



**Dr. D. Y. Patil Pratishthan's**

**DR. D. Y. PATIL INSTITUTE OF ENGINEERING, MANAGEMENT &  
RESEARCH**

Approved by A.I.C.T.E, New Delhi, Maharashtra State Government, Affiliated to Savitribai Phule Pune University Sector No. 29, PCNTDA, Nigdi Pradhikaran, Akurdi, Pune 411044. Phone: 020-27654470, Fax: 020-27656566 Website : [www.dypiemr.ac.in](http://www.dypiemr.ac.in) Email : [principal.dypiemr@gmail.com](mailto:principal.dypiemr@gmail.com)

# **Department of Artificial Intelligence and Data Science**

## **LAB MANUAL**

## **Fundamentals of Data Structures Laboratory**

### **Second Year Engineering (2020 Course) Semester-I**

**Prepared By: Ms. Arti Singh**

**Mrs. Priyanka Abhale**

**Ms. Kalyani Kute**



## 217522: Data Structures Laboratory

<b>Teaching Scheme</b> <b>Practical: 04</b> <b>Hours/Week</b>	<b>Credit</b> <b>Scheme 02</b>	<b>Examination Scheme and</b> <b>Marks Term Work: 25 Marks</b> <b>Practical: 50 Marks</b>
---	-----------------------------------	---

**Companion Course: 210242: Fundamental of Data Structures**

### **Course Objectives:**

To understand basic techniques and strategies of algorithm analysis, the memory requirement for various data structures like array, linked list, stack, queue etc using concepts of python and C++ programming language.

### **Course Outcomes:**

On completion of the course, learner will be able to–

CO1: Use algorithms on various linear data structure using sequential organization to solve real life problems.

CO2: Analyze problems to apply suitable searching and sorting algorithm to various applications.

CO3: Analyze problems to use variants of linked list and solve various real life problems.

CO4: Designing and implement data structures and algorithms for solving different kinds of problems.

Sr.No		Title of the Experiment	Page No
<b>Group A</b>			
1	A1	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 <b>Python</b> 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. (Note- While realizing the group, duplicate entries should be avoided, Do not use SET built-in functions)	5
2	A2	Write a <b>Python</b> 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	9
3	A4	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. Suppose the following input is supplied to the program: D 300, D 300 , W 200, D 100 Then, the output should be: 500	12
<b>Group B</b>			
4	B12	a) Write a <b>Python</b> program to store names and mobile numbers of your friends in sorted order on names. Search your friend from list using binary search (recursive and non- recursive). Insert friend if not present in phonebook b) Write a <b>Python</b> program to store names and mobile numbers of your friends in sorted order on names. Search your friend from list using Fibonacci search. Insert friend if not present in phonebook.	17
5	B14	Write a <b>Python</b> 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.	22
6	B16	Write a <b>Python</b> 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.	26
<b>Group C</b>			
7	C19	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	30

		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: a) Add and delete the members as well as president or even secretary. b) Compute total number of members of club c) Display members d) Two linked lists exists for two divisions. Concatenate two lists.	
8	C22	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 linkedlist. compute and display- a) Set of students who like both vanilla and butterscotch b) Set of students who like either vanilla or butterscotch or not both c) Number of students who like neither vanilla nor butterscotch	39
<b>Group D</b>			
9	D25	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 the same 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- a) To print original string followed by reversed string using stack b) To check whether given string is palindrome or not	47
10	D26	In any language program mostly syntax error occurs due to unbalancing delimiter such as (), {}, []. Write C++ program using stack to check whether given expression is well parenthesized or not.	51
<b>Group E</b>			
11	E29	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.	56
12	E31	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.	61
13	E32	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.	64

## Assignment no: 01

**Aim:** To study and understand the concept of set theory using python.

### Problem definition:

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.

(Note-While realizing the group, duplicate entries should be avoided, do not use SET built-in functions)

### Learning Objectives:

To understand basic techniques and strategies of algorithm using concepts of python.

### Learning Outcome:

Students will be able to use algorithms on various linear data structure using sequential organization to solve real life problems.

### Theory:

#### Python:

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

#### Set Operations:

We have to perform here different set operations like Union, Intersections, Difference, Symmetric Difference.

#### Universal set U:

Often a discussion involves subsets of some particular set called the universe of discourse (or

---

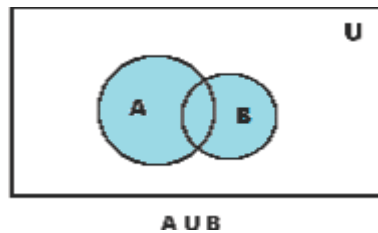
briefly universe), universal set or space. The elements of a space are often called the points of the space. We denote the universal set by  $U$ .

**Example.** The set of all even integers could be considered a subset of a universal set consisting of all the integers. Or they could be considered a subset of a universal set consisting of all the rational numbers. Or of all the real numbers.

Often the universal set may not be explicitly stated and it may be unclear as to just what it is. At other times it will be clear.

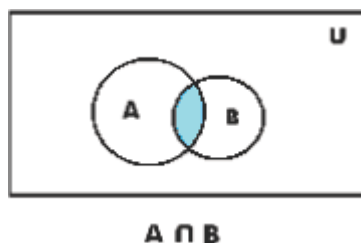
1. Union Operation: In set theory the union of collection of sets is the set of all distinct elements in the collection. The union of two sets  $A$  and  $B$  is the set consisting of all elements in  $A$  plus all elements in  $B$  and is denoted by  $A \cup B$  or  $A + B$ .

**Example.** If  $A = \{a, b, c, d\}$  and  $B = \{b, c, e, f, g\}$  then  $A \cup B = \{a, b, c, d, e, f, g\}$ .



1. Intersection operation: In set theory intersection is a operation where we collect common elements of different sets. The intersection of two sets  $A$  and  $B$  is the set consisting of all elements that occur in both  $A$  and  $B$  (i.e. all elements common to both) and is denoted by  $A \cap B$ ,  $A \cdot B$  or  $AB$ .

**Example.** If  $A = \{a, b, c, d\}$  and  $B = \{b, c, e, f, g\}$   
then  $A \cap B = \{b, c\}$ .



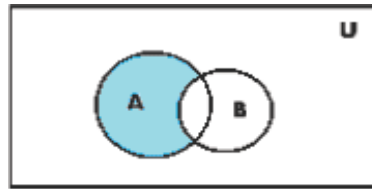
2. Difference Operation: It is a generalization of the idea of the complement of a set and as a

---

such is sometimes called the relative complement of T with respect to S where T and s are two sets.

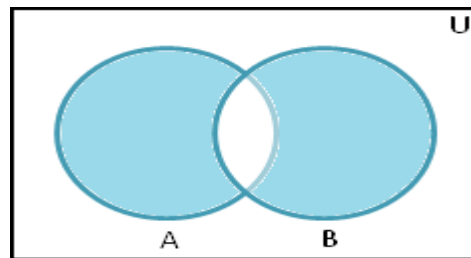
The set consisting of all elements of a set A that do not belong to a set B is called the difference of A and B and denoted by  $A - B$ .

**Example.** If  $A = \{a, b, c, d\}$  and  $B = \{b, c, e, f, g\}$  then  $A - B = \{a, d\}$ .



$A - B$

3. Symmetric Difference: The symmetric difference between two sets S and t is the union of S-T and T-S. The symmetric difference using Venn diagram of two subsets A and B is a sub set of U, denoted by  $A \Delta B$  and is defined by  $A \Delta B = (A - B) \cup (B - A)$



$A \Delta B$

**Input:** Enter the total number of students in class, also enter the student who plays cricket, badminton and football.

**Output:** Union, intersection, set difference of entered students.

**Algorithm/Pseudo code:**

**1. Function for union:**

```
def
find_union_set(A,B,C): for
    i in range(len(A)):
        C.append(A[i])
    for i in range(len(B)):
        flag = search_set(A,B[i]);
```

```
    if(flag == 0) :  
        C.append(B[i])
```

## 2. Function for Intersection:

```
def find_intersection_set(A,B,C):  
    for i in range(len(A)):  
        flag = search_set(B,A[i]);  
        if(flag == 1) :  
            C.append(A[i])
```

## 3. Function for Difference:

```
def  
    find_difference_set(A,B,C):  
    for i in range(len(A)):  
        flag = search_set(B,A[i]);  
        if(flag == 0) :  
            C.append(A[i])
```

**Software required:** Open Source Python, Programming tool like Jupyter Notebook, Pycharm, Spyder.

**Conclusion:** Thus, we have studied use of set operations using python.

---



## Assignment No-02

**Aim:** To illustrate the various functions in python.

**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

### Learning Objectives:

To understand basic techniques and strategies of algorithm analysis, the memory requirement for various data structures using the concepts of python.

**Learning Outcome:** Students will be able to use algorithms on various linear data structure using sequential organization to solve real life problems

**Theory:** Lists are one of the most powerful tools in python. They are just like the arrays declared in other languages. But the most powerful thing is that list need not be always homogenous. A single list can contain strings, integers, as well as objects. Lists can also be used for implementing stacks and queues. Lists are mutable, i.e., they can be altered once declared.

A tuple is a sequence of immutable Python objects. Tuples are just like lists with the exception that tuples cannot be changed once declared. Tuples are usually faster than lists

**Input:** enter marks scored in subject “Fundamental of Data Structure” by N students in the class.

**Output:** Average score, highest score, lowest score, count of absentees, marks with highest frequency.

---

### Algorithm/Pseudocode:

#### The average score of class

```
def find_average_score_of_class(A) :  
    sum = 0  
    for i in range(len(A)) :  
        if(A[i] != -1) :  
            sum = sum +  
            A[i]  
    avg = sum /  
    len(A)  
    display_marks(A)  
    print("\nAverage score of class is %.2f\n\n"%avg)
```

#### Highest score and lowest score of class

```
def find_highest_and_lowest_score_of_class(A) :  
    max = -1  
    min = 31  
    for i in range(1,len(A)) :  
        if(max < A[i]) :  
            max = A[i]  
            max_ind =  
                i  
        if(min > A[i] and A[i] != -1) :  
            min = A[i]  
            min_ind = i  
    display_marks(A  
        )  
    print("Highest Mark Score of class is %d scored by student  
%d"%(max,max_ind+1))  
    print("Lowest Mark Score of class is %d scored by student  
%d"%(min,min_ind+1))
```

---

**Count of students who were absent for the test:**

```
def find_count_of_absent_students(A)
: count = 0
for i in range(len(A)):
    if(A[i] == -1) :
        count += 1
display_marks(A
    )
print("\tAbsent Student Count = %d"%count)
```

**Display mark with highest frequency:**

```
def display_mark_with_highest_frequency(A) :
    freq = 0
    for i in range(len(A)) :
        count = 0
        if(A[i] != -1) :
            for j in range(len(A)):
                if(A[i] == A[j]) :
                    count += 1
            if(freq < count)
            :
                Marks =
                A[i] freq =
                count
    display_marks(A)
    print("\nMarks with highest frequency is %d (%d)"%(Marks,freq))
```

**Software required:** Open Source Python, Programming tool like Jupyter Notebook, PyCharm, Spyder.

**Conclusion:** Thus, we have studied the implementation of various python operations.

---

## Assignment no: 03

**Aim:** To study and understand the concept of a bank account based transaction using Python.

### Problem definition:

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. Suppose the following input is supplied to the program:

D 300, D 300 , W 200, D 100 Then, the output should be: 500

### Learning Objectives:

To understand the concept of function using python.

