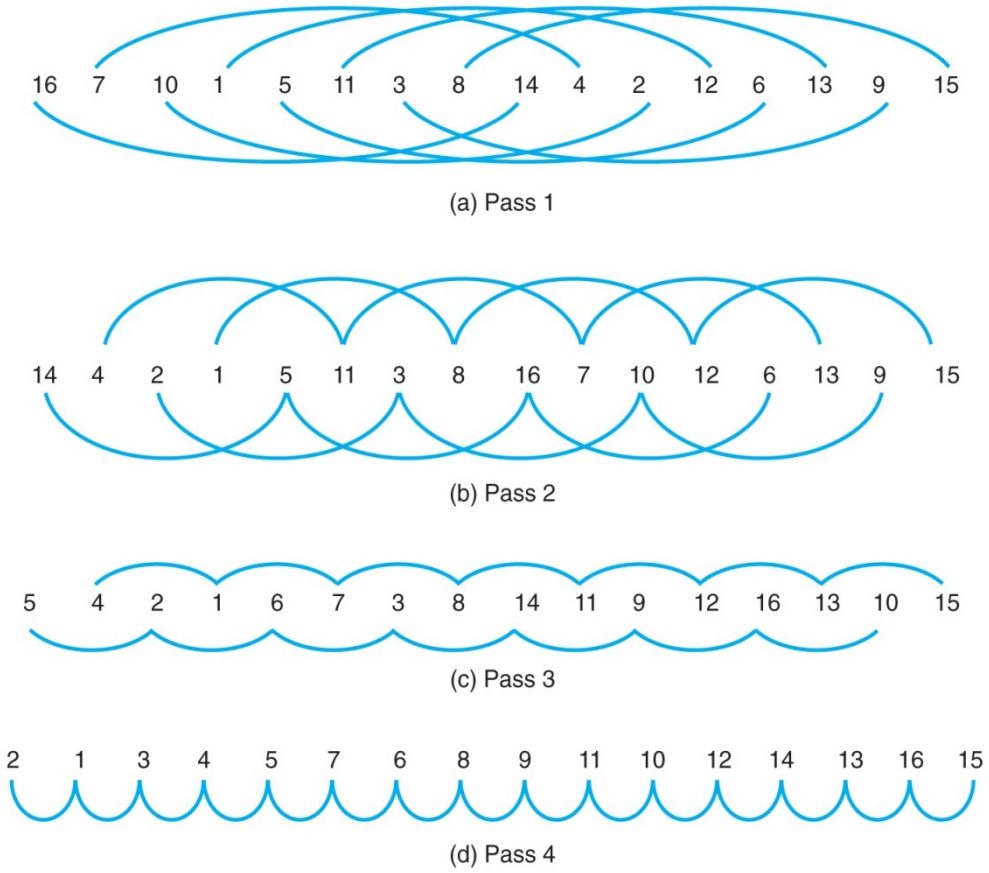
Insertion sort

Shell Sort:

The shell sort, improves on the insertion sort by breaking the original list into a number of smaller sublists, each of which is sorted using an insertion sort. The unique way that these sublists are chosen is the key to the shell sort. Instead of breaking

the list into sublists of contiguous items, the shell sort uses an increment i, sometimes called the gap, to create a sublist by choosing all items that are i items apart.

The running time of Shellsort is heavily dependent on the gap sequence it uses. For many practical variants, determining their time complexity remains an open problem.



Shell Sort iterations

Algorithm: **Algorithm for Insertion Sort:**

Step 1 – If it is the first element, it is already sorted. return 1;
Step 2 – Pick next element
Step 3 – Compare with all elements in the sorted sub-list
Step 4 – Shift all the elements in the sorted sub-list that is greater than the value to be sorted
Step 5 – Insert the value
Step 6 – Repeat until list is sorted

Algorithm for Shell Sort:

Step 1 – Initialize the value of h
Step 2 – Divide the list into smaller sub-list of equal interval h
Step 3 – Sort these sub-lists using insertion sort
Step 3 – Repeat until complete list is sorted

Test Cases: **Input** :Unsorted Array: [65 35 45 15 85 5 95 25]
Output :Sorted Array: [5 15 25 35 45 65 85 95]

Conclusion : Thus, we implemented Insertion sort and Shell sort to sort the given array in ascending order.

Questions :

1) What is stability in sorting? Which sorting algorithms are not stable?
2) What is 'gap' in shell sort?
3) What are the time complexities of insertion and shell sort?

Assignment No.:	03 (GROUP B)
Title:	Implementation of quick sort in python.
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 03 (Group B)

Title : Implementation of quick sort in python.

Objectives :
a) To understand quick sort
b) To understand divide and conquer

Problem Statement : Write a Python program to store first year percentage of students in array. Write function for sorting array of floating point numbers in ascending order using quick sort and display top five scores.

Outcomes:
- Understanding quicksort
- Understanding divide and conquer

Software Requirements: Open Source Python, Programming tool like Jupyter Notebook, Pycharm, Spyder (64bit)

Theory:

Quick Sort:

The basic version of quick sort algorithm was invented by C. A. R. Hoare in 1960 and formally introduced quick sort in 1962. It is based on the principle of divide-and-conquer. It has two phases:

- the partition phase
- the sort phase

The quick sort partitions an array and then calls itself recursively twice to sort the resulting two subarray. This algorithm is quite efficient for large sized data sets as its average and worst case complexity are of O(nlogn) where n are no. of items.

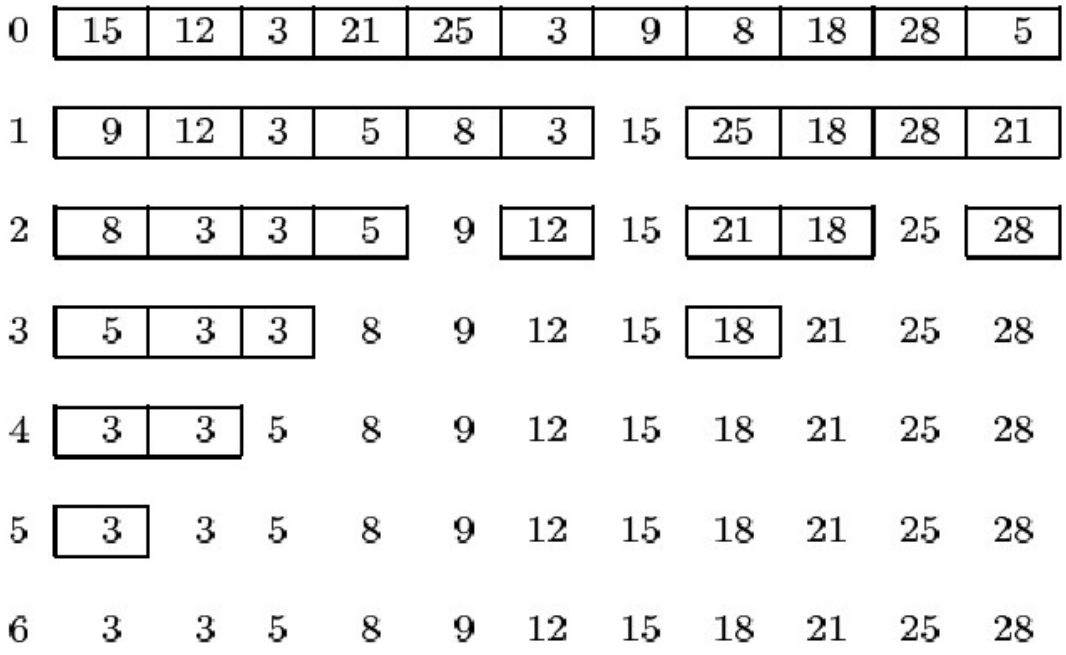
Most of the work is done in the partition phase - it works out where to divide the work. The sort phase simply sorts the two smaller problems that are generated in the partition phase.

