Assignment no: 01

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

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.

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

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

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.

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∩B, A · B or AB.

Difference Operation: It is a generalization of the idea of the compliment of a set and as a such is sometimes called the relative compliment 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.

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 o B and is defined by A o B = (A – B) U (B – A).

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])
```

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

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.

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

- **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))
```

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.

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

Assignment no: 03

Aim: To study and understand the concept of matrix realization using Python.

Problem definition:

Write a Python program to compute following computation on matrix:

a) Addition of two matrices

b) Subtraction of two matrices

c) Multiplication of two matrices

d) Transpose of a matrix

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

Theory:

Matrix:

We say a matrix is m × n if it has m rows and n columns. These values are sometimes called the dimensions of the matrix. Note that, in contrast to Cartesian coordinates, we specify the number of rows (the vertical dimension) and then the number of columns (the horizontal dimension). In most contexts, the rows and columns are numbered starting with 1. Several programming APIs, however, index rows and columns from 0. We use aij to refer to the entry in i th row and j th column of the matrix A.

The Order of a Matrix is its size or dimensions. The order is always given as the number of rows by the number of columns (R x C).

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$ This matrix is a 3 x 2 matrix. (rows by columns)

$$\begin{bmatrix} 1 & 5 & 6 & 7 & 8 \\ -2 & 3 & 5 & 4 & -1 \end{bmatrix}$$ 2 x 5 matrix

For two matrices to be added or subtracted, the dimensions must be the same. If they are the same, then the corresponding entries are added or subtracted whichever the operation.

$$\begin{bmatrix} 2 & 5 \\ 3 & -6 \\ 4 & 0 \end{bmatrix} + \begin{bmatrix} -1 & 7 \\ 2 & 4 \\ -4 & 5 \end{bmatrix} = \begin{bmatrix} 1 & 12 \\ 5 & -2 \\ 0 & 5 \end{bmatrix}$$

For two matrices to be multiplied, their dimensions need to be analyzed to determine if it is possible.

The number of columns of the first matrix MUST EQUAL the number of rows of the second matrix

Input: Enter the data for first matrix and second matrix.

Output: addition, subtraction & multiplication of entered matrix and transpose of matrix.

Conclusion: thus, we have studied and implemented the matrix and performed different operations on it.

Assignment no: 04

Aim: To study and understand the concept of magic square operation in a matrix in Python .

Problem Satement: Write a **Python** Program for magic square. A magic square is an n * n matrix of the integers 1 to n2 such that the sum of each row, column, and diagonal is the same. The figure given below is an example of magic square for case n=5. In this example, the common sum is 65.

15	8	1	24	17
16	14	7	5	23
22	20	13	6	4
3	21	19	12	10
9	2	25	18	11

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

Theory:

A magic square of order n is an arrangement of n2 numbers, usually distinct integers, in a square, such that the n numbers in all rows, all columns, and both diagonals sum to the same constant. A magic square contains the integers from 1 to n2.

The constant sum in every row, column and diagonal are called the magic constant or magic sum, M. The magic constant of a normal magic square depends only on n and has the following value:

$M = n(n2+1)/2$

Three conditions hold:

1. The position of next number is calculated by decrementing row number of the previous number by 1, and incrementing the column number of the previous number by 1. At any time, if the calculated row position becomes -1, it will wrap around to n-1. Similarly, if the calculated column position becomes n, it will wrap around to 0.
2. If the magic square already contains a number at the calculated position, calculated column position will be decremented by 2, and calculated row position will be incremented by 1.
3. If the calculated row position is -1 & calculated column position is n, the new position would be: (0, n-2).

Input: Enter the odd number .

Output: Magic Square is generated

Conclusion: thus, we have studied and implemented magic square in python.

Assignment no: 05

Aim: To study and understand the concept of Searching.

Problem Statement: Write a Python program to maintain club members, sort on roll numbers in ascending order. Write function "Ternary_Search" to search whether particular student is member of club or not. Ternary search is modifie binary search that divides array into 3 halves instead of two.