### Learning Outcome:

Students will be able to use algorithms on various data structures using sequential organization to solve real life problems.

### Theory:

We define a Bank class representing a bank. It has an attribute called customers, which is initially an empty dictionary.

### create\_account():

The "create\_account()" method takes an account number and an optional initial balance as arguments. It checks if the account number already exists in the customers' dictionary. If it does, it prints a message indicating that the account number already exists. Otherwise, the account number is added as a key with the initial balance as the value.

### Make\_deposit():

The "make\_deposit()" method takes an account number and an amount as arguments. It checks if the

---

account number exists in the customer's dictionary. If it does, it adds the amount to the current balance of the account. An error message is printed if the account number does not exist.

### **Make\_withdrawal():**

The "make\_withdrawal()" method takes an account number and an amount as arguments. It checks if the account number exists in the customer's dictionary. If it does, it checks if the account has sufficient funds to withdraw. If it does, it subtracts the amount from the current account balance. Otherwise, it prints an error message indicating insufficient funds. An error message is displayed if the account number does not exist.

### **Check\_balance():**

The "check\_balance()" method takes an account number as an argument. It checks if the account number exists in the customer's dictionary. If it does, it retrieves and prints the current account balance. If the account number does not exist, it prints a message.

In the example usage section, we create an instance of the Bank class called bank. We use various methods to create customer accounts, make deposits, withdraw, and check account balances.

**Input:** Enter the data D 300, D 300 , W 200, D 100

**Output:** Then, the output should be: 500

### **Algorithm/Pseudocode:**

- **Deposit:**

```
def deposit(self):  
  
    self.total_amount += float(input('Hello {}, please enter amount to deposit:  
'.format(self.name)))  
  
    self.print_current_balance()
```



- **Withdraw:**

```
def withdraw(self):  
  
    amount_to_withdraw = float(input('Enter amount to withdraw: '))  
  
    if amount_to_withdraw > self.total_amount:  
  
        print('Insufficient Balance !!')  
  
    else:  
  
        self.total_amount -= amount_to_withdraw  
  
    self.print_current_balance()
```

**Software required:** Open Source Python, Programming tool like Jupyter Notebook, Spyder.

**Conclusion:** Thus, we have studied and implemented the bank account based transaction using Python



## Assignment No- 04

**Aim:** To illustrate the various searching techniques.

**Problem Statement:** a) Write a Python program to store names and mobile numbers of your friends in sorted order on names. Search your friend from list using binary search (recursive and non- recursive). Insert friend if not present in phonebook

b) Write a Python program to store names and mobile numbers of your friends in sorted order on names. Search your friend from list using Fibonacci search. Insert friend if not present in phonebook.

### Learning Objectives:

To understand concept of searching.

To compare time complexity of various searching techniques.

**Learning Outcome:** Students will be able to analyze problems to apply suitable searching and sorting algorithm to various applications.

### Theory:

#### Binary Search:

For a binary search to work, it is mandatory for the target array to be sorted. We shall learn the process of binary search with a pictorial example. The following is our sorted array and let us assume that we need to search the location of value 31 using binary search.

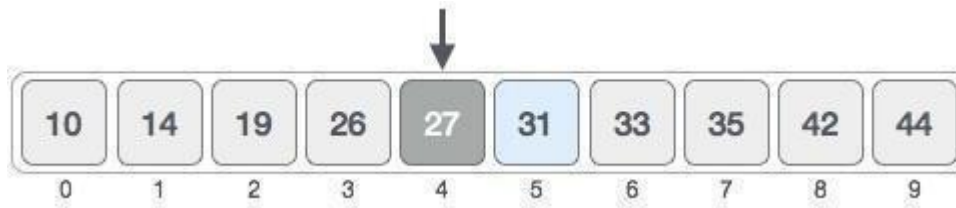


First, we shall determine half of the array by using this formula

$$\text{mid} = \text{low} + (\text{high} - \text{low}) / 2$$

Here it is,  $0 + (9 - 0) / 2 = 4$  (integer value of 4.5). So, 4 is the mid of the array.





Now we compare the value stored at location 4, with the value being searched, i.e. 31. We find that the value at location 4 is 27, which is not a match. As the value is greater than 27 and we have a sorted array, so we also know that the target value must be in the upper portion of the array.



We change our low to mid + 1 and find the new mid value again.

$low = mid + 1$

$mid = low + (high - low) / 2$

Our new mid is 7 now. We compare the value stored at location 7 with our target value 31.

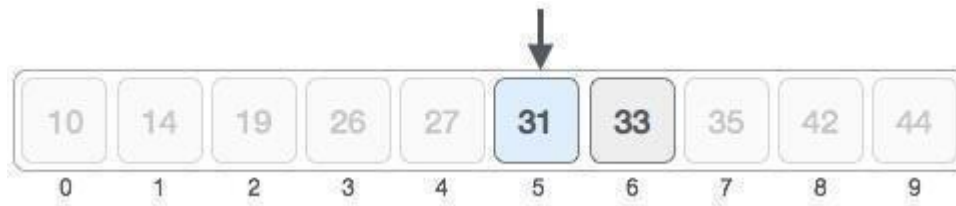


The value stored at location 7 is not a match, rather it is more than what we are looking for. So, the value must be in the lower part from this location.



Hence, we calculate the mid again. This time it is 5.





We compare the value stored at location 5 with our target value. We find that it is a match.



We conclude that the target value 31 is stored at location 5.

Binary search halves the searchable items and thus reduces the count of comparisons to be made to very less numbers.

### **Fibonacci Search:**

Fibonacci search is an efficient search algorithm based on divide and conquer principle that can find an element in the given sorted array with the help of Fibonacci series in  $O(\log N)$  time complexity. This is based on Fibonacci series which is an infinite sequence of numbers denoting a pattern which is captured by the following equation:

$$F(n+1) = F(n) + F(n-1)$$

where  $F(i)$  is the  $i$ th number of the Fibonacci series where  $F(0)$  and  $F(1)$  are defined as 0 and 1 respectively.

The first few Fibonacci numbers are:

0, 1, 1, 2, 3, 5, 8, 13, ....

$$F(0) = 0$$

$$F(1) = 1$$

$$F(2) = F(1) + F(0) = 1 + 0 = 1$$

$$F(3) = F(2) + F(1) = 1 + 1 = 2$$

$$F(4) = F(3) + F(2) = 1 + 2 = 3 \text{ and so continues the series}$$



Other searches like binary search also work for the similar principle on splitting the search space to a smaller space but what makes Fibonacci search different is that it divides the array in unequal parts and operations involved in this search are addition and subtraction only which means light arithmetic operations takes place and hence reducing the work load of the computing machine.

**Input:** Enter the names and mobile numbers of your friend in sorted order

**Output:** Searching by Binary search and Fibonacci search

**Algorithm/Pseudo code:**

**Binary Search:**

Procedure binary\_search

A  $\leftarrow$  sorted array

n  $\leftarrow$  size of array

x  $\leftarrow$  value to be searched

Set lowerBound = 1

Set upperBound = n

while x not found

if upperBound <

lowerBound EXIT: x does

not exists.

set midPoint = lowerBound + ( upperBound - lowerBound ) / 2

if A[midPoint] < x

set lowerBound = midPoint + 1

if A[midPoint] > x

set upperBound = midPoint - 1

---

```
    if A[midPoint] = x
        EXIT: x found at location
    midPoint end while

end procedure
```

### **Fibonacci Search:**

Let the length of given array be  $n$   $[0...n-1]$  and the element to be searched be  $x$ . Then we use the following steps to find the element with minimum steps:

1. Find the smallest Fibonacci number greater than or equal to  $n$ . Let this number be  $fb(M)$  [ $m$ 'th Fibonacci number]. Let the two Fibonacci numbers preceding it be  $fb(M-1)$  [ $(m-1)$ 'th Fibonacci number] and  $fb(M-2)$  [ $(m-2)$ 'th Fibonacci number].
2. While the array has elements to be checked:
  - > Compare  $x$  with the last element of the range covered by  $fb(M-2)$
  - > If  $x$  matches, return index value
  - > Else if  $x$  is less than the element, move the third Fibonacci variable two Fibonacci down, indicating removal of approximately two-third of the unsearched array.
  - > Else  $x$  is greater than the element, move the third Fibonacci variable one Fibonacci down. Reset offset to index. Together this results into removal of approximately front one-third of the unsearched array.
3. Since there might be a single element remaining for comparison, check if  $fbMm1$  is '1'. If Yes, compare  $x$  with that remaining element. If match, return index value.

**Software required:** Open Source Python, Programming tool like Jupyter Notebook, PyCharm, Spyder.

**Conclusion:** Thus, we have studied the implementation of various sorting techniques (Bubble and Selection)

---

## Assignment No- 05

**Aim:** To illustrate the various sorting techniques.

**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

### Learning Objectives:

To understand concept of sorting.

To compare time complexity of various sorting techniques.

**Learning Outcome:** Students will be able to analyze problems to apply suitable searching and sorting algorithm to various applications.

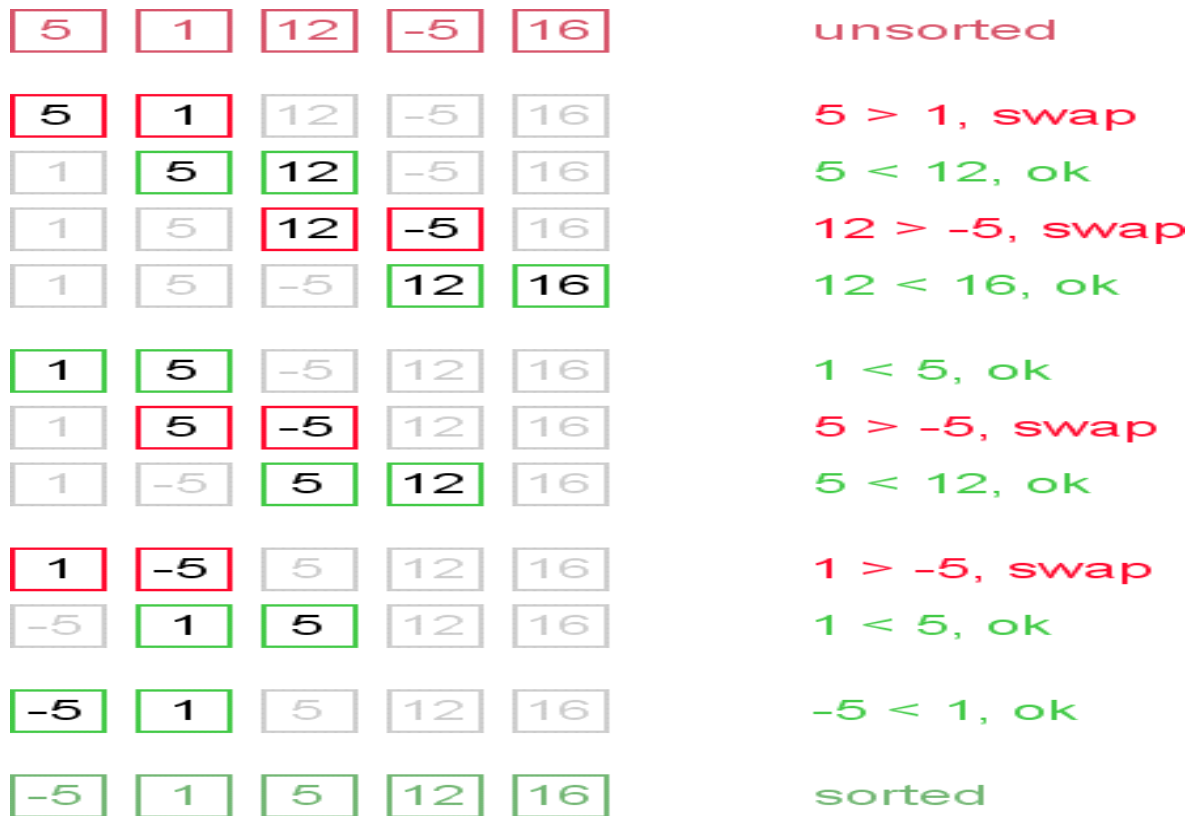
### Theory:

#### Bubble Sort

Bubble sort is a simple and well-known sorting algorithm. It is used in practice once in a blue moon and its main application is to make an introduction to the sorting algorithms. Bubble sort belongs to  $O(n^2)$  sorting algorithms, which makes it quite inefficient for sorting large data volumes. Bubble sort is **stable** and **adaptive**.