Pseudocode:

```
quicksort(A, i, k):
  if i < k:
    p := partition(A, i, k)
    quicksort(A, i, p - 1)
    quicksort(A, p + 1, k)

// left is the index of the leftmost element of the subarray
// right is the index of the rightmost element of the subarray
(inclusive)
// number of elements in subarray = right-left+1
 partition(array, left, right)
    pivotIndex := choosePivot(array, left, right)
    pivotValue := array[pivotIndex]
    swap array[pivotIndex] and array[right]
    storeIndex := left
    for i from left to right - 1
        if array[i] < pivotValue
            swap array[i] and array[storeIndex]
            storeIndex := storeIndex + 1
    swap array[storeIndex] and array[right]  // Move pivot to its
```

```
final place
    return storeIndex
```



Iterations showing quicksort implementation

This makes Quicksort a good example of the divide and conquers strategy for solving problems. This is similar in principle to the binary search. In the quicksort, we divide the array of items to be sorted into two partitions and then call the quicksort procedure recursively to sort the two partitions, i.e. we divide the problem into two smaller ones and conquer by solving the smaller ones.

Algorithm: **Algorithm for Quick Sort:**

Step 1 – Choose the lowsest index value has pivot
Step 2 – Take two variables to point left and right of the list excluding pivot
Step 3 – left points to the low index
Step 4 – right points to the high
Step 5 – while value at left is less than pivot move right
Step 6 – while value at right is greater than pivot move left
Step 7 – if both step 5 and step 6 does not match swap left and right
Step 8 – if left ≥ right, the point where they met is new pivots

Test Cases: **Input** :Unsorted Array: [65 35 45 15 85 5 95 25]
Output :Sorted Array: [5 15 25 35 45 65 85 95]

Conclusion : Thus, we implemented quick sort the given array in ascending order.

Questions :

1) What is divide and conquer algorithmic strategy?
2) What is internal and external sorting? give example.
3) Compare all sorting algorithms and comment about best sorting algorithm.

GROUP C : ASSIGNMENTS

Assignment No.:	01 (GROUP C)
Title:	Student Club through SLL in C++.
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 01 (Group C)

Title : Student Club through SLL in C++.

Objectives :

a) To understand the concept of linked list data structure
b) To understand dynamic memory allocation
c) To understand types of linked list
d) To understand different operations on singly liked list(SLL).

Problem Statement : Department of Computer Engineering has student's club named 'Pinnacle Club'. Students of second, third and final year of department can be granted membership on request. Similarly one may cancel the membership of club. First node is reserved for president of club and last node is reserved for secretary of club. Write C++ program to maintain club member's information using singly linked list. Store student PRN and Name. Write functions to:

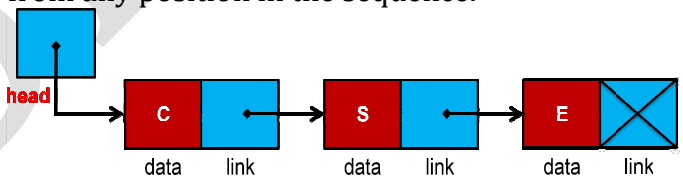
1. Add and delete the members as well as president or even secretary.
2. Compute total number of members of club
3. Display members
4. Two linked lists exist for two divisions. Concatenate two lists.

Outcomes:

- Understanding the linked list data structure
- Understanding the dynamic memory allocation.
- Understanding different operations on SLL.

Software Requirements: g++ compiler on Ubuntu 14.04 (64bit)

Theory : **Linked List:** A linked list is a linear data structure consisting of a group of nodes which together represent a sequence. Under the simplest form, each node is composed of a data and a reference (in other words, a link) to the next node in the sequence; more complex variants add additional links. This structure allows for efficient insertion or removal of elements from any position in the sequence.



Each element (node) of a list comprises of two items - the data and a reference/link to the next node. The last node has a reference to NULL. The entry point into a linked list is called the head of the list. It should be noted that head is not a separate node, but the reference to the first node. If the list is empty then the head is a NULL reference.

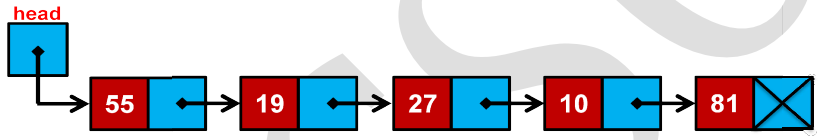
Difference between array and linked list:

Array	Linked List:
a) Arrays are linear data structures (for accessing and for storing in memory).	a) Linked lists are linear for accessing, and non-linear for storing in memory.
b) Size of array is fixed.	b) Size of linked list is not fixed.

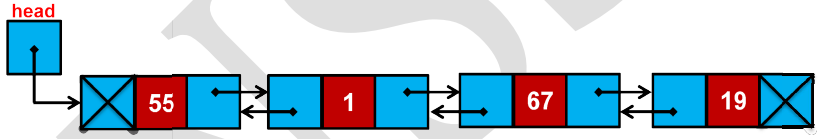
c) Static data structure.	c) Dynamic data structure.
d) Array elements cannot be added, deleted once it is declared.	d) The nodes in the linked list can be added and deleted from the list.
e) Array elements can be modified easily by identifying the index value.	e) It is a complex process for modifying the node in a linked list.
f) Array are not difficulty to sort.	f) Linked list are very difficult to sort.
g) Easy to locate the desired element.	g) Can not immediately locate the desires element. Need to traverse the whole list to reach that element.
h) Memory allocation at the compilation time.	h) Memory allocation at the run time.
i) Basically two types are there:- one dimension , two dimension.	i) Basically 3 types are there:- singly, doubly and circular.

Types of linked list:

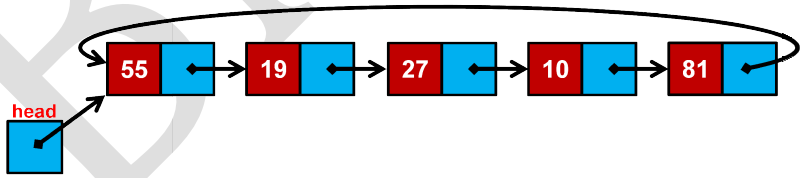
a) **Singly Linked List**: Item Navigation is forward only.