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

Theory:

Ternary search is a searching algorithm that is used to find the position of a target value within a sorted array. It operates on the principle of dividing the array into three parts instead of two, as in binary search. The basic idea is to narrow down the search space by comparing the target value with elements at two points that divide the array into three equal parts.

mid1 = l + (r-l)/3

mid2 = r – (r-l)/3

Working of Ternary Search:

The concept involves dividing the array into three equal segments and determining in which segment the key element (the element being sought) is located. It works similarly to a binary search, with the distinction of reducing time complexity by dividing the array into three parts instead of two.

Below are the step-by-step explanation of working of Ternary Search:

1. **Initialization:**
   - Begin with a sorted array.
   - Set two pointers, **left** and **right**, initially pointing to the first and last elements of the array.
2. **Divide the Array:**
   - Calculate two midpoints, **mid1** and **mid2**, dividing the current search space into three roughly equal parts:
   - mid1 = left + (right – left) / 3
   - mid2 = right – (right – left) / 3
   - The array is now effectively divided into **[left, mid1]**, **(mid1, mid2**), and **[mid2, right]**.
3. **Comparison with Target:**.
   - If the **target** is equal to the element at **mid1** or **mid2**, the search is successful, and the index is returned
   - If the **target** is less than the element at **mid1**, update the **right** pointer to **mid1 – 1**.
   - If the **target** is greater than the element at **mid2**, update the **left** pointer to **mid2 + 1**.
   - If the **target** is between the elements at **mid1** and **mid2**, update the **left** pointer to **mid1 + 1** and the **right** pointer to **mid2 –** 1.
4. **Repeat or Conclude:**
   - Repeat the process with the reduced search space until the **target** is found or the search space becomes empty.
   - If the search space is empty and the **target** is not found, return a value indicating that the **target** is not present in the array.

Input: Enter the list and the element to be searched.

Output: Searched element is found and displayed.

Conclusion: thus, we have studied and implemented ternary search in python.

Assignment no: 06

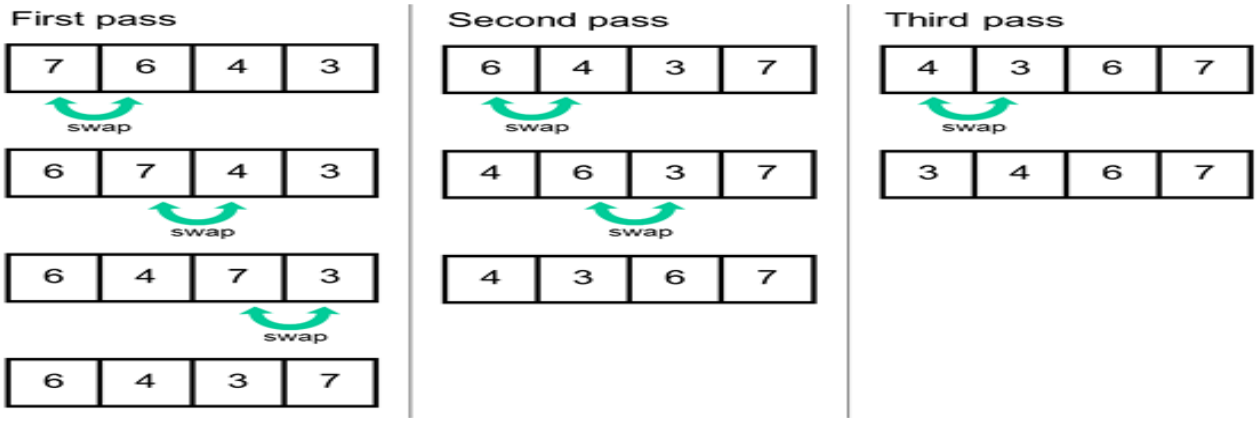
Aim: Sorting of an array using **selection and bubble sort.**

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.

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(n2) sorting algorithms, which makes it quite inefficient for sorting large data volumes. Bubble sort is **stable** and **adaptive**.