#### Example :

---



### Selection Sort

Selection sort is one of the  $O(n^2)$  sorting algorithms, which makes it quite inefficient for sorting large data volumes. Selection sort is notable for its programming simplicity and it can outperform other sorts in certain situations (see complexity analysis for more details).



**Input:** Enter the first percentage of students

**Output:** sorting by selection and bubble sort and display top 5 students marks

#### Algorithm for Bubble Sort:

We assume list is an array of n elements. We further assume that swap function, swaps the values of given array elements.

begin BubbleSort(list)

```
    for all elements of
        list if list[i] >
            list[i+1]
                swap(list[i],
                    list[i+1]) end if
    end for
return
list
end BubbleSort
```

**Time Complexity =  $O(n^2)$**

### **Algorithm for Selection**

#### **Sort:**

We assume list is an array of n elements. We further assume that swap function, swaps the values of given array elements.

```
Begin SelectionSort(list)
    for all elements of list
        for j to all element of
            list
                if list[i] > list[j]
                    swap(list[i],
                        list[j])
                end if
            end for
        end for
    return
    list
end SelectionSort
```

**Software required:** Open Source Python, Programming tool like Jupyter Notebook, PyCharm, Spyder.

**Conclusion:** Thus, we have studied the implementation of various sorting techniques (Bubble and Selection)

---

## Assignment No- 06

**Aim:** To illustrate Quick 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 quick sort and display top five scores.

### Learning Objectives:

To understand concept of sorting.

To find time complexity of Quick Sort.

**Learning Outcome:** Students will be able to analyze problems to apply suitable searching and sorting algorithm to various applications.

### Theory of Quick sort:

Quicksort is a fast sorting algorithm, which is used not only for educational purposes, but widely applied in practice. On the average, it has  $O(n \log n)$  complexity, making quicksort suitable for sorting big data volumes. The idea of the algorithm is quite simple and once you realize it, you can write quicksort as fast as bubble sort.

#### Algorithm

The divide-and-conquer strategy is used in quicksort. Below the recursion step is described:

Choose a pivot value. We take the value of the middle element as pivot value, but it can be any value, which is in range of sorted values, even if it doesn't present in the array.

Partition. Rearrange elements in such a way, that all elements which are lesser than the pivot go to the left part of the array and all elements greater than the pivot, go to the right part of the array. Values equal to the pivot can stay in any part of the array. Notice, that array may be divided in non-equal parts.

Sort both parts. Apply quicksort algorithm recursively to the left and the right parts.

---

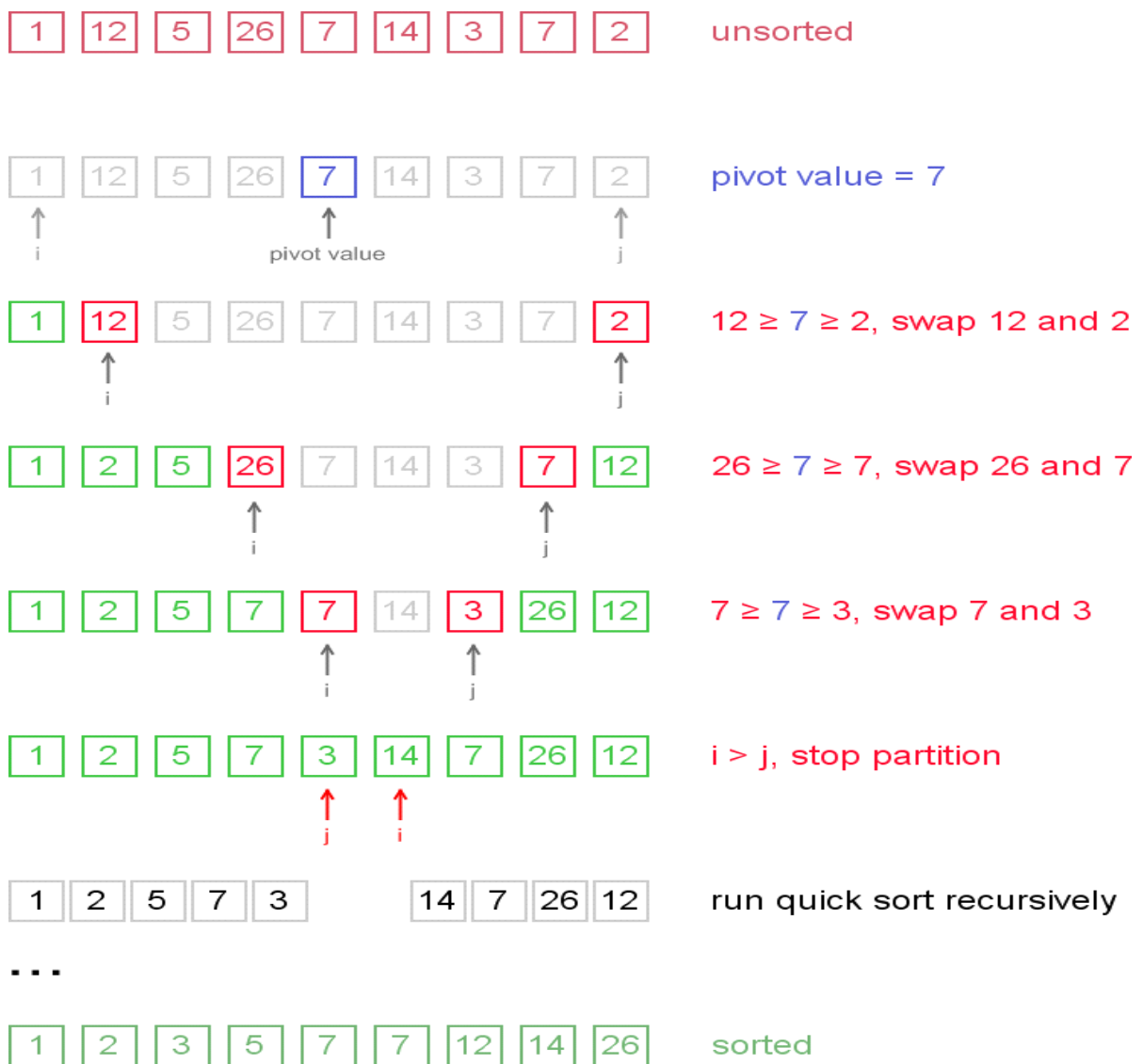


## Partition algorithm in detail

There are two indices  $i$  and  $j$  and at the very beginning of the partition algorithm  $i$  points to the first element in the array and  $j$  points to the last one. Then algorithm moves  $i$  forward, until an element with value greater or equal to the pivot is found. Index  $j$  is moved backward, until an element with value lesser or equal to the pivot is found. If  $i \leq j$  then they are swapped and  $i$  steps to the next position ( $i + 1$ ),  $j$  steps to the previous one ( $j - 1$ ). Algorithm stops, when  $i$  becomes greater than  $j$ .

After partition, all values before  $i$ -th element are less or equal than the pivot and all values after  $j$ -th element are greater or equal to the pivot.

Example. Sort  $\{1, 12, 5, 26, 7, 14, 3, 7, 2\}$  using quicksort.



**Time complexity:  $O(n \log n)$**

**Input:** Enter the first-year percentage of students

**Output:** Sorting by Quick sort

**Algorithm /Pseudo Code:**

```
void quickSort(int arr[], int left, int right) {  
    int i = left, j = right;  
    int tmp;  
    int pivot = arr[(left + right) / 2];  
  
    /* partition */  
    while (i <= j) {  
        while (arr[i] < pivot)  
            i++;  
        while (arr[j] > pivot)  
            j--;  
        if (i <= j) {  
            tmp = arr[i];  
            arr[i] = arr[j];  
            arr[j] = tmp;  
            i++;  
            j--;  
        }  
    }  
};  
  
/* recursion */  
if (left < j)  
    quickSort(arr, left, j);  
if (i < right)  
    quickSort(arr, i, right);
```

---

**Software required:** Open Source Python, Programming tool like Jupyter Notebook, PyCharm, Spyder.

**Conclusion:** Thus, we have studied the implementation of Quick Sort.



## Assignment no: 07

**Aim:** To understand and implement singly linked list

**Problem definition:** 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

- a) Add and delete the members as well as president or even secretary.
- b) Compute total number of members of club
- c) Display members
- d) Display list in reverse order using recursion
- e) Two linked lists exist for two divisions. Concatenate two lists.

### Learning objectives

To understand concept of sll.

To analyze the working of functions.

### Learning outcome

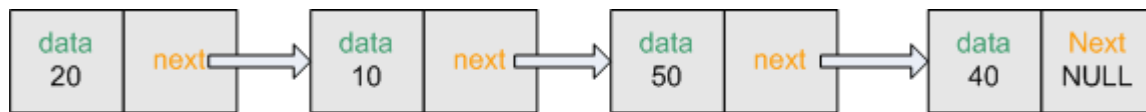
Students will be able to analyze problems to use variants of linked list and solve various real-life problems.

### Theory:

#### Linked list:

Linked list is one of the fundamental data structures, and can be used to implement other data structures. In a linked list there are different numbers of nodes. Each node consists of two fields. The first field holds the value or data and the second field holds the reference to the next node or null if the linked list is empty.

---



Linked list

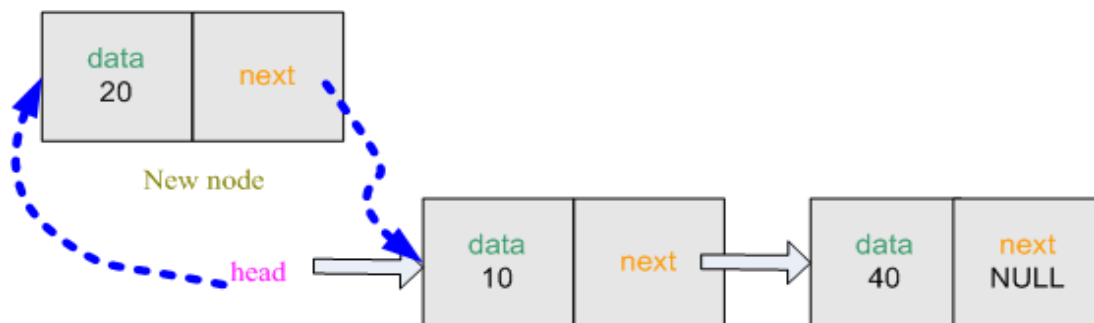
The singly-linked list is the easiest of the linked list, which has one link per node.