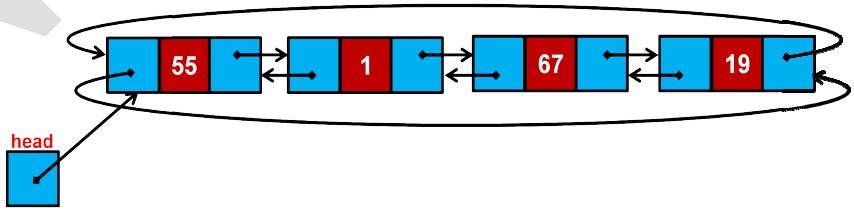
b) **Doubly Linked List**: Items can be navigated forward and backward way.



c) **Circular Linked List**: Last item contains link of the first element as next (in SLL and DLL) and first element has link to last element as prev (in DLL).



Circular Singly Linked List



Circular Doubly Linked List

Dynamic Memory:

In the programs, memory needs were determined before program

execution by defining the variables needed. But there may be cases where the memory needs of a program can only be determined *during runtime*. For example, when the memory needed depends on **user input**. On these cases, programs need to *dynamically allocate memory*, for which the **C++ language** integrates the operators **new** and **delete**.

Operator ***new*** **:**
Dynamic memory is allocated using operator *new*. *new* is followed by a data type specifier and, if a sequence of more than one element is required, the number of these within brackets [ ]. It returns a pointer to the beginning of the new block of memory allocated. Its syntax is:

Syntax:	*Example:*
pointer = new type; **or** pointer=new type[number_of_elements]	*int* * ptr; ptr = *new int* [5];

In this case, the system dynamically allocates space for five elements of type int and returns a pointer to the first element of the sequence, which is assigned to ptr (an integer pointer). Therefore, ptr now points to a valid block of memory with space for five elements of type int.

Operator ***delete*****:**
In most cases, memory allocated dynamically is only needed during specific periods of time within a program; once it is no longer needed, it can be freed so that the memory becomes available again for other requests of dynamic memory. This is the purpose of operator delete, whose syntax is:

Syntax:	*Example:*
delete pointer; **or** *delete*[] pointer;	*delete ptr; //if ptr is pointer to variable* ***or*** *delete[] ptr; //if ptr is pointer to array*

Algorithm: **Structure:**

```
1. struct node
2. {
3.   int iPRN, iYear;
4.   char cName[30],cPost;
5.   struct node *next;
6. };
```

Algorithm to insert element in Singly Linked List:

```
1. void funAddMem()
2. {
3.      temp=new struct node;
4.      accept PRN, name, post(designation), year(division) in new node;
5.      set new node's link(next) to NULL;
6.
7.       if(linked list is empty)
```

```
8.      {
9.         set head to temp node;
10.     }
11.     else
12.     {
13.       if(temp's post is president) //new node's post is president, make it 1st node
14.       {
15.           set temp's next to head's next;
16.           set head to temp node;
17.       }
18.       else                  //new node's post may be secretary or member
19.       {
20.           trav=head; prev=NULL;
21.           while(trav->next != NULL)
22.           {
23.                prev=trav;
24.                trav=trav->next;
25.           }
26.           if(temp's post is secretary)//new node's post is secretary make it last
27.           {
28.                set trav's next to temp;
29.           }
30.           else                        //new node's post is member
31.           {
32.                if(trav's post is secretary) //if SLL's last node's post is secretary
33.                {
34.                     if(prev==NULL) //SLL has only one node i.e. secretary
35.                     {
36.                          set temp's next to trav;
37.                          set head to temp node;
38.                     }
39.                     else          //SLL has at least two node & last is secretary
40.                     {
41.                          set temp's next to trav;
42.                          set prev's next to temp;
43.                     }
44.                }
45.                else // if SLL's last node's post is member
46.                {
47.                     set trav's next to temp;
48.                }
49.           }
50.      }
51.     }
52. }
```

Algorithm to Compute total number of members of club:

```
1. void compMember()
2. {
3.    trav=head;
4.    int pcount=0,scount=0,mcount=0;
5.    if(trav!=NULL)   //whether SLL is empty
6.    {
7.       while(trav!=NULL) //count SLL till last node
8.       {
9.              if(trav's post is president)
10.                   pcount++;
11.            else if(trav's post is secretary)
```

```
12.                     scount++;
13.              else
14.                     mcount++;
15.
16.              trav=trav->next;
17.       }
18.
19.       print pcount;
20.       print scount;
21.       print mcount;
22.       print total as pcount + scount + mcount;
23.     }
24.     else
25.     {
26.        print SLL is empty;
27.     }
28.  }
```

Algorithm to display Singly Linked List:

```
1. void displaySLL()
2. {
3.     trav=head;
4.     if(trav!=NULL)    //whether SLL is empty
5.     {
6.        while(trav!=NULL)  //prints SLL till last node
7.        {
8.             print PRN, name, post and year of student;
9.             trav=trav->next;
10.       }
11.    }
12.    else
13.    {
14.       print SLL is empty;
15.    }
16.  }
```

Algorithm to delete a member/president /secretary of club:

```
1. void delSLL(char delmem)
2. {
3.       if(SLL is empty)    //check whether SLL is present
4.       {
5.          return;
6.       }
7.
8.       if(delmem is president and head's post is president)
9.       {                        //delete if president node is present in SLL
10.           temp=head;
11.           head=head->next;
12.           delete temp;
13.           return;
14.       }
15.
16.       if(delmem is secretary)
17.       {                    //function to delete secretary node
53.             trav=head; prev=NULL;
54.             while(trav->next!= NULL) //reach to last node
55.             {
```

```
56.                 prev=trav;
57.                 trav=trav->next;
58.           }
18.
19.           if(trav's post is secretary) //if last node is secretary then only delete
20.           {
21.                 if(prev==NULL)
22.                 {
23.                       head=NULL;
24.                       delete trav;
25.                 }
26.                 else
27.                 {
28.                      prev->next=trav->next;
29.                      delete trav;
30.                 }
31.                 return;
32.           }
33.     }
34.
35.     if(delmem is member) //delete if user asked to delete a member node
36.     {
37.           accept delname(name of member to be delted) from user;
59.           trav=head; prev=NULL;
60.           while(trav!= NULL && trav->post=='m' && strcmp(trav-
   >name,delname)!=0)
61.           {                          //find node to be deleted is member and name matches
62.                 prev=trav;
63.                 trav=trav->next;
64.           }
38.
39.           if(trav->post=='m' && prev==NULL)
40.           {                                  //member node to be deleted is a first node
41.                 head=head->next;
42.                 free trav;
43.                 return;
44.           }
45.           else if(trav->post=='m')
46.           {                                //member node to be deleted is not a first node
47.
48.                 prev->next=trav->next;
49.                 free trav;
50.                 return;
51.           }
52.     }
53.     print President/Secretary/Member to be deleted not present;
54.  }
```

Time Complexity:

Insert / Delete at beginning:	O(1)
Insert / Delete at end:	O(n)
Search:	O(n)
Indexing:	O(n)

Test Cases: A menu driven program asking for given inputs:
Input: Enter Student's PRN, Name, Year, Post

- If student post is 'president' put as first node in SLL (though SLL is empty or not).

- If student post is 'secretary' put as last node in SLL (if SLL is empty then its first node).
- If student post is 'member' put (as first node if SLL is empty or only secretary node is in SLL / as last if secretary node absent / second last if secretary present).