Selection Sort

Selection sort is one of the O(n2) sorting algorithms, which makes it quite inefficient for sorting large data volumes. Selection sort is notable for its programming simplicity and it can over perform 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
```

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
```

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

Assignment no: 07

Aim: To understand and implement singly linked list

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

a) Add and delete the members as well as president or even secretary.

b) Compute total number of members of club

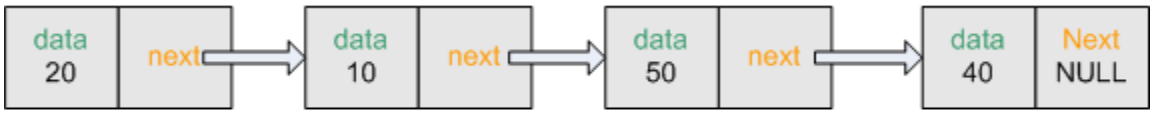
c) Display members

d) Display list in reverse order using recursion

e) Two linked lists exist for two divisions. Concatenate two lists.

Theory:

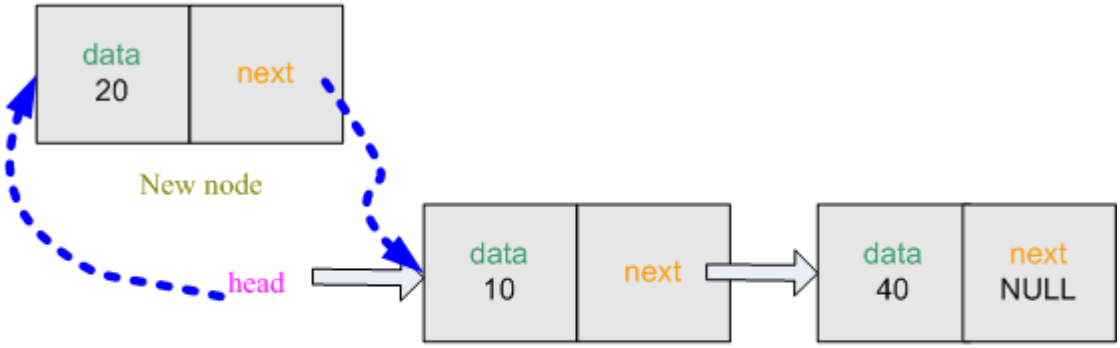
Linked list: Linked list is one of the fundamental data structures, and can be used to implement other data structures. In a linked list there are different numbers of nodes. Each node is consists of two fields. The first field holds the value or data and the second field holds the reference to the next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



Linked list

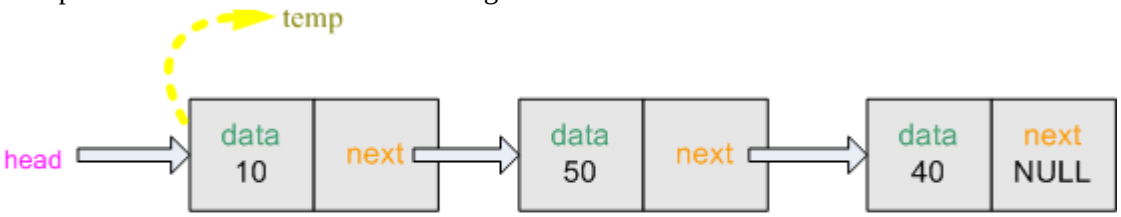
The singly-linked list is the easiest of the linked list, which has one link per node. Linked list operation: First, we create a structure “node”. It has two members and first is int data which will store the information and second is node *next which will hold the address of the next node. Linked list 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.



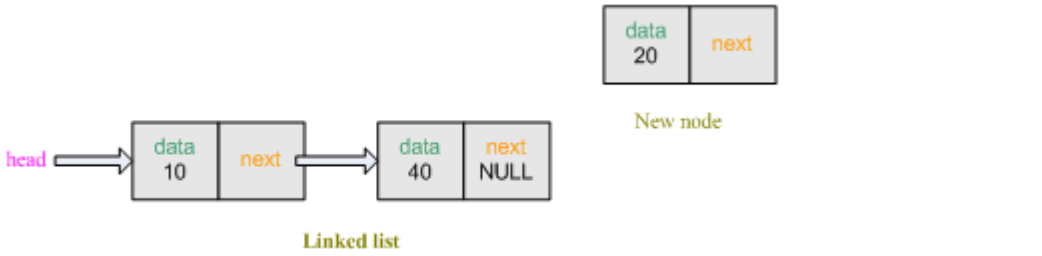
Linked list

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.