### Linked list operation:

First, we create a structure “node”. It has two members and first is int data which will store the information and second is node \*next which will hold the address of the next node. Linked list structure is complete so now we will create linked list. We can insert data in the linked list from 'front' and at the same time from 'back'. Now we will examine how we can insert data from front in the linked list.

#### 1) Insert from front:

At first initialize node type. Then we take the data input from the user and store in the node info variable. Create a temporary node node \*temp and allocate space for it. Then place info to temp->data. So the first field of the node \*temp is filled. Now temp->next must become a part of the remaining linked list (although now linked list is empty but imagine that we have a 2 node linked list and head is pointed at the front) So temp->next must copy the address of the \*head (Because we want insert at first) and we also want that \*head will always point at front. So \*head must copy the address of the node \*temp.



Linked list

## 2) Traverse

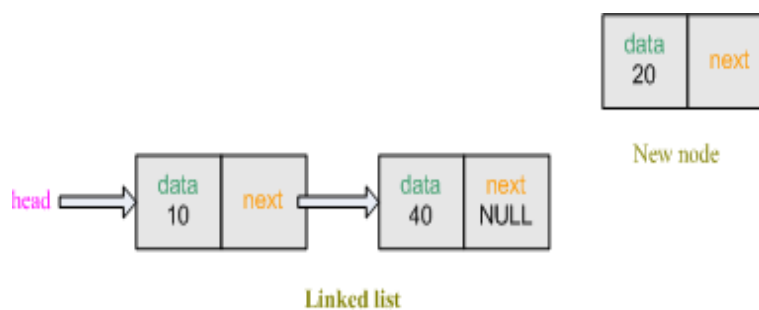
Now we want to see the information stored inside the linked list. We create node\*temp1. Transfer the address of \*head to \*temp1. So \*temp1 is also pointed at the front of the linked list. Linked list has 3 nodes. We can get the data from first node using temp1->data . To get data from second node, we shift \*temp1 to the second node. Now we can get the data from second node.

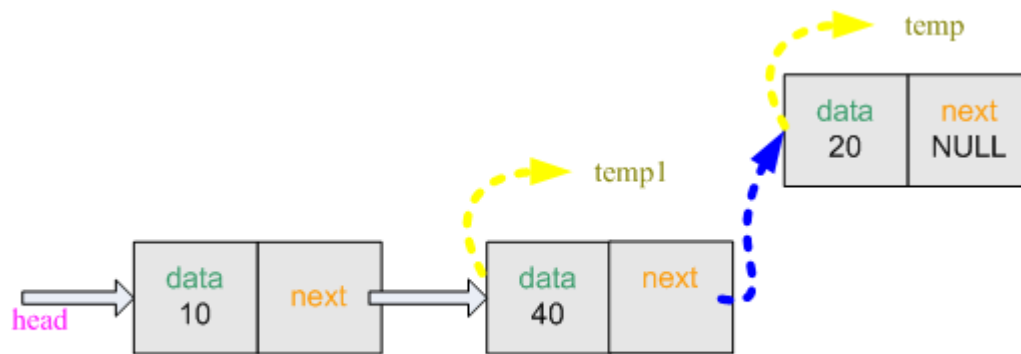


This process will run until the linked list's next is NULL.

## 3) Insert from back

Insert data from back is very similar to the insert from front in the linked list. Here the extra job is to find the last node of the linked list. Now, Create a temporary node node \*temp and allocate space for it. Then place info to temp->data, so the first field of the node node \*temp is filled. node \*temp will be the last node of the linked list. For this reason,temp->next will be NULL. To create a connection between linked list and the new node, the last node of the existing linked list node \*temp1's second field temp1->next is pointed to node \*temp.



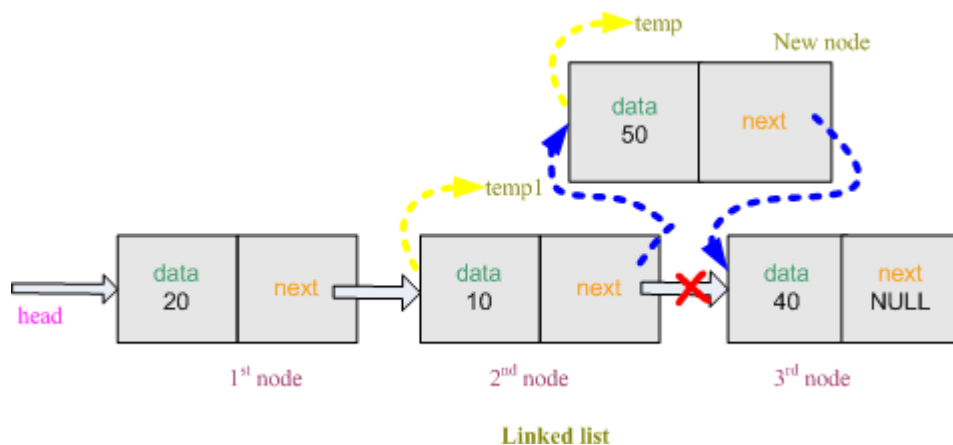


#### Linked list

#### 4) Insert after specified number of nodes

Insert data in the linked list after specified number of node is a little bit complicated. But the idea is simple. Suppose, we want to add a node after 2nd position. So, the new node must be in 3rd position. The first step is to go the specified number of node. Let, node \*temp1 is pointed to the 2nd node now. Now, Create a temporary node node \*temp and allocate space for it. Then place info to temp->next , so the first field of the node node \*temp is filled. To establish the connection between new node and the existing linked list, new node's next must pointed to the 2ndnode's (temp1) next

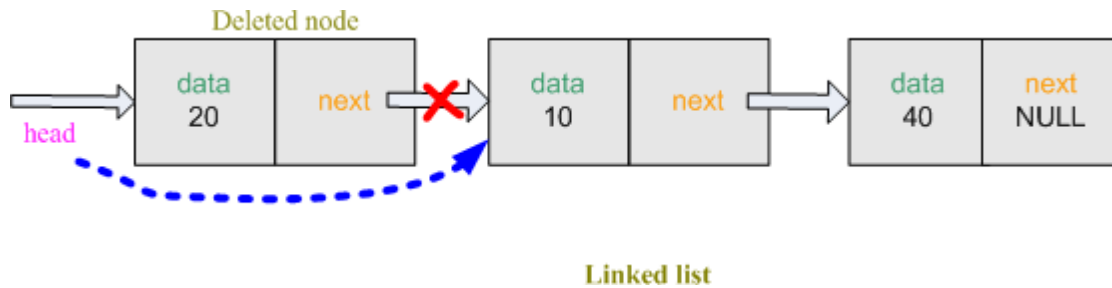
. The 2nd node's (temp1) next must pointed to the new node(temp).



#### 5) Delete from front:

Delete a node from linked list is relatively easy. First, we create node\*temp. Transfer the address of \*head to\*temp . So \*temp is pointed at the front of the linked list. We want to delete the first node. So transfer the address of temp->next tohead so that it now pointed to the second node. Now

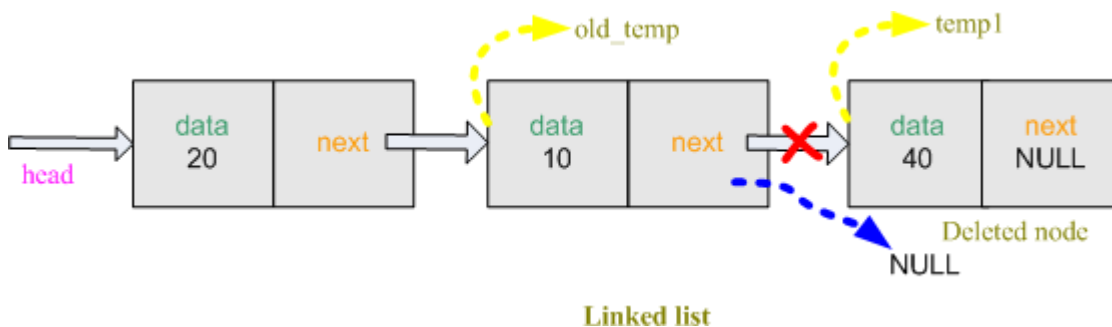
free the space allocated for first node.



### 6) Delete from back

The last node's next of the linked list always pointed to NULL. So when we will delete the last node, the previous node of last node is now pointed at NULL. So, we will track last node and previous node of the last node in the linked list. Create temporary node \*temp1 and \*old\_temp. Now node

\*temp1 is now pointed at the last node and \*old\_temp is pointed at the previous node of the last node. Now rest of the work is very simple. Previous node of the last node old\_temp will be NULL so it become the last node of the linked list. Free the space allocated for last node.

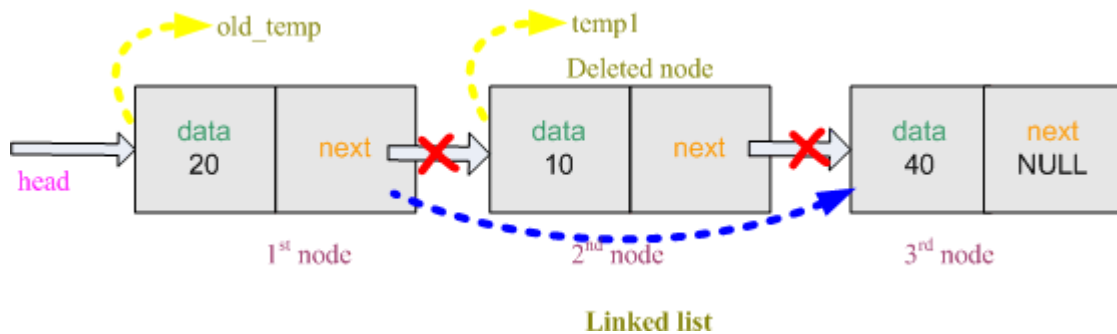


### 7) Delete specified number of node

To delete a specified node in the linked list, we also require to find the specified node and previous node of the specified node. Create temporary node \*temp1, \*old\_temp and allocate space for it. Take the input from user to know the number of the node. Now node \*temp1 is now pointed at the specified node and \*old\_temp is pointed at the previous node of the specified node. The previous node of the specified node must connect to the rest of the linked list so we transfer the address of



temp1->next to old\_temp->next. Now free the space allocated for the specified node.



### Input:

Enter members, president and secretary having their PRN and Name to be stored in the singly linked list. Every node would contain 2 fields: data and next pointer.

### Output:

Insertion of node at front, at end, at any given position, deletion of node, reverse of linked list, count total number of nodes, merge two linked list.