Output:
Display student list (President, Member(s), Secretary)
Display reverse student list (Secretary, Member(s), President)

Input: Enter Post to be deleted

- If 'p' then delete president from SLL (If president present).
- If 's' then delete secretary from SLL (If secretary present).
- If 'm' then ask name, then delete member with given name from SLL (If member with given name present).

Output:
Display student list (if deletion is successful)
or
President/Secretary/Member to be deleted not present in SLL

Conclusion : Thus, we implemented linear-dynamic data structure singly linked list to store students/members of student's club named 'Pinnacle Club'.

Questions :

1) What is the difference between static and dynamic data structure?
2) Differentiate SLL and DLL?
3) What are the advantages of Circular LL over Linear LL?

Assignment No.:	02 (GROUP C)
Title:	Two set of linked list for Vanilla and butterscotch in C++.
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 02 (Group B)

Title : Two set of linked list for Vanilla and butterscotch in C++.

Objectives :
a) To understand the concept of linked list data structure.
b) To understand different operations on singly liked list (SLL).

Problem Statement : Second year Computer Engineering class, set A of students like Vanilla Ice-cream and set B of students like butterscotch ice-cream. Write C++ program to store two sets using linked list. compute and display :
Write C/C++ program to store two sets using linked list, compute and display:
a. Set of students who like either vanilla or butterscotch or both
b. Set of students who like both vanilla and butterscotch
c. Number of students who like neither vanilla nor butterscotch

Outcomes:
- Understanding the linked list data structure
- Understanding different operations on SLL.

Software Requirements: g++ compiler on Ubuntu 14.04 (64bit)

Theory :

Operations on set:

Definition: Let A and B be subsets of a set U.

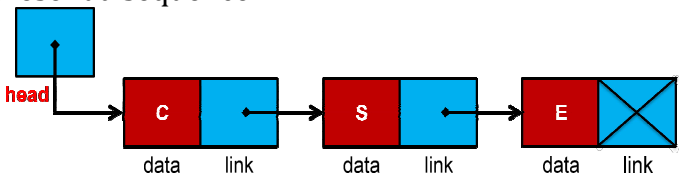
Union of A and B: $A \cup B = \{x \in U \mid x \in A \text{ or } x \in B\}$
Union of the sets *A* and *B*, denoted $A \cup B$, is the set of all objects that are a member of *A*, or *B*, or both. The union of {1, 2, 3} and {2, 3, 4} is the set {1, 2, 3, 4}.
We can represents the above set in terms of programming using array data structure as follows:
A[ ]={1,2,3} and B[ ]={2, 3, 4}
Suppose C is set for storing union result.
So therefore union is C [ ] = {1, 2, 3, 4}.

Intersection of A and B: $A \cap B = \{x \in U \mid x \in A \text{ and } x \in B\}$
Intersection of the sets *A* and *B*, denoted $A \cap B$, is the set of all objects that are members of both *A* and *B*. The intersection of {1, 2, 3} and {2, 3, 4} is the set {2, 3} .
Intersection using array:
A[ ]={1,2,3} and B[ ]={2, 3, 4}
Suppose C is set for intersection union result.
So therefore intersection is C [ ] = {2, 3}.

Difference of B minus A: $B-A = \{x \in U \mid x \in B \text{ and } x \notin A\}$
The difference of two sets B and A is the collection of objects in B that are not in A . The difference is written B – A.

Complement of A: $A^c = \{x \in U \mid x \notin A\}$
The compliment of a set A is the collection of objects in the universal set that are not in A.

Linked List: A linked list is a linear data structure consisting of a group of nodes which together represent a sequence.



Following are the basic operations supported by a list.

a) **Insertion –** add an element at the beginning of the list.

b) **Deletion –** delete an element at the beginning of the list.

c) **Display –** displaying complete list.

d) **Search –** search an element using given key.

e) **Delete –** delete an element using given key.

Algorithm: **Structure:**

```
1. struct node
2. {
3.     int iRoll;
4.     struct node *next;
5. };
```

Algorithm to insert node in Linked List:

```
1. struct node *funAddNode(struct node *Head,int r)
2. {
3.   create a 'temp' node and insert 'r' in 'iRoll'
4.
5.   if Head is NULL
6.         Head=temp;
7.   else
8.         trav=Head;
9.         while(trav->next!=NULL)
10.              trav=trav->next;
11.
12.        trav->next=temp;
13.
14.  return Head;
15.  }
```

Algorithm to print either or both flavors:

```
1. void funEitherBoth()
2. {
3.   set 'vTrav' to vanilla linked list
4.   set 'bTrav' to butterscotch linked list
5.
6.   while(vanilla linked list is not empty)
7.   {
8.        print vanilla linked list
9.    }
10.
```

```
11.         while(butterscotch ll is not emptys)
12.         {
13.                 flag=1;
14.                 vTrav=vHead;
15.                 while(vanilla ll is not empty)
16.                 {
17.                         if(bTrav->iRoll == vTrav->iRoll)
18.                                 flag=0;
19.
20.                         vTrav=vTrav->next;
21.                 }
22.                 if(flag==1)
23.                         then print vTrav->iRoll
24.
25.                 bTrav=bTrav->next;
26.         }
27.  }
```

Algorithm to print both flavors:

```
1. void funBoth()
2. {
3.    set 'vTrav' to vanilla linked list
4.    set 'bTrav' to butterscotch linked list
5.
6.   int flag;
7.   while(bTrav ll is not empty )
8.   {
9.          flag=1;
10.                 while(vTrav ll is not empty)
11.                 {
12.                         if(bTrav->iRoll == vTrav->iRoll)
13.                                 flag=0;
14.
15.                         vTrav=vTrav->next;
16.                 }
17.                 if(flag==0)
18.                         pirnt bTrav->iRoll
19.
20.                 bTrav=bTrav->next;
21.         }
22.  }
```

Time Complexity:

Insert / Delete at beginning:	O(1)
Insert / Delete at end:	O(n)
Search:	O(n)
Indexing:	O(n)

Test Cases:

```
Enter how many students: 4
Enter Roll: 25
Likes Vanilla          (yes=1 / no=0):1
Likes Butterscotch (yes=1 / no=0):1

Enter Roll: 26
Likes Vanilla          (yes=1 / no=0):1
Likes Butterscotch (yes=1 / no=0):0
Enter Roll: 27
```

```
Likes Vanilla        (yes=1 / no=0):0
Likes Butterscotch (yes=1 / no=0):1

Enter Roll: 28
Likes Vanilla        (yes=1 / no=0):0
Likes Butterscotch (yes=1 / no=0):0

Set of students who like either vanilla or butterscotch or both
={ 25, 26, 27}
Set of students who like both vanilla and butterscotch
={ 25}
Number of students who like neither vanilla nor butterscotch
=1
```

Conclusion : Thus, we implemented linear-dynamic data structure singly linked list to store two sets and have performed various operations on it.

Questions :

1) What are the advantages of LL implementation over array implementation?
2) What are the different set operations?
3) Explain node in LL.

GROUP D : ASSIGNMENTS

Assignment No.:	01 (GROUP D)
Title:	To check string for palindrome using stack in C++.
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 01 (Group D)

Title : Write C++ program to check string palindrome, to remove punctuation, to reverse the given string using stack data structure.