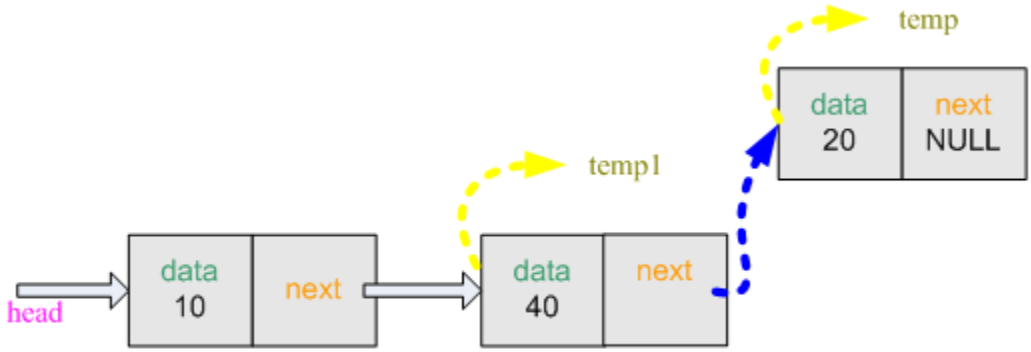


Linked list

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.

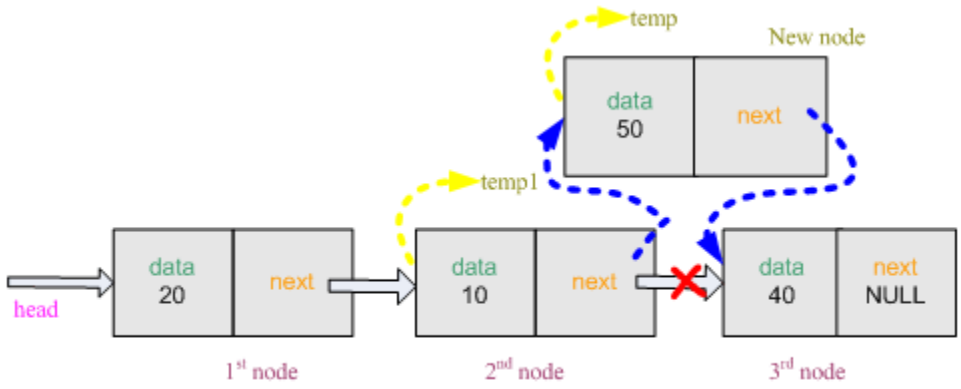


Linked list



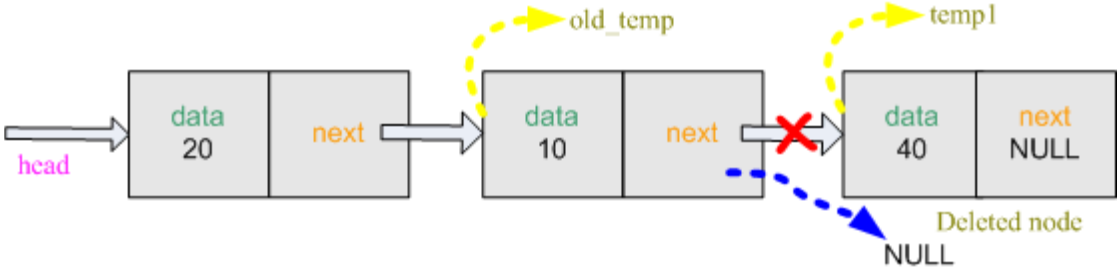
Linked list

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



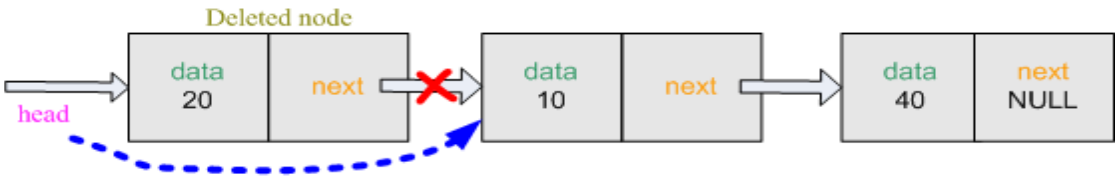
Linked list

5) 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 . Nownode *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 lode.



Linked list

6) 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.



Linked list

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.

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

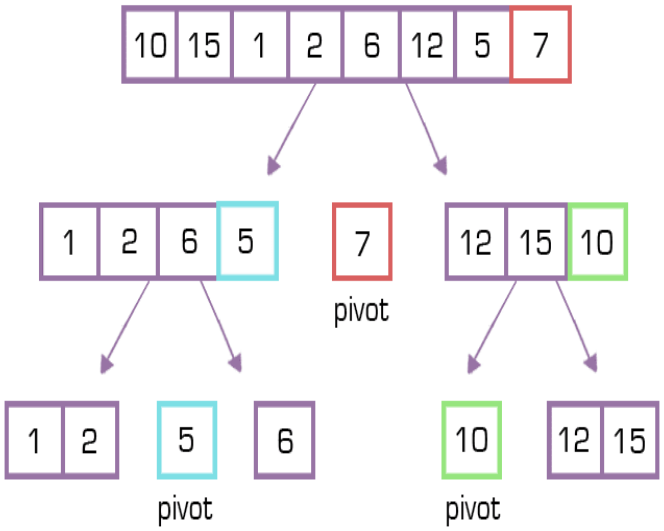
Assignment no: 08

Aim: To understand and implement 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.

Theory:

Sorting is a way of arranging items in a systematic manner. Quicksort is the widely used sorting algorithm that makes n log n comparisons in average case for sorting an array of n elements. It is a faster and highly efficient sorting algorithm. This algorithm follows the divide and conquer approach. Divide and conquer is a technique of breaking down the algorithms into subproblems, then solving the subproblems, and combining the results back together to solve the original problem.



Divide: In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element.

Conquer: Recursively, sort two subarrays with Quicksort.

Combine: Combine the already sorted array.