### Algorithm:

#### Algorithm to insert President at first node:

Step 1: p - pointer to a linked list

Step 2 : e - element to be added

Step 3: Getnode() returns a new node

$q = \text{Getnode}()$

$\text{info}(q) = e$

$\text{next}(q) = p$

$p = q$

return p

Step 4: end

**Algorithm to insert members at any position in link list:**

Step 1:  $p$  - pointer to a linked list

Step 2: Getnode() returns a new  
node

Step 3:  $x$  - key node after which  $e$  is inserted

Step 4:  $e$  - element to be added

$k = p$

while  $k \neq \text{NIL}$  and  $\text{info}(k) = x$  do // find the key node

$k = \text{next}(k)$  if  $k = \text{NIL}$  then

Write "Node not found" return  $p$

$q = \text{Getnode}()$

$\text{info}(q) = e$

$\text{next}(q) = \text{next}(k)$

$\text{next}(k) = q$

return  $p$

Step5: Stop

**Algorithm to insert secretary at Last node:**

Step 1:  $p$  - pointer to a linked list

Step 2: Getnode() returns a new

node Step 3:  $e$  - element to be added

if  $p = \text{NIL}$  then

return  $p$

$k = p$

while  $\text{next}(k) \neq \text{NIL}$  do // find the last node  $k = \text{next}(k)$

$q = \text{Getnode}()$

$\text{info}(q) = e$

$\text{next}(k) = q$

$\text{next}(q) = \text{NIL}$

return  $p$

Step 4: end .

---

### Algorithm to Delete any member Node

Step 1:  $p$  - pointer to linked list

Step 2:  $x$  - key node to be

deleted

Step 3:  $k$  and  $pred$  – temporary

variables Step 4:  $k = p$ ;  $pred = \text{NIL}$

while  $k \neq \text{NIL}$  and  $\text{info}(k) = x$  do // find the key node  $pred = k$

$k = \text{next}(k)$

if  $k = \text{NIL}$  then

Write "Node not found" else

if  $pred = \text{NIL}$  // only one node in the list then

$p = \text{next}(p)$  else

$\text{next}(pred) = \text{next}(k)$

return  $p$

end

DeleteNode.

Step 5: end

### Algorithm to Display the members of link list.

Step 1: Create link list

---

Step 2: Scan and print the entire link list of the members having their PRN and name one by one.

Step 3: end

Algorithm to Count the total number of members nodes in the link list.

Step 1: count = 0

Step 2: Increment count as each node is

traversed current = head

while (current != NULL) then

count++

Step 3: end

### **Algorithm to merge two linked list**

Step 1: enter two linked list

Step 2: after first list is traverse merge the second linked list.

Step 3: end

### **Algorithm to reverse the link list**

Step 1: Iterative list reverse. Iterate through the list left-right. Move/insert each node to the front of the result list like a Push of the node.

Step 2: result = NULL;

Step 3: while (current != NULL)

next = current->next ( note the next node current->next = result move the node onto the result)

result =

current current

= next Step 4 :

end

**Software required:** g++ / gcc compiler- / 64 bit fedora.

**Conclusion:** We understand & implement different operations on of linked list.

---

## Assignment no: 08

**Aim:** To understand and implement set operation using linked list.

**Problem definition:** Second year Computer Engineering class, set A of students like Vanilla Ice-cream and set B of students like butterscotch ice-cream. Write C/C++ program to store two sets using linked list. compute and display i. Set of students who like either vanilla or butterscotch or both ii. Set of students who like both vanilla and butterscotch iii. Set of students who like only vanilla not butterscotch iv. Set of students who like only butterscotch not vanilla v. Number of students who like neither vanilla nor butterscotch

### Learning objectives

To understand concept of set operations and linked list.

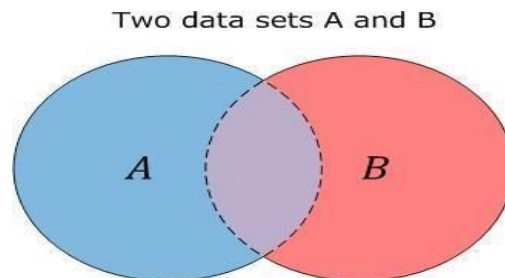
### Learning outcome

Students will be able to analyze problems to use variants of linked list and solve various real- life problems.

### Theory:

Set

**Elements** are the objects contained in a **set**. A set may be defined by a common property amongst the objects. For example, the set E of positive even integers is the set  $E = \{2, 4, 6, 8, 10, \dots\}$



**Definition (Union):** The union of sets  $A$  and  $B$ , denoted by  $A \cup B$ , is the set defined as

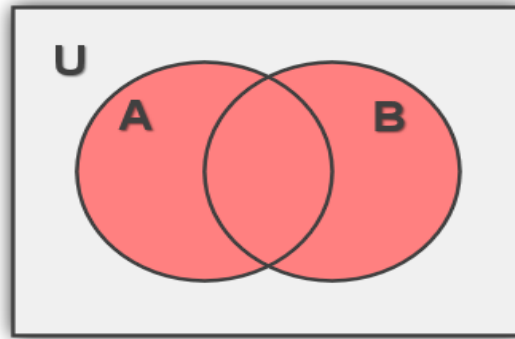
$$A \cup B = \{ x \mid x \in A \vee x \in B \}$$

---

Example 1: If  $A = \{1, 2, 3\}$  and  $B = \{4, 5\}$ , then  $A \cup B = \{1, 2, 3, 4, 5\}$ .

Example 2: If  $A = \{1, 2, 3\}$  and  $B = \{1, 2, 4, 5\}$ , then  $A \cup B = \{1, 2, 3, 4, 5\}$ .

Note that elements are not repeated in a set.



**Definition (Intersection):** The intersection of sets  $A$  and  $B$ , denoted by  $A \cap B$ , is the set defined as

$$A \cap B = \{x \mid x \in A \wedge x \in B\}$$

Example 3: If  $A = \{1, 2, 3\}$  and  $B = \{1, 2, 4, 5\}$ , then  $A \cap B = \{1, 2\}$ .

Example 4: If  $A = \{1, 2, 3\}$  and  $B = \{4, 5\}$ , then  $A \cap B = \emptyset$ .

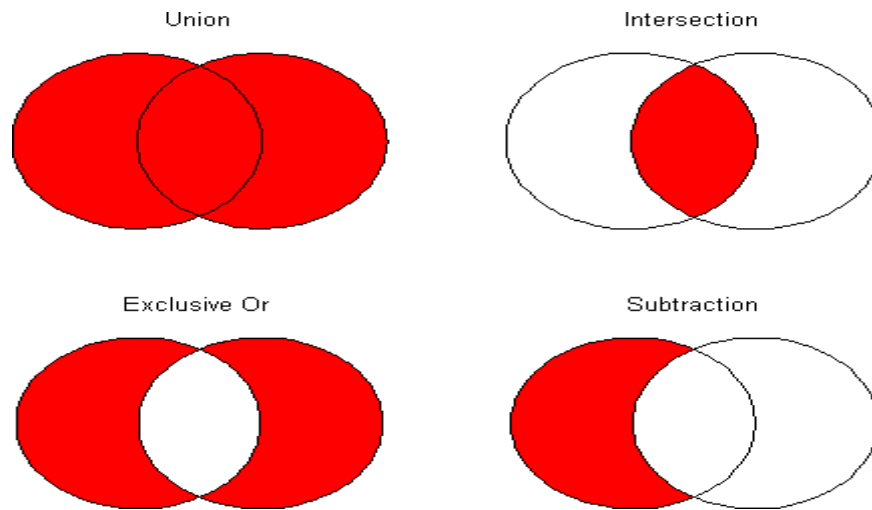
**Definition (Difference):** The difference of sets  $A$  from  $B$ , denoted by  $A - B$ , is the set defined as

$$A - B = \{x \mid x \in A \wedge x \notin B\}$$

Example 5: If  $A = \{1, 2, 3\}$  and  $B = \{1, 2, 4, 5\}$ , then  $A - B = \{3\}$ .

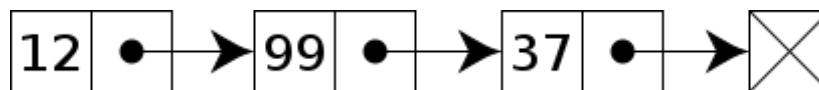
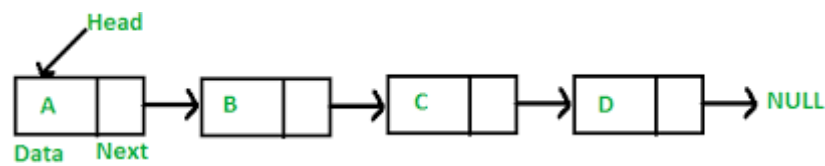
Example 6: If  $A = \{1, 2, 3\}$  and  $B = \{4, 5\}$ , then  $A - B = \{1, 2, 3\}$ .

Note that in general  $A - B \neq B - A$



### Singly linked list

Singly linked lists contain nodes which have a data field as well as a 'next' field, which points to the next node in line of nodes. Operations that can be performed on singly linked lists include insertion, deletion and traversal.



*A singly linked list whose nodes contain two fields: an integer value and a link to the next node*

### Union and Intersection of two Linked Lists

Input:

List1: 10->15->4->20

lsit2: 8->4->2->10

Output:

1



Intersection List: 4->10

Union List: 2->8->20->4->15->10

***Intersection (list1, list2):***

Initialize result list as NULL. Traverse list1 and look for its each element in list2, if the element is present in list2, then add the element to result.

***Union (list1, list2):***

Initialize result list as NULL. Traverse list1 and add all of its elements to the result. Traverse list2. If an element of list2 is already present in result then do not insert it to result, otherwise insert.

**Input:** enter set A of students like Vanilla Ice-cream and set B of students like butterscotch ice-cream.

**Output:** Intersection, Union, Neither nor.

**Algorithm:**

**Set of students who like both vanilla and butterscotch**

```
intersection()
{
    Node *cur1=Butter;
    Node *cur2=Vanilla;
    int found=0;
    while(cur1!=NULL)
    {
        while(cur2!=NULL)
        {
            if(cur1->data==cur2->data)
            {
                found=1;
                break;
            }
        }
    }
}
```

else

}

cur2=cur2->next;

if(found==1)

{

}

else

cout<<cur

1->dat

a<<" ";

cur1=c

ur1->n

ext;

found=

0;

cur1=cur1->next;

cur2=Vanilla;

}

}

### **Set of students who like only vanilla**

void onlyV()

{

Node \*cur1=Butter;

Node \*cur2=Vanilla; int

found=0;

while(cur2!=NULL)

{

while(cur1!=NULL)

{

if(cur2->data==cur1->data)

```

        {

        }

else

        }
        if(found==1)
        {

        }
        else
        {

        }

cur2=cur2->next; found=0;
cout<<cur2->data<<" "; cur2=cur2->next;

        cur1=Butter;
    }
}

```

### **Set of students who like only butterscotch**

```

void onlyB()
{
    Node *cur1=Butter;
    Node *cur2=Vanilla; int
    found=0;
    while(cur1!=NULL)
    {

```

```

while(cur2!=NULL)
{
    if(cur1->data==cur2->data)
        found=1;break;
    {

        cur2=cur2->next
    }
    else{

    }

}
if(found==1)
{

}
else
{

}

    cur1=cur1->next; found=0;

    cout<<cur1->data<<" ";
    cur1=cur1->next;

    cur2=Vanilla;
}
}

```

---

**Set of students who like either vanilla or butterscotch or both**

```
void uni()
{
    onlyB();
    intersection();
    onlyV();
}
```

**Number of students who like neither vanilla nor butterscotch**

```
void neither()
{
    cout<<"\nstudents who like neither vanila nor butterscotch\n";
    temp=h1;
    while(temp!=NULL)
    { temp3=head3;
      f=0;
      while(temp3!=NULL)
```

## Data Structure Laboratory Manual (210246)

```
{ if(temp->roll==temp3->roll)
    { f=1;          }
    temp3=temp3->next;
}
```

```
if(f==0)
{ cout<<"\n"<<temp->roll;  }
  temp=temp->next;
}
```

```
}
```

**Software required:** g++ / gcc compiler- / 64 bit fedora.

**Conclusion:** We understand & implement different operations on of linked list.

## Assignment No- 09

**Aim:** To illustrate the concept of stack and 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 the same 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 check whether given string is palindrome or not that uses a stack to determine whether a string is a palindrome.
2. To remove spaces and punctuation in string, convert all the Characters to lowercase, and then call above Palindrome checking function to check for a palindrome
3. To print string in reverse order using stack

### Learning Objectives:

To understand concept of stack and string. To analyze the string is palindrome or not.