Objectives :

a) To understand the concept of stack using array.
b) To understand various stack operations.
c) To understand use of stack to check string palindrome.
d) To understand the use of stack to remove punctuation and to reverse the given string.

Problem Statement : A palindrome is a string of character that's the same forward and backward. Typically, punctuation, capitalization, and spaces are ignored. For example, "Poor Dan is in a droop" is a palindrome, as can be seen by examining the characters "poor danisina droop" and observing that they are thesame forward and backward. One way to check for a palindrome is to reverse the characters in the string and then compare with them the original-in a palindrome, the sequence will be identical. Write C++ program with functions-

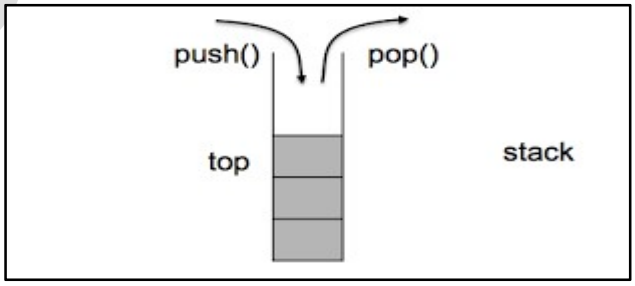
1. To print original string followed by reversed string using stack
2. To check whether given string is palindrome or not

Outcomes:

- Understanding the concept of stack using array.
- Understanding various stack operations.
- Understanding use of stack to check string palindrome.
- Understanding the use of stack to remove punctuation and to reverse the given string.

Software Requirements: g++ compiler on Ubuntu 14.04 (64bit)

Theory: **Stack** is a linear data structure which follows a particular order in which the operations are performed. The order may be LIFO(Last In First Out) or FILO(First In Last Out). In Stack all Operations such as **Insertion and Deletion** are permitted at only one end called **Top.**



Representation of stack

Mainly the following basic operations are performed in the stack:

- **Push():** Adds an item in the stack. If the stack is full, then it is said to be an Overflow condition.
- **Pop():** Removes an item from the stack. The items are popped in the reversed order in which they are pushed. If the stack is empty, then it is said to be an Underflow condition.

- **isFull()** – check if stack is full.
- **isEmpty()** – check if stack is empty.

Implementation:
There are two ways to implement a stack:

- Using array
- Using linked list

Array Implementation of stack:

```
void push(int x)
{
 if( isFull())
 {
  cout << "Stack Overflow";
 }
 else
 {
  a[++top] = x;
  cout << "Element Inserted";
 }
}
```

```
int pop()
{
 if(isEmpty())
 {
  cout << "Stack Underflow";
  return 0;
 }
 else
 {
  int d = a[top--];
  return d;
 }
}
```

```
bool isEmpty()
{
 if(top < 0)
    return 1;
 else
  return -1;
}
```

```
bool isFull()
{
 if( top >= Max)
 {
  return 1;
 }
 else
  return -1;
)
```

Algorithm: **Algorithm to check palindrome string using stack**

1. Find the length of the input string using strlen function and store it in a integer variable "length".
2. Using a for loop, traverse input string from index 0 to length-1 and push all characters in stack.
3. Remove (Pop) characters from stack one by one using a for loop and compare it with corresponding character of input string from beginning(traverse from index 0 to length-1). If we found a mismatch the input string is not a palindrome string otherwise palindrome string.

Algorithm to reverse a string using stack.

1. Create an empty stack.
2. One by one push all characters of string to stack.

3. One by one pop all characters from stack and put them back into another string.
4. Display string.

Test Cases: Take followign cases to check whether the string is palindrome or not:

1. **Input:** Enter a sting: **madam**
   Output: Sting is a palindrome,
   reverse of string is: **madam**

2. **Input:** Enter a string: **Data Structures Laboratory**
   Convert this string to desired format (i.e. remove all spaces, punctuations and convert uppercases into lowercases) to check for palindrome as following,
   datastructureslaboratory
   now check for palindrome,
   Output: Sting is NOT a palindrome,
   reverse of string is: **yrotarobaL serutcurtS ataD**

Conclusion : Thus, we implemented various stack operation to check string palindrome, to reverse the given string.

Questions :

1) What is stack?
2) Explain stack implementation using array.
3) Explain various stack operations.
4) What are the different applications of stack?

Assignment No.:	02 (GROUP D)
Title:	Infix to postfix conversion and Postfix evaluation using stack in C++.
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 02 (Group D)

Title : Implement C++ program for expression conversion as infix to postfix and its evaluation using stack.

Objectives :
a) To understand the use of stack.
b) To understand expression conversion as infix to postfix using stack.
c) To understand postfix expression evaluation using stack.

Problem Statement : Implement C++ program for expression conversion as infix to postfix and its evaluation using stack based on given conditions
i. Operands and operator, both must be single character.
ii. Input Postfix expression must be in a desired format.
iii. Only '+', '-', '*' and '/ ' operators are expected.

Outcomes:
- Understanding the use of stack.
- Understanding of expression conversion as infix to postfix using stack.
- Understanding the postfix expression evaluation using stack.

Software Requirements: g++ compiler on Ubuntu 14.04 (64bit)

Theory:

Stack is mainly used for expression conversion and expression evaluation purpose.
In any programming language, if we want to perform any calculation or to frame a condition etc., we use a set of symbols to perform the task. These set of symbols makes an expression.

An expression can be defined as follows:
An **expression** is a collection of operators and operands that represents a specific value.
In above definition, **operator** is a symbol which performs a particular task like arithmetic operation or logical operation or conditional operation etc.,
Operands are the values on which the operators can perform the task. Here operand can be a direct value or variable or address of memory location.

Based on the operator position, expressions are divided into three types. They are as follows:
- Infix Expression
- Postfix Expression
- Prefix Expression

Infix Expression
In infix expression, operator is used in between operands.
The general structure of an Infix expression is as follows...
Operand1 Operator Operand2
Example : a + b

Postfix Expression
In postfix expression, operator is used after operands. We can say that "**Operator follows the Operands**".

The general structure of Postfix expression is as follows...
Operand1 Operand2 Operator
Example : a b +

Prefix Expression
In prefix expression, operator is used before operands. We can say that "**Operands follows the Operator**".

The general structure of Prefix expression is as follows...
Operator Operand1 Operand2
Example : + a b

Any expression can be represented using the above three different types of expressions. And we can convert an expression from one form to another form like **Infix to Postfix**, **Infix to Prefix**, **Prefix to Postfix** and vice versa.

Infix to postfix conversion:

1. Scan through an expression, getting one token at a time.
2. Fix a priority level for each operator. For example, from high to low:
   3. - (unary negation)
   2. * /
   1. + - (subtraction)
3. Thus, high priority corresponds to high number in the table.
   If the token is an operand, do not stack it. Pass it to the output.
4. If token is an operator or parenthesis, do the following:
   i. Pop the stack until you find a symbol of lower priority number than the current one. An incoming left parenthesis will be considered to have higher priority than any other symbol. A left parenthesis on the stack will not be removed unless an incoming right parenthesis is found.
   The popped stack elements will be written to output.