Quicksort picks an element as pivot, and then it partitions the given array around the picked pivot element. In quick sort, a large array is divided into two arrays in which one holds values that are smaller than the specified value (Pivot), and another array holds the values that are greater than the pivot.

After that, left and right sub-arrays are also partitioned using the same approach. It will continue until the single element remains in the sub-array.

Choosing the pivot

Picking a good pivot is necessary for the fast implementation of quicksort. However, it is typical to determine a good pivot. Some of the ways of choosing a pivot are as follows –

- Pivot can be random, i.e. select the random pivot from the given array.
- Pivot can either be the rightmost element of the leftmost element of the given array.
- Select median as the pivot element.

Input: Enter the first percentage of students

Output: sorting by Quick sort and display top 5 students marks

Algorithm:

```
QUICKSORT (array A, start, end)
{
1 if (start < end)
        {
         p = partition(A, start, end)
        QUICKSORT (A, start, p - 1)
        QUICKSORT (A, p + 1, end)
         }
}
```

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

Assignment no: 09

Aim: To understand and implement Circular linked list

Problem Statement: Write C++ program for storing appointment schedule for day. Appointments are booked randomly using linked list. Set start and end time and min and max duration for visit slot. Write functions for-

A) Display free slots

B) Book appointment

C) Sort list based on time

D) Cancel appointment (check validity, time bounds, availability)

E) Sort list based on time using pointer manipulation.