**Learning Outcome:** Students will be able to implement stack and queue data structures and algorithms for solving different kinds of problems.

### Theory:

#### Stack

It is named stack as it behaves like a real-world stack, for example – deck of cards or pile of plates etc. A stack is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and pop, which removes the most recently added element that was not yet removed. The order in which elements come off a stack gives rise to its alternative name, LIFO (for last in, first out). Additionally, a peek operation may give access to the top without modifying the stack.

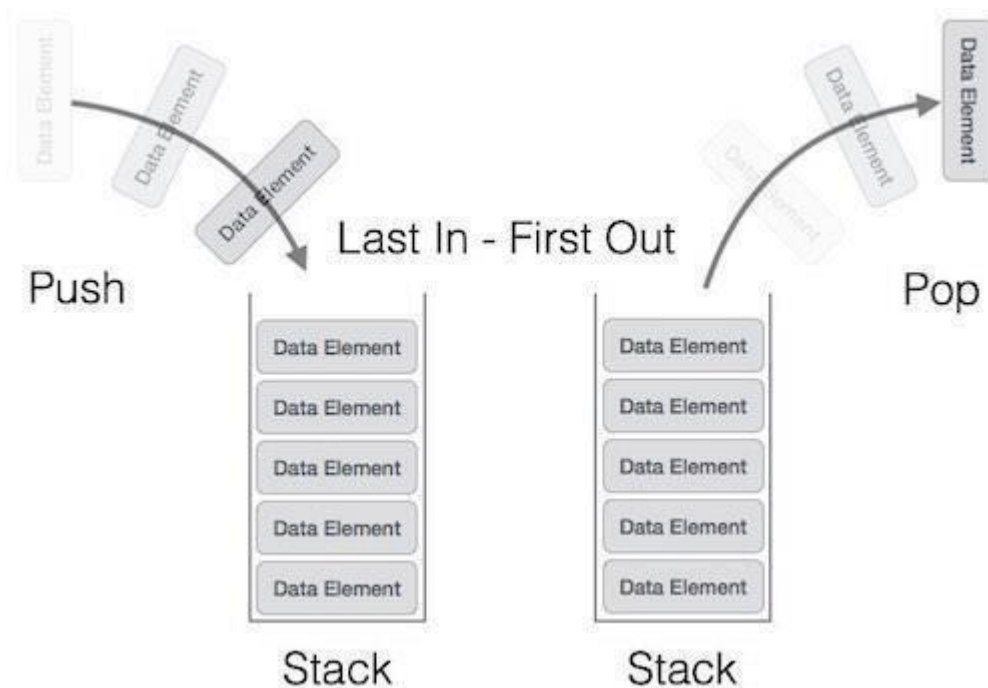
---

A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, We can only access the top element of a stack.

This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last, is accessed first. In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation.

### StackRepresentation

Below given diagram tries to depict a stack and its operations –



A stack can be implemented by means of Array, Structure, Pointer and Linked-List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays which makes it a fixed size stack implementation. Basic Operations Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations –

**push()** – pushing (storing) an element on the stack.

**pop()** – removing (accessing) an element from the stack.

When data is PUSHed onto stack.



To use a stack efficiently we need to check status of stack as well. For the same purpose, the following functionality is added to stacks –

**peek()** – get the top data element of the stack, without removing it.

**isFull()** – check if stack is full.

**isEmpty()** – check if stack is empty.

At all times, we maintain a pointer to the last PUSHed data on the stack. As this pointer always represents the top of the stack, hence named **top**. The **top** pointer provides top value of the stack without actually removing it.

**Input:**

Enter the string.

**Output:**

Entered string is palindrome or not

**Algorithm:**

Step 1: Input S (string) Step

2: Len = 0 , Flag =0

Step 3: While (S[Len] != NULL)

Len++ Step

4: I = 0 , J = Len-1

Step 5: While ( I < (Len/2)+1 )

If ( S[I] == S[J] )

Flag=0

else

Flag=1

I++ , J--

Step 6: If ( Flag == 0 )

Print Key Is a Palindrome

Step 7: End

Print Key Is Not a Palindrome

**Software required:** g++ / gcc compiler- / 64 bit fedora.

**Conclusion :** Thus we have studied the implementation of string is palindrome or not

## Assignment No- 10

**Aim:** To illustrate the various concept of stack.

**Problem Statement:** In any language program mostly syntax error occurs due to unbalancing delimiter such as (), {}, []. Write C++ program using stack to check whether given expression is well parenthesized or not.

### Learning Objectives:

To understand concept of stack.

To analyze the working of various functions of stack.

**Learning Outcome:** Students will be able to implement stack and queue data structures and algorithms for solving different kinds of problems.

### Theory:

#### Stack

A stack is a container of objects that are inserted and removed according to the last-in first-out (LIFO) principle.

In the pushdown stacks only two operations are allowed:

**push** the item into the stack, and

**pop** the item out of the stack.

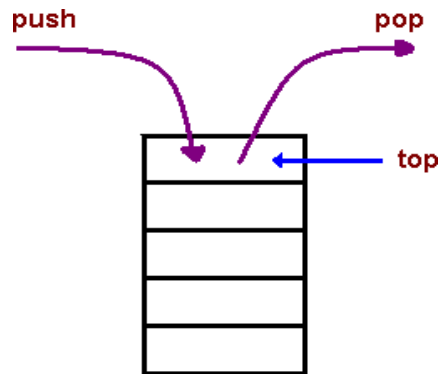
A stack is a limited access data structure - elements can be added and removed from the stack only at the top. **push** adds an item to the top of the stack, **pop** removes the item from the top. A helpful analogy is to think of a stack of books; you can remove only the top book, also you can add a new book on the top.

A stack is a **recursive** data structure. Here is a structural definition of a Stack:

a stack is either empty or

it consists of a top and the rest which is a stack;

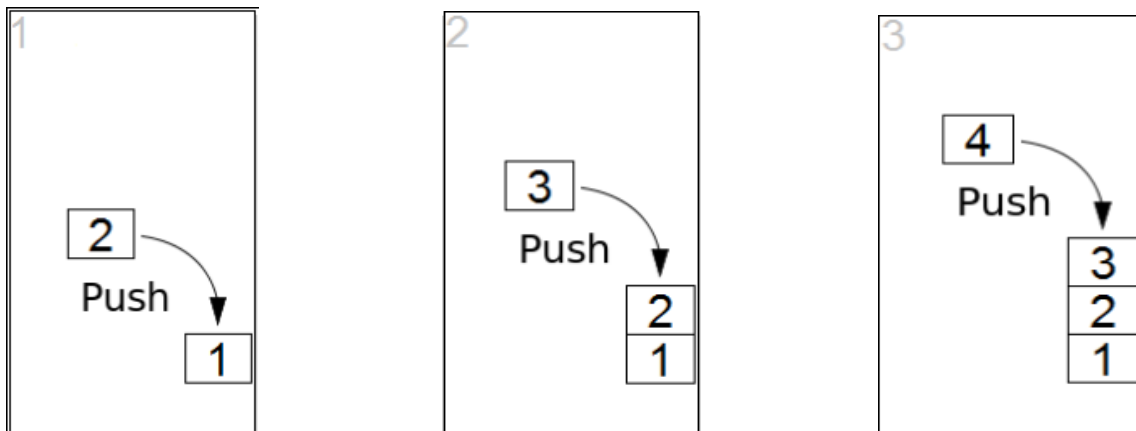
---

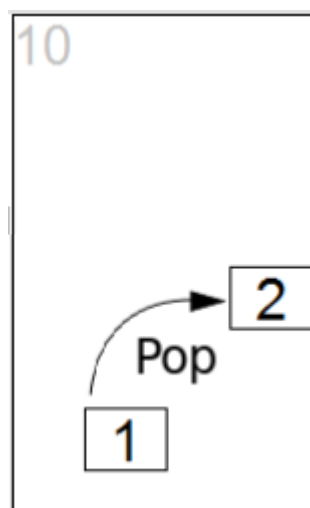
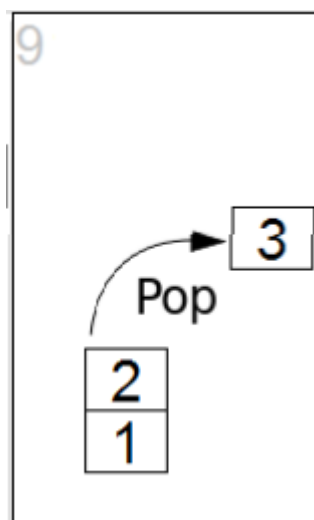
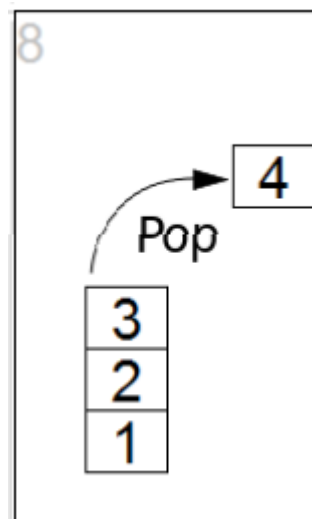
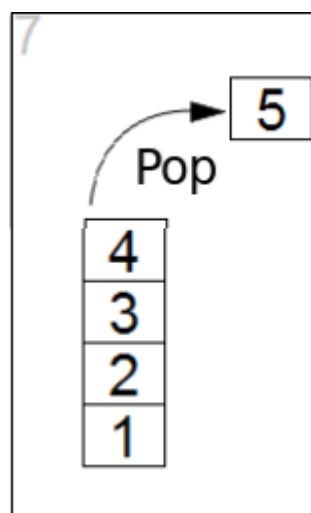
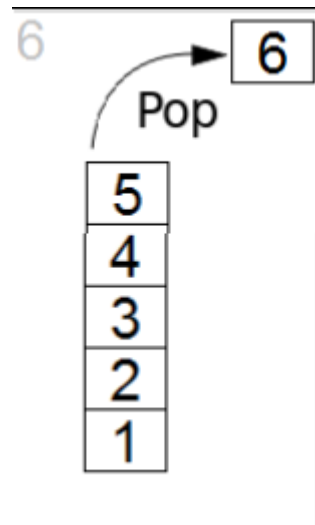
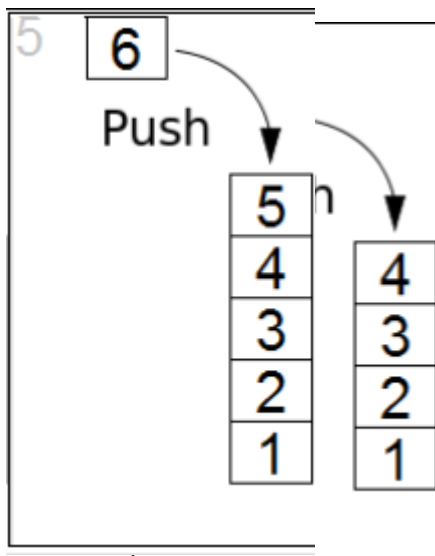


A **stack** is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and pop, which removes the most recently added element that was not yet removed. The order in which elements come off a stack gives rise to its alternative name, **LIFO** (for **last in, first out**). Additionally, a peek operation may give access to the top without modifying the stack.