Expression	Stack	Output
2	Empty	2
*	*	2
3	*	23
/	/	23*
(	/(	23*
2	/(	23*2
-	/(-	23*2
1	/(-	23*21
)	/	23*21-
+	+	23*21-/
5	+	23*21-/5
*	+*	23*21-/53
3	+*	23*21-/53
	Empty	23*21-/53*+

Figure1 :Infix to posfix conversion

   ii. Stack the current symbol.
   iii. If a right parenthesis is the current symbol, pop the stack down to (and including)

the first left parenthesis. Write all the symbols except the left parenthesis to the output (i.e. write the operators to the output).

iv. After the last token is read, pop the remainder of the stack and write any symbol (except left parenthesis) to output.

Postfix Evaluation:

Postfix Expression: 4 5+7 2 - *

In following figure evaluation of postfix expression is given.

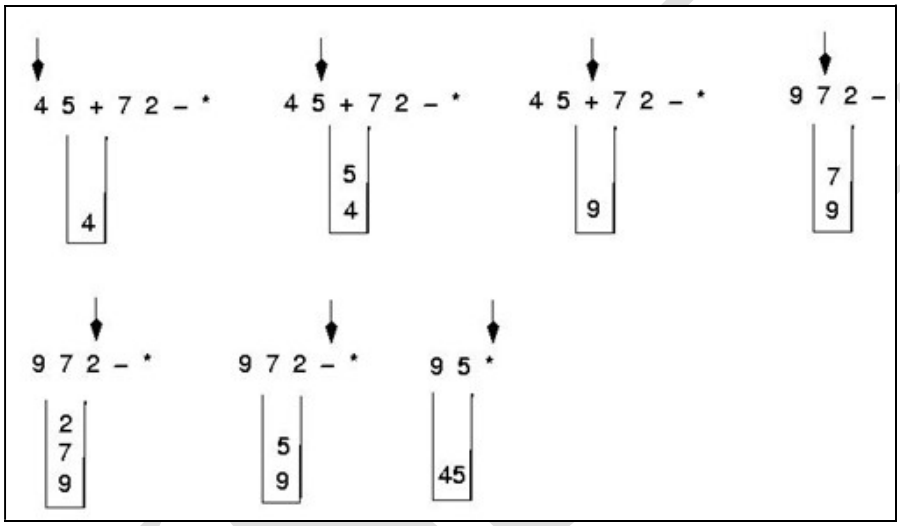


Figure 2: Postfix Evaluation

Algorithm: **Algorithm for Infix to Postfix conversion:**

1. Examine the next element in the input.
2. If it is operand, output it.
3. If it is opening parenthesis, push it on stack.
4. If it is an operator, then
   a. If stack is empty, push operator on stack.
   b. If the top of stack is opening parenthesis, push operator on stack
   c. If it has higher priority than the top of stack, push operator on stack.
   d. Else pop the operator from the stack and output it, repeat step 4
5. If it is a closing parenthesis, pop operators from stack and output them until an opening parenthesis is encountered. pop and discard the opening parenthesis.
6. If there is more input go to step 1
7. If there is no more input, pop the remaining operators to output.

Algorithm for evaluation of postfix expression

1. Initialize an empty stack.
2. Repeat the following until the end of the expression is encountered:
   - 2.1 Get next token (constant, variable, arithmetic operator) in the postfix expression.
   - 2.2 If token is an operand, push it onto the stack.

- If it is an operator, then
  i. Pop top two values from the stack.
  ii. If stack does not contain two items, error due to a malformed postfix.

Evaluation terminated.

iii. Apply the operator to these two values.

iv. Push the resulting value back onto the stack.

3. When the end of expression encountered, its value is on top of the stack (and, in fact, must be the only value in the stack).

Test Cases: **For conversion of expression from infix to postfix:**

Input: (A + B)

Output: A B +

For evaluation of postfix expression:

Input: A B *

Enter A: 4

Enter B: 5

Output: 20

Conclusion : Thus, we implemented expression conversion as infix to postfix and its evaluation using stack.

Questions :

1) What is expression?
2) Explain different type of expression.
3) Explain Infix to postfix conversion.
4) Explain Postfix evaluation.

GROUP E : ASSIGNMENTS

Assignment No.:	01 (GROUP E)
Title:	C++ program for simulating job queue (Priority Queue).
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 01 (Group E)

Title : C++ program for simulating job queue.

Objectives :
a) To understand the concept of queue.
b) To understand various operation of queue.
c) To understand use of various operations of queue using array.
d) To understand the concept and use of priority queue.

Problem Statement : Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write C++ program for simulating job queue. Write functions to add job and delete job from queue.

Outcomes:
- Understanding the concept of queue.
- Understanding the various operation of queue.
- Understanding use of various operations of queue using array.
- Understanding the use of priority queue.

Software Requirements: g++ compiler on Ubuntu 14.04 (64bit)

Theory: **Queue** is a linear structure which follows a particular order in which the operations are performed. The order is **F**irst **I**n **F**irst **O**ut (FIFO). A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.
The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Array implementation Of Queue
For implementing queue, we need to keep track of two indices, front and rear. We enqueue an item at the rear and dequeue an item from front.

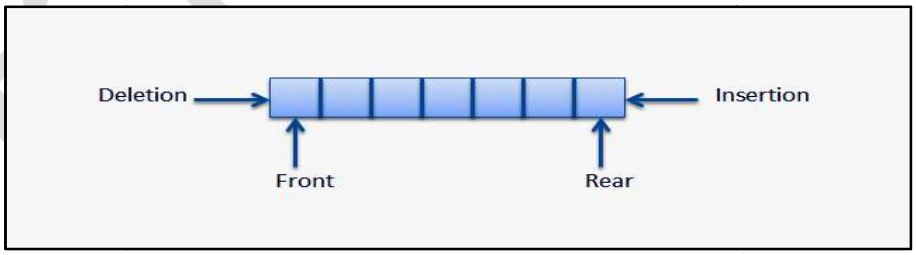


Figure 1: Representation of queue

Operations on Queue:
Mainly the following four basic operations are performed on queue:

- **enqueue():** Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.
- **dequeue():** Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.

- **isfull():** checks if queue is full.
- **isempty():** checks if queue is empty.

isfull() Operation:	**isempty() Operation :**
bool isfull() { if(rear == MAXSIZE - 1) return true; else return false; }	bool isempty() { if(rear==front-1) return true; else return false; }

Enqueue Operation(i.e. insertion):	**Dequeue Operation(i.e deletion):**
int enqueue(int data) { if(isfull()) return 0; rear = rear + 1; queue[rear] = data; return 1; }	int dequeue() { if(isempty()) return 0; int data = queue[front]; front = front + 1; return data; }

Priority Queue is more specialized data structure than Queue. Like ordinary queue, priority queue has same method but with a major difference. In Priority queue items are ordered by key value so that item with the lowest value of key is at front and item with the highest value of key is at rear or vice versa.

A priority queue is used to handle the job queue of an operating system. So for the implementation of jobs of operating system priority queue is used. As the jobs are assigned in queue depending upon their priority.