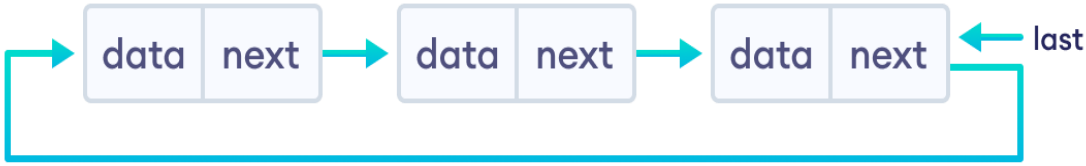
Theory:

A circular linked list is a type of linked list in which the first and the last nodes are also connected to each other to form a circle.

There are basically two types of circular linked list:

1. Circular Singly Linked List

Here, the address of the last node consists of the address of the first node.



2. Circular Doubly Linked List

Here, in addition to the last node storing the address of the first node, the first node will also store the address of the last node.

Input:

Enter the start time, end time and name of the person for which the appointment is booked and the maximum and minimum time of the appointment.

Output:

Sorted list and the book slot list , free slot displayed , appointment cancellation is done.

Conclusion: Thus we have studied the concept of circular link list.

Assignment 10

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.

Theory:

Stack

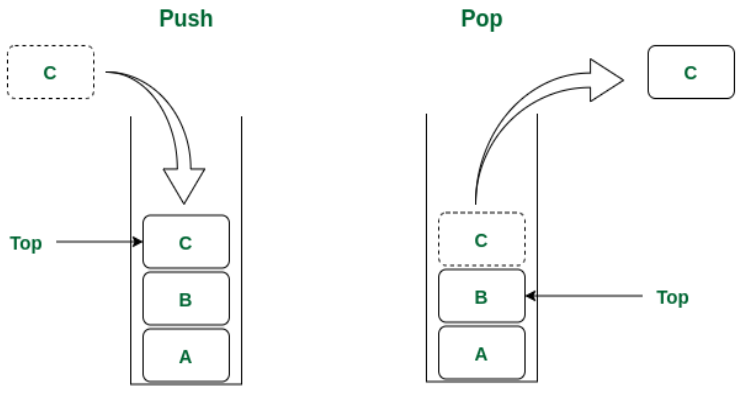
It is named stack as it behaves like a real-world stack, for example − deck of cards or pile of plates etc. A stack is an abstract data type that serves as a collection of elements, with two principal operations: push, which adds an element to the collection, and pop, which removes the most recently 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.

Stack Representation

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



Stack Data Structure

`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

else

Print Key Is Not a Palindrome

Step 7: End

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

Assignment 11

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.

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:

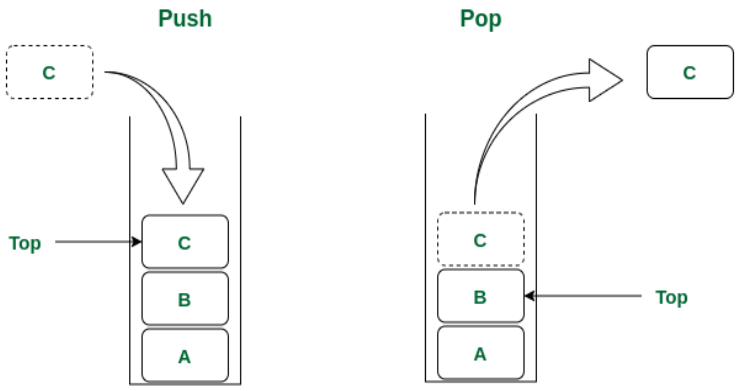
- push the item into the stack, and
- pop the item out of the stack.

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.



Stack Data Structure

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.

Balanced parentheses

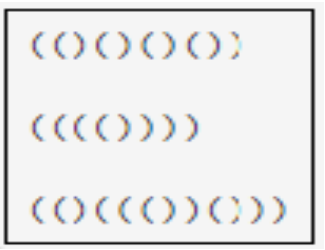
We now turn our attention to using stacks to solve real computer science problems.