The name "stack" for this type of structure comes from the analogy to a set of physical items stacked on top of each other, which makes it easy to take an item off the top of the stack, while getting to an item deeper in the stack may require taking off multiple other items first.<sup>[1]</sup>

Considered as a linear data structure, or more abstractly a sequential collection, the push and pop operations occur only at one end of the structure, referred to as the top of the stack. A stack may be implemented to have a bounded capacity. If the stack is full and does not contain enough space to accept an entity to be pushed, the stack is then considered to be in an overflow state. The pop operation removes an item from the top of the stack.





## Balanced parentheses

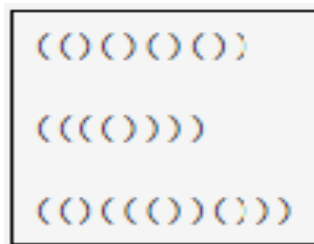
We now turn our attention to using stacks to solve real computer science problems.

You have no doubt written arithmetic expressions such as

$(5+6)*(7+8)/(4+3)(5+6)*(7+8)/(4+3)$

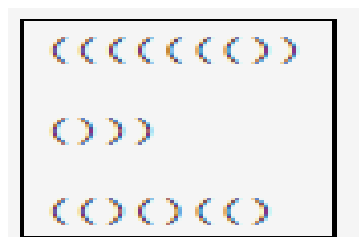
where parentheses are used to order the performance of operations.

**Balanced parentheses** means that each opening symbol has a corresponding closing symbol and the pairs of parentheses are properly nested. Consider the following correctly balanced strings of parentheses:



Three examples of balanced parentheses strings are shown, each enclosed in a box. The first box contains the string `()()()()`. The second box contains the string `((((()))))`. The third box contains the string `(()((())()()))`.

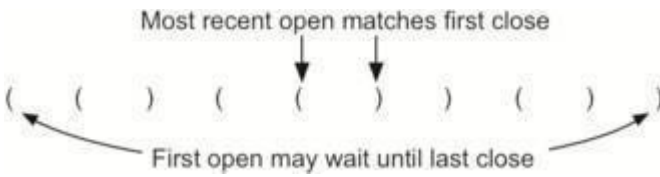
Compare those with the following, which are not balanced:



Three examples of unbalanced parentheses strings are shown, each enclosed in a box. The first box contains the string `(((((((('))`. The second box contains the string `()))`. The third box contains the string `(()()()()`.

The ability to differentiate between parentheses that are correctly balanced and those that are unbalanced is an important part of recognizing many programming language structures.

The challenge then is to write an algorithm that will read a string of parentheses from left to right and decide whether the symbols are balanced. To solve this problem we need to make an important observation. As you process symbols from left to right, the most recent opening parenthesis must match the next closing symbol. Also, the first opening symbol processed may have to wait until the very last symbol for its match. Closing symbols match opening symbols in the reverse order of their appearance; they match from the inside out. This is a clue that stacks can be used to solve the problem.

**Input:**

Enter any expression having number of opening & closing brackets.

**Output:**

Entered expression is well parenthesized or not.

**Algorithm:**

Step 1: Declare a character stack S.

Step 2: Now traverse the expression string exp.

If the current character is a starting bracket ('(' or '{' or '[') then  
push it to stack.

If the current character is a closing bracket (')' or '}' or ']') then  
pop from stack and  
if the popped character is the matching starting bracket  
then

fine

else

parenthesis are not balanced

Step 4: After complete traversal, if there is some starting bracket left in stack then  
“not balanced”

Step 5:Exit

**Software required:** g++ / gcc compiler- / 64 bit fedora.

**Conclusion:** Thus we have studied the implementation of expression is well parenthesized or not using stack.

## Assignment No - 11

**Aim:** To illustrate the concept of 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.

### Learning Objectives:

To understand concept of queue.

To analyze the various functions of queue.

**Learning Outcome:** Students will be able to implement stack and queue data structures and algorithms for solving different kinds of problems.

### Theory:

A queue is logically a first in first out (FIFO or first come first serve) linear data structure. The concept of queue can be understood by our real life problems. For example a customer come and join in a queue to take the train ticket at the end (rear) and the ticket is issued from the front end of queue. That is, the customer who arrived first will receive the ticket first. It means the customers are serviced in the order in which they arrive at the servicecentre. It is a homogeneous collection of elements in which new elements are added at one end called rear, and the existing elements are deleted from other end called front. The basic operations that can be performed on queue are A minimal set of operations on a queue is as follows:

1. create()—creates an empty queue,  $Q$
  2. add( $i, Q$ )—adds the element  $i$  to the rear end of the queue,  $Q$  and returns the new queue
  3. delete( $Q$ )—takes out an element from the front end of the queue and returns the resulting queue
  4. getFront( $Q$ )—returns the element that is at the front position of the queue
  5. Is\_Empty( $Q$ )—returns true if the queue is empty; otherwise returns false
-



Push operation will insert (or add) an element to queue, at the rear end, by incrementing the array index. Pop operation will delete (or remove) from the front end by decrementing the array index and will assign the deleted value to a variable. Total number of elements present in the queue is  $\text{front} - \text{rear} + 1$ , when implemented using arrays. Following figure will illustrate the basic operations on queue.



Fig. 4.1. Queue is empty.

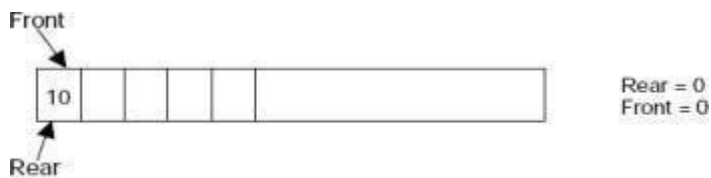


Fig. 4.2. push(10)

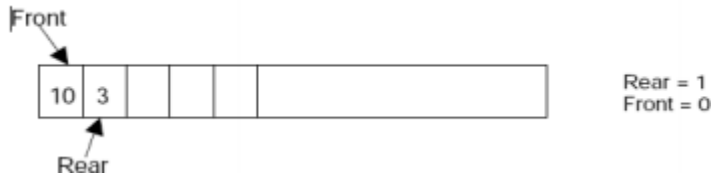


fig 4.3 push(3)

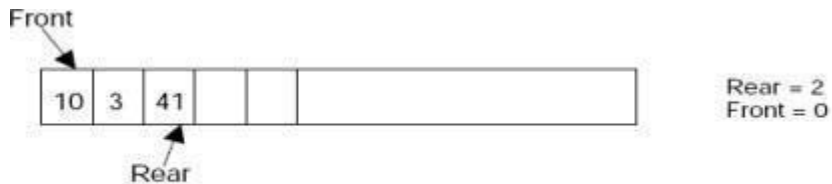


Fig. 4.4. push(41)

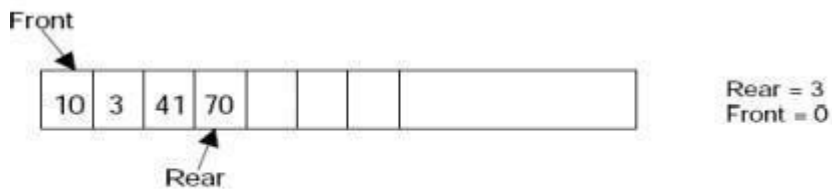


Fig. 4.5. push(70)

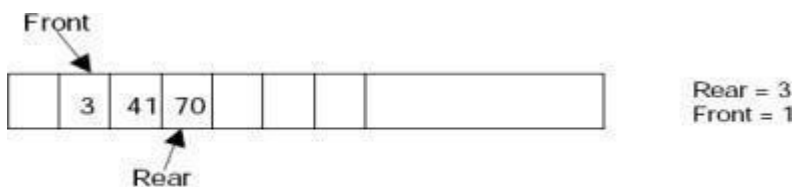


Fig. 4.6.  $x = \text{pop}()$  (i.e.;  $x = 10$ )

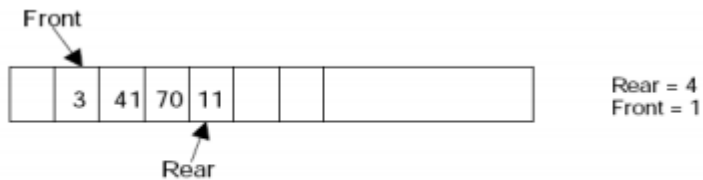


Fig. 4.7. push(11)

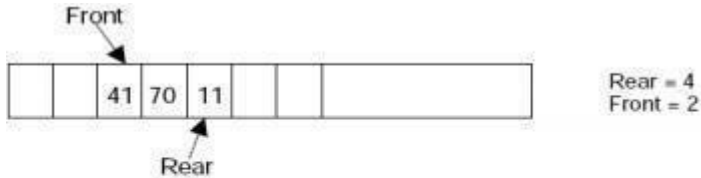


Fig. 4.8.  $x = \text{pop}()$  (i.e.;  $x = 3$ )

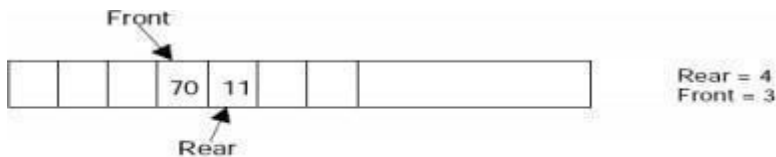


Fig. 4.9.  $x = \text{pop}()$  (i.e.,  $x = 41$ )



Fig. 4.10

**Input:** Enter the jobs to an operating system

**Output:** Add jobs and delete jobs from queue.

**Algorithm:**

#### Algorithm to define class

Step 1: class queue(element) Step

2: declare create() A queue

Step 3: add(element, queue) A queue

Step 4: delete(queue) A queue

Step 6: getFront(queue) A queue Step

7: Is\_Empty(queue) A Boolean;

Step 8: For all Q belongs to queue, i belongs to element let Step

9: Is\_Empty(create()) = true

Step 10: Is\_Empty(add(i,Q)) = false

Step 11: delete(create()) = error

```
Step 12:delete(add(i,Q)) =  
    if Is_Empty(Q)  
    then  
        create  
    else  
        add(i, delete(Q))
```

Step 13:getFront(create) = error

```
Step 14:getFront(add(i, Q)) =  
    if Is_Empty(Q)  
    then  
        i  
    else  
        getFront(Q)  
    end
```

Step 15:End

#### **Algorithm to create an empty queue**

Step 1: Initialize queue Front =

Rear = -1;

Step 2: End

#### **Algorithm to checks whether the queue is empty or not**

Step 1: Is\_Empty()

Step 2: if(Front == Rear)

return 1;

else

return 0;

Step 3: End End

#### **Algorithm to checks whether queue is full or not.**

Step 1:Is\_Full()

tep 2: if(Rear == max - 1)

---

else

End Step 3: End

return 0;

### **Algorithm to adds an element in the queue**

Step 1: Add(int Element)

Step 2:if(Is\_Full())

then

print Error, Queue is full Queue[++Rear] = Element;

else

End

Step 3: End

### **Algorithm to deletes an element from the front of the queue**

Step 1: Delete()

Step 2:if(Is\_Empty())

then

print Sorry, queue is Emptyreturn(Queue[++Front]);

else

End

Step 3: End

### **Algorithm to returns the element at the front**

Step 1:getFront() Step

2:if(Is\_Empty())

then

print Sorry, queue is Empty return(Queue[Front + 1])

elseEnd

Step3: End

**Software required:** g++ / gcc compiler- / 64 bit fedora.

**Conclusion :** Thus we have studied the implementation of queue.

---

## Assignment No-12

**Aim:** To illustrate the concept of double-ended queue (deque).

**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.

### Learning Objectives:

To understand concept of double-ended queue (deque).