Priority Queue is Collection of entities or elements in which :

- **Addition** of **element** is done on **basis of priority**.
- **Removal** of **element** is done at **FRONT.**

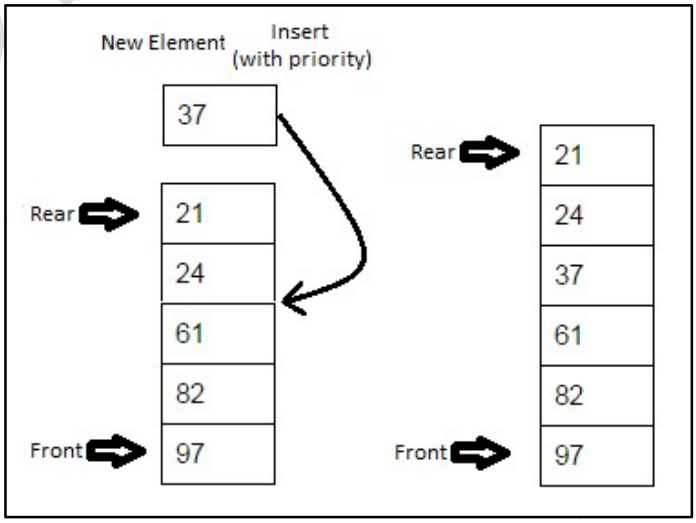


Figure 2: Insertion of element in priority queue depending upon priority.

Applications of Queue:

Queue is used when things don't have to be processed immediately, but have to be processed in **F**irst **I**n**F**irst **O**ut order like Breadth First Search. This property of Queue makes it also useful in following kind of scenarios.

1) When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
2) When data is transferred asynchronously (data not necessarily received at same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.

Algorithm: **Enqueue Operation**

As queue maintains two data pointers, **front** and **rear**, its operations are comparatively more difficult to implement than Queue.

The following steps should be taken to enqueue (insert) data into a queue :

Step 1 – Check if queue is full.
Step 2 – If queue is full, produce overflow error and exit.
Step 3 – If queue is not full, increment **rear** pointer to point next empty space.
Step 4 – Add data element to the queue location, where rear is pointing.
Step 5 – return success.

Dequeue Operation

Accessing data from queue is a process of two tasks – access the data where **front** is pointing and remove the data after access.

The following steps are taken to perform **dequeue** operation:

Step 1 – Check if queue is empty.
Step 2 – If queue is empty, produce underflow error and exit.
Step 3 – If queue is not empty, access data where **front** is pointing.
Step 3 – Increment **front** pointer to point next available data element.
Step 5 – return success.

Insertion operation of priority queue:

```
void insert(int data)
{
  int i = 0;

  if(!isFull())
  {                              // if queue is empty, insert the data
        if(itemCount == -1 )
        {
              intArray[++itemCount] = data;
        }
       else
       {                                  // start from the right end of the queue
             itemCount++;
             for(i = itemCount - 1; i >= 0; i-- )
              {                           // if data is larger, shift existing item to right end
                    if(data > intArray[i])
```

```
                    {
                        intArray[i+1] = intArray[i];
                    }
                    else
                    {
                        break;
                    }
                }
                intArray[i+1] = data;        // insert the data
            }
        }
```

Test Cases:

```
Enter how many jobs: 5
Enter Job (job priority 1-9) : 7
Enter Job (job priority 1-9) : 3
Enter Job (job priority 1-9) : 4
Enter Job (job priority 1-9) : 1
Enter Job (job priority 1-9) : 2

Dequeue jobs:
Job: 1
Job: 2
Job: 3
Job: 4
Job: 7
```

Conclusion : Thus, we implemented simulation of jobs using priority queue.

Questions :

1. Explain queue ADT.
2. Explain priority queue ADT.
3. Explain array representation of queue.
4. What are the applications of priority queue?

Assignment No.:	02 (GROUP D)
Title:	Deque implementation in C++.
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 02 (Group E)

Title : Deque implementation in C++.

Objectives :
a) To understand the concept of deque.
b) To understand various deque operations.
c) To understand the use of various deque operation.

Problem Statement : A double-ended queue (deque) is a linear list in which additions and deletions may be made at either end. Obtain a data representation mapping a deque into a one-dimensional array. Write C++ program to simulate deque with functions to add and delete elements from either end of the deque.

Outcomes:
- Understanding the concept of deque.
- Understanding various deque operations.
- Understanding the use of various deque operation.

Software Requirements: g++ compiler on Ubuntu 14.04 (64bit)

Theory: A **deque**, also known as a double-ended queue, is an ordered collection of items of similar types. It has two ends, a front and a rear. It allows inserting and deleting from both ends means items can be added or deleted from the front or rear end. Deque can be behave like a queue by using only enq_front and deq_front , and behaves like a stack by using only enq_front and deq_rear.

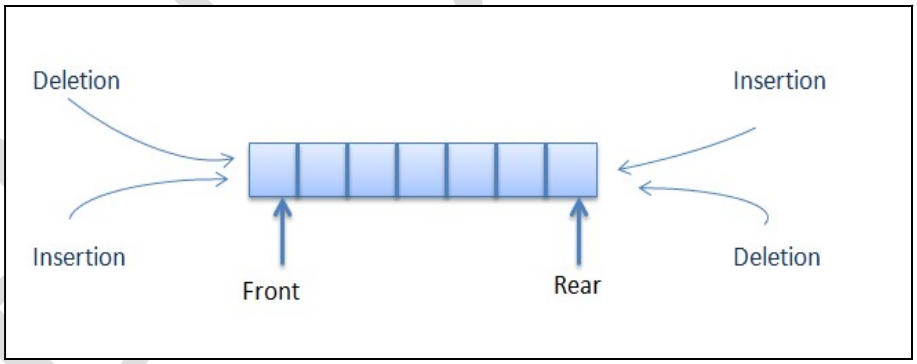


Figure1: Representation of deque

Double Ended Queue can be represented in two ways, those are as follows...
- Input Restricted Double Ended Queue
- Output Restricted Double Ended Queue

Input Restricted Double Ended Queue
In input restricted double ended queue, the insertion operation is performed at only one end and deletion operation is performed at both the ends.

Output Restricted Double Ended Queue
In output restricted double ended queue, the deletion operation is performed at only one end and insertion operation is performed at both the ends.

Various operations of deque:

- Insert element at rear
- Insert element at front
- Remove element at front
- Remove element at rear

Algorithm: **Algorithm for Insertion of element from front:**

Step1. Start
Step2. Check the queue is full or not as if (r == maxsize-1) &&(f==0)
Step3. If false update the pointer f as f= f-1(**OR** f=((f-1)+ maxsize)% maxsize)
Step4. Insert the element at pointer f as Q[f] = element
Step5. Stop

Algorithm for Insertion of element from rear:

Step1: Start
Step2: Check the queue is full or not as if (r == maxsize-1) &&(f==0) if yes queue is full
Step3: If false update the pointer r as r= r+1
Step4: Insert the element at pointer r as Q[r] = element
Step5: Stop

Algorithm for deletion of element from front:

Step1: Start
Step2: Check the queue is empty or not as if (f == r) if yes queue is empty.
Step3: If false update pointer f as f = f+1and delete element at position f as element= Q[f]
Step4: If (f== r) reset pointer f and r as f = r = -1
Step5: Stop

Algorithm for deletion of element from rear:

step1. Start
step2. Check the queue is empty or not as if (f == r) if yes queue is empty
step3. If false delete element at position r as element = Q[r]
step4. Update pointer r as r = r-1(**OR** r=((r-1)+maxsize)%maxsize)
step5. If (f == r) reset pointer f and r as f = r= -1
step6. Stop.