You have no doubt written arithmetic expressions such as

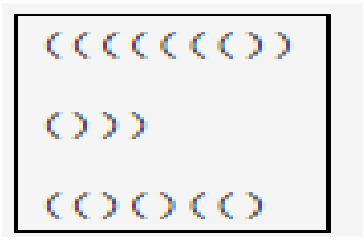
(5+6)∗(7+8)/(4+3)(5+6)∗(7+8)/(4+3)

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:



Compare those with the following, which are not balanced:



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.

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

Assignment No – 12

Aim: To illustrate the concept of queue.

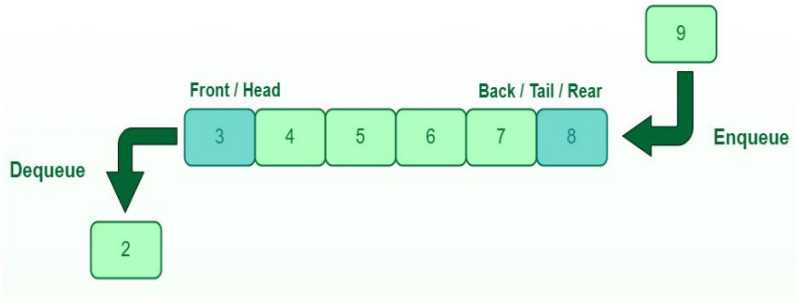
Problem Statement: Queues are frequently used in computer programming, and a typical example is the creation of a job queue by an operating system. If the operating system does not use priorities, then the jobs are processed in the order they enter the system. Write C++ program for simulating job queue.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

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 theresulting 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 front-rear+1, when implemented using arrays. Following figure will illustrate the basic operations on queue.



Queue Data Structure

Input: Enter the jobs to an operating system

Output: Add jobs and delete jobs from queue.

Conclusion : Thus we have studied the implementation of queue.

Assignment No-13

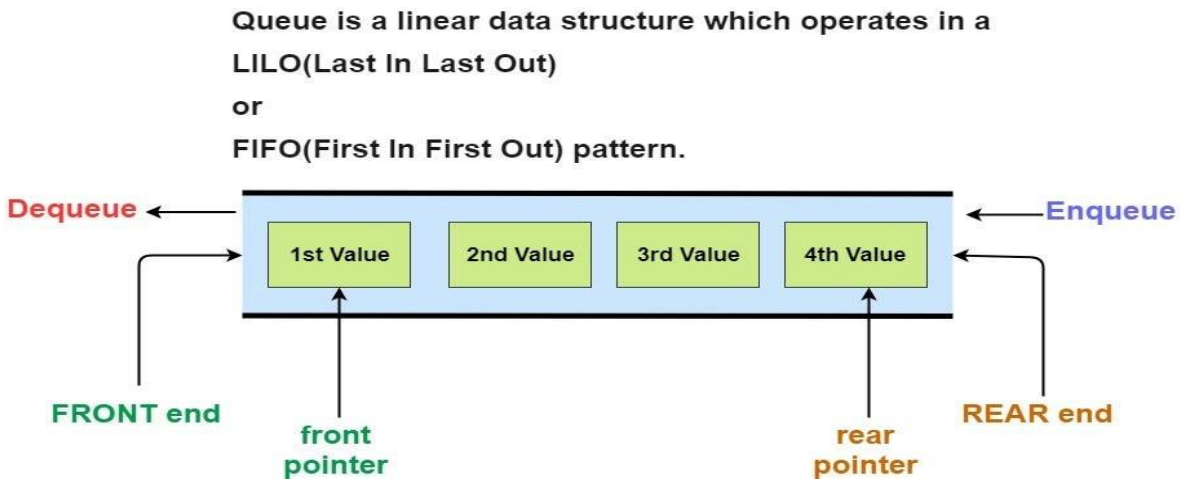
Aim: To illustrate the concept of double-ended queue (deque).

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.

Theory:

DeQueue:

Dequeue 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 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. .The DeQueue is represented as follows.



The output restricted Dequeue allows deletions from only one end and input restrictedDequeue 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,-1Q[] 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 (r == max-1) if yes
queue is fullstep

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

step4. Insert the element at pointer f as Q[f] = 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] = 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 element = Q[f]

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

step5. Stop

- **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

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