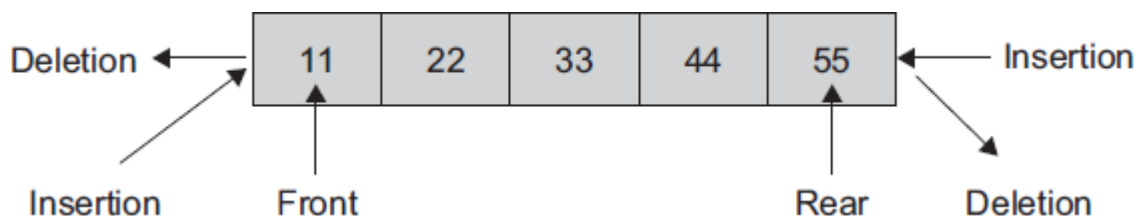
To analyze the various functions of double-ended queue (deque).

**Learning Outcome:** Students will be able to implement stack and queue data structures and algorithms for solving different kinds of problems.

### Theory:

#### DeQueue:

Deque is a data structure in which elements may be added to or deleted from the front or the rear. Like an ordinary queue, a double-ended queue is a data structure it supports the following operations: `enq_front`, `enq_back`, `deq_front`, `deq_back`, and `empty`. Dequeue can 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`. The DeQueue is represented as follows.



The output restricted Dequeue allows deletions from only one end and input restricted Dequeue allow insertions at only one end.

**Input :** Enter the elements in dequeue.

---

**Output:** Add elements from either end, delete elements from both ends.

**Algorithm:**

**Algorithm to add an element into DeQueue :**

Assumptions: pointer f,r and initial values are -1,-1

Q[] is an array

max represent the size of a queue

**Algorithm to add an element from front :**

step1. Start

step2. Check the queue is full or not as if  $(f \geq \max)$

step3. If false update the pointer f as  $f = f - 1$

step4. Insert the element at pointer f as  $Q[f] = \text{element}$  step5.

Stop

**Algorithm to add an element from back :**

step1. Start

step2. Check the queue is full or not as if  $(r == \max - 1)$  if yes queue is full step3.

If false update the pointer r as  $r = r + 1$

step4. Insert the element at pointer r as  $Q[r] = \text{element}$

step5. Stop

**Algorithm to delete an element from the DeQueue**

**Algorithm to delete an 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  $f = f + 1$  and delete element at position f as  $\text{element} = Q[f]$

step4. If  $(f == r)$  reset pointer f and r as  $f = r = -1$

step5. Stop

---

## Data Structure Laboratory Manual (210246)

### **Algorithm to delete an element from back :**

- 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
- step5. If (f == r ) reset pointer f and r as f = r = -1
- step6. Stop

**Software required:** g++ / gcc compiler.

**Conclusion:** Thus, we have studied the implementation of double-ended queue(deque).



## Assignment No - 13

**Aim:** To illustrate the concept of circular queue.

**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.

### Learning Objectives:

To understand concept of circular queue.

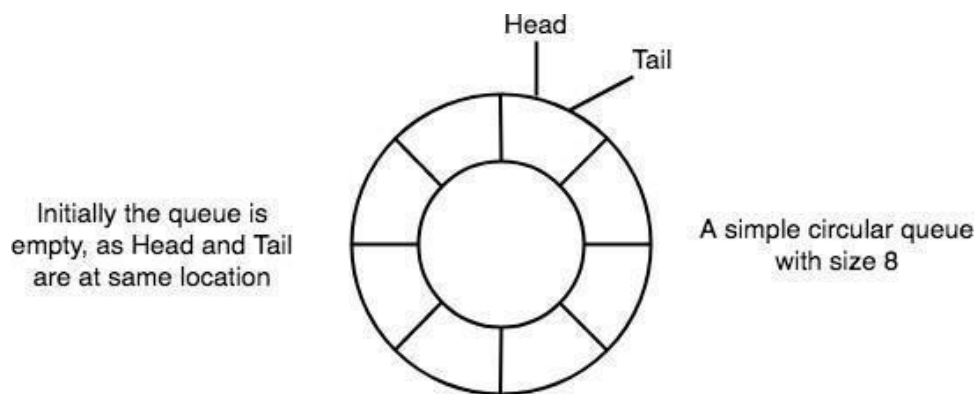
To analyze the various functions of circular queue.

**Learning Outcome:** Students will be able to implement stack and queue data structures and algorithms for solving different kinds of problems.

### Theory:

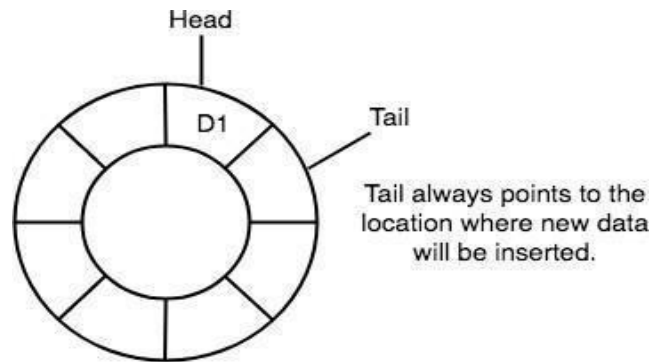
Circular Queue:

1. In case of a circular queue, head pointer will always point to the front of the queue, and tail pointer will always point to the end of the queue.
2. Initially, the head and the tail pointers will be pointing to the same location, this would mean that the queue is empty.

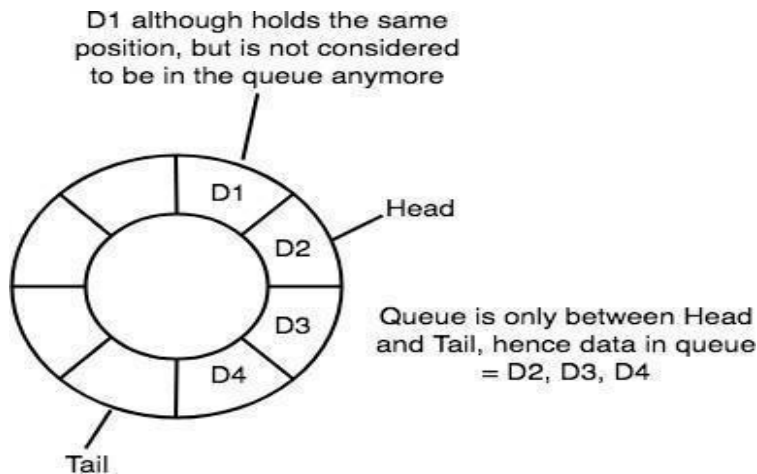


3. New data is always added to the location pointed by the tail pointer, and once the data is added, tail pointer is incremented to point to the next available location.

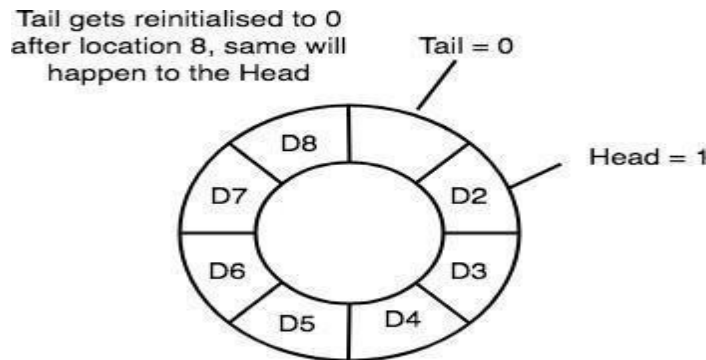




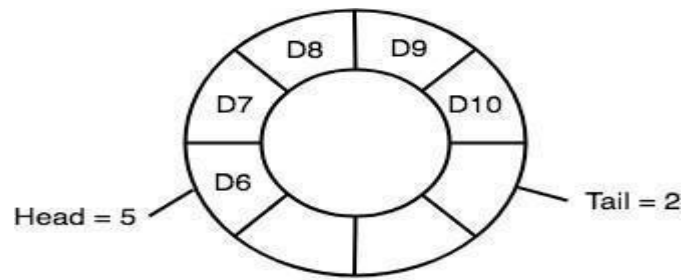
4. In a circular queue, data is not actually removed from the queue. Only the head pointer is incremented by one position when dequeue is executed. As the queue data is only the data between head and tail, hence the data left outside is not a part of the queue anymore, hence removed.



5. The head and the tail pointer will get reinitialized to 0 every time they reach the end of the queue



6. Also, the head and the tail pointers can cross each other. In other words, head pointer can be greater than the tail. Sounds odd? This will happen when we dequeue the queue a couple of times and the tail pointer gets reinitialized upon reaching the end of the queue.



In such a situation the value of the Head pointer will be greater than the Tail pointer

**Input:** Enter the orders of Pizza Parlor.

**Output:** Add orders and serve orders of pizza.

**Algorithm:** Implementation of circular queue

1. Initialize the queue, with size of the queue defined (maxSize), and head and tail pointers.
2. enqueue: Check if the number of elements is equal to  $\text{maxSize} - 1$ :
  - o If Yes, then return Queue is full.
  - o If No, then add the new data element to the location of tail pointer and increment the tail pointer.
3. dequeue: Check if the number of elements in the queue is zero:
  - o If Yes, then return Queue is empty.
  - o If No, then increment the head pointer.
4. Finding the size:
  - o If,  $\text{tail} \geq \text{head}$ ,  $\text{size} = (\text{tail} - \text{head}) + 1$
  - o But if,  $\text{head} > \text{tail}$ , then  $\text{size} = \text{maxSize} - (\text{head} - \text{tail}) + 1$

**Software required:** g++ / gcc compiler- / 64 bit fedora.

**Conclusion :** Thus we have studied the implementation of circular queue.