Test Cases: Insert and delete the element in queue from rear end or front end.
Menu driven program with four choice:
1. Insertion from rear end
2. Insertion from front end
3. Deletion from front end
4. Deletion from rear end
Input : Element to be inserted or deleted
Output : Successful implementation of operation(i.e. insertion or deletion as per user choice)

Conclusion : Thus, we implemented various deque operations using array.

Questions :

1) Explain Deque ADT.
2) Explain concept of deque.
3) What are the applications of deque?
4) Differentiate queue, Deque and Priority queue.

Assignment No.:	03 (GROUP E)
Title:	C++ program for simulating circular queue.
Subject:	Data Structures Laboratory
Class:	S.E. (C.E.)
Roll No.:	
Assessment (Marks):	
Signature and Date of Assessment:	

Assignment No. 03 (Group E)

Title : C++ program for simulating circular queue.

Objectives :
a) To study Circular Queue as data structure.
b) To understand implementation of Circular Queue and perform various operations on it.

Problem Statement : Pizza parlor accepting maximum M orders. Orders are served in first come first served basis. Order once placed cannot be cancelled. Write C++ program to simulate the system using circular queue using array.

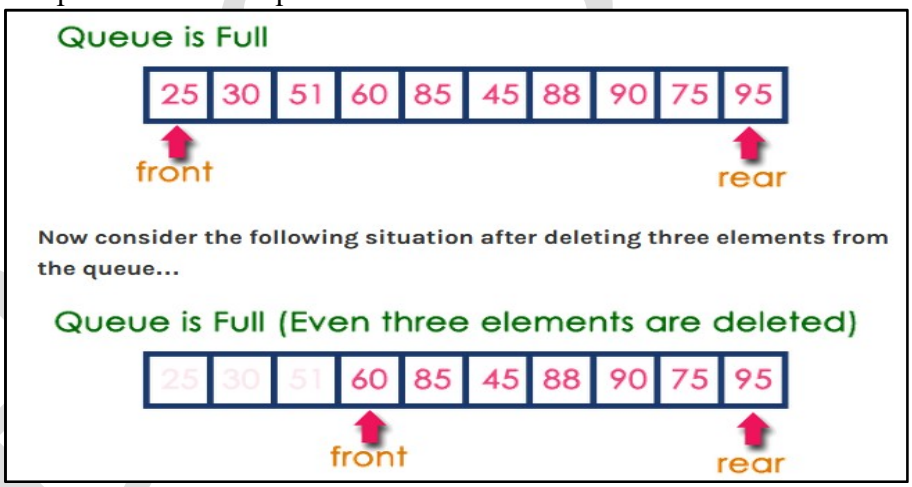
Outcomes:
- Understanding the concept of circular queue.
- Implement Circular Queue and perform various basic operations on it.

Software Requirements: g++ compiler on Ubuntu 14.04 (64bit)

Theory:

Circular Queue

In a normal Queue Data Structure, we can insert elements until queue becomes full. But once if queue becomes full, we can't insert the next element until all the elements are deleted from the queue. For example consider the queue below:

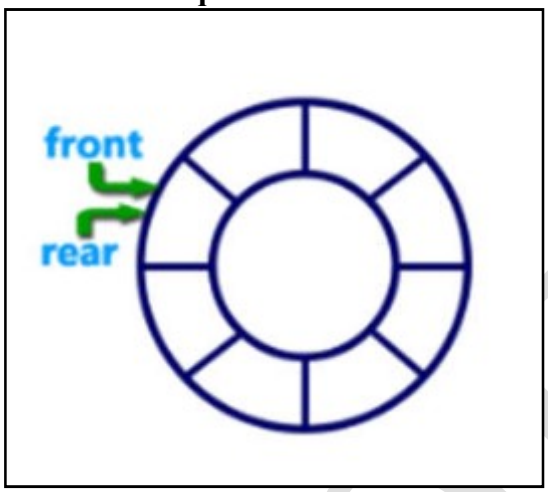


This situation also says that Queue is Full and we can't insert the new element because, 'rear' is still at last position. In above situation, even though we have empty positions in the queue we can't make use of them to insert new element. This is the major problem in normal queue data structure. To overcome this problem we use circular queue data structure.

What is a Circular Queue?

A Circular Queue can be defined as follow:
Circular Queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle.

Graphical representation of a circular queue is as follows:



Algorithm: Implementation of Circular Queue:

To implement a circular queue data structure using array, we first perform the following steps before we implement actual operations.

Step 1: Include all the header files which are used in the program and define a constant 'SIZE' with specific value.
Step 2: Declare all user defined functions used in circular queue implementation.
Step 3: Create a one dimensional array with above defined SIZE (int cQueue[SIZE])
Step 4: Define two integer variables 'front' and 'rear' and initialize both with '-1'. (int front = -1, rear = -1)
Step 5: Implement main method by displaying menu of operations list and make suitable function calls to perform operation selected by the user on circular queue.

enQueue(value) – Inserting value into the Circular Queue:

In a circular queue, enQueue() is a function which is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at rear position. The enQueue() function takes one integer value as parameter and inserts that value into the circular queue. We can use the following steps to insert an element into the circular queue...
Step 1: Check whether queue is FULL. ((rear == SIZE-1 && front == 0) || (front == rear+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 check rear == SIZE – 1 && front != 0 if it is TRUE, then set rear = -1.
Step 4: Increment rear value by one (rear++), set queue[rear] = value and check 'front == -1' if it is TRUE, then set front = 0.

deQueue() – Deleting a value from the Circular Queue:

In a circular queue, deQueue() is a function used to delete an element from the circular queue. In a circular queue, the element is always deleted from front position. The deQueue() function doesn't take any value as parameter. We can use the following steps to delete an element from the circular queue…
Step 1: Check whether queue is EMPTY. (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 display queue[front] as deleted element and increment the front value by one (front ++). Then check whether front == SIZE, if it is TRUE, then set front = 0. Then check whether both front – 1 and rear are equal (front -1 == rear), if it TRUE, then set both front and rear to '-1' (front = rear = -1).

display() – Displays the elements of a Circular Queue:

We can use the following steps to display the elements of a circular queue…
Step 1: Check whether queue is EMPTY. (front == -1)
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'.
Step 4: Check whether 'front <= rear', if it is TRUE, then display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until 'i <= rear' becomes FALSE.
Step 5: If 'front <= rear' is FALSE, then display 'queue[i]' value and increment 'i' value by one (i++). Repeat the same until'i <= SIZE – 1′ becomes FALSE.
Step 6: Set i to 0.
Step 7: Again display 'cQueue[i]' value and increment i value by one (i++). Repeat the same until 'i <= rear' becomes FALSE.

Test Cases:

```
----------PizzaParlor Menu ----------
1. Place Order
2. Serve Order
3. Display Orders
4. Exit

Enter your choice: 1

Enter Order ID: 1
Enter Orderer Name : ABC Piza

Order Placed successfully
```

Conclusion : Thus, we implemented Circular Queue and perform various operations of it

Questions :

1. What are the advantages of circular queue over linear queue.
2. Explain circular queue ADT.