Experiment 1
Implementation Of Bubble Sort Algorithm

Date: 26-08-2020

Aim: To implement the bubble sort and it's algorithm

Data Structure Used: Arrays

Operation Used: Comparisons and Swapping

Algorithm:

Input: Unsorted array of length n
Output: Sorted array of length n

Step 1 : Start
Step 2 : Receive the size of the array in a variable n
Step 3 : i ← 0 //Receive the elements in the array
Step 4 : Repeat Step 5 to 6 until i=n
Step 5 : Receive an element and store it in a[i]
Step 6 : i ← i+1
Step 7 : i ← 0 //Beginning the bubble sort
Step 8 : Repeat steps 9 to Step 16 until I = n-1
Step 9 : j ← 0
Step 10 : Repeat steps 11 to step 15 until j=n-i-1
Step 11 : if arr[j]>arr[j+1] then do Step 12 to Step 14 else skip to Step 15
Step 12 : temp = arr[j]
Step 13 : arr[j] = arr[j+1]
Step 14 : arr[j+1] = temp
Step 15 : j=j+1
Step 16 : i+=i+1 //Bubble sort ends here
Step 17 : i ← 0 //Print the sorted array
Step 18 : Repeat Step 19 to 20 until i=n
Step 19 : Print the value of arr[i]
Step 20: i=i+1
Step 21 : Stop

Details of the Algorithm:
The Bubble sort algorithm takes a given array and keeps swapping the elements in such a way that the largest number (in case of ascending order) is guaranteed to come at the end of the loop on the first iteration of the i loop. This then divides the array into two parts, sorted and the unsorted, the sorted part is the one from n-i-1 to n-1 and the unsorted part is from 0 to (n-i-2)th element. In the second iteration of the I loop the largest element of the unsorted part gets moved to the n-i-1 th position (the previous (n-i-1)th position since the value of i is incremented by one). This continues until the value of i=n-1 in which case the value of the last element of the unsorted sub array becomes n-n+1-2 = -1 and the first position of the sorted sub-array becomes n-n+1-1=0 this proves that the whole array is sorted. The time complexity is $O(n^2)$, since the total number of comparison is n(n-1)/2

Result: the Program is successfully compiled and the desired output is obtained.

Program/ Source Code:

```
#include<stdio.h>

void main(){
    int comp=0,swaps=0,i,j;
    int arr[100];
    int n,temp;
    printf("Enter the number of elements in the arrat: ");
    scanf("%d",&n);
    printf("Enter the elements in the array : ");
    for(i=0;i<n;i++)
        scanf("%d%*c",arr+i);

    for(i=0;i<n-1;i++){
        for(j=0;j<n-i-1;j++){
            if(arr[j]>arr[j+1]){
                temp = arr[j];
                arr[j]=arr[j+1];
                arr[j+1]=temp;
                swaps++;
            }
            comp++;
        }
    }
    printf("Sorted Array: ");
    for(i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\nNumber of comparisons = %d\n",comp);
    printf("Number of swaps= %d\n",swaps);

}
```

Sample Input/Output

```
Sample Input 1:
5
23 43 56 78 91

Sample Output 1:
Sorted Array: 23 43 56 78 91
Number of comparisons = 10
Number of swaps= 0


Sample Input 2:
5
91 78 56 43 23

Sample Output 2:
Sorted Array: 23 43 56 78 91
Number of comparisons = 10
Number of swaps= 10
```

```
Sample Input 3:
5
78 43 56 91 23

Sample Output 3:
Sorted Array: 23 43 56 78 91
Number of comparisons = 10
Number of swaps= 6
```

Experiment 2
Implementation of Selection Sort Algorithm

Date: 26-08-2020

Aim: To implement the Selection sort and it's algorithm

Data Structure Used: Arrays

Operation Used: Comparisons and Swapping

Algorithm:

Input: An unsorted array of length n
Output: Sorted Array of length n

Step 1 : Start
Step 2 : Receive the size of the array in a variable n
Step 3 : i ← 0 //Receive the elements in the array
Step 4 : Repeat Step 5 and 6 until i=n
Step 5 : Receive an element and store it in a[i]
Step 6 : i ← i+1
Step 7 : i ← 0 // Beginning of Sorting process
Step 8 : Repeat steps 9 to 20 until i=n
Step 9 : pos ← i
Step 10: smallest ← arr[i]
Step 11: j ← i
Step 12: Repeat Steps 13 to 16 until j=n
Step 13: if arr[j] < smallest then do Steps 14 to 15
Step 14: smallest ← arr[j]
Step 15: pos ← j
Step 16: j ← j+1
Step 17: if pos != I then do steps 18 to 20
Step 18: temp ← arr[i]
Step 19: arr[i] ← arr[pos]
Step 20: arr[pos] ← temp
Step 21: i ← i++ //Selection Sort ends here
Step 22: i ← 0
Step 23: Repeat Step 22 to 23 until i=n //Print the sorted array
Step 24: Print the value of arr[i]
Step 25: i=i+1
Step 26: Stop

Description of the Algorithm:

The selection sort as the name implies selects the smallest element (in case of ascending order) from the unsorted sub array, initially the unsorted sub array is from 0 to n-1 where n is the number of element in the array, and swaps it with the first element in the said sub-array making the array from 0 to the i-1 sorted and the remaining () unsorted. This process goes on until the value of i becomes n at which the starting and ending indices of the sorted array becomes 0 and i-1 which is equal to n-1. Hence we can say that the array is sorted. The time complexity is $O(n^2)$, since the total number of comparison is n(n-1)/2

Result: the Program is successfully compiled and the desired output is obtained.

Program/ Source Code:

```
#include<stdio.h>

void main(){
    int comp=0,swaps=0,i,j;
    int arr[100];
    int n,temp,smallest,pos;
    printf("Enter the number of elements in the array: ");
    scanf("%d",&n);
    printf("Enter the elements in the array: ");
    for(i=0;i<n;i++)
        scanf("%d%*c",arr+i);

    for(i=0;i<n;i++){
        smallest =arr[i];
        pos =i;
        for(j=i;j<n;j++){
            if(arr[j]<smallest){
                smallest= arr[j];
                pos =j;
            }
            comp++;
        }
        if(pos!=i){
            temp = arr[pos];
            arr[pos] = arr[i];
            arr[i]=temp;
            swaps++;
        }

    }

    printf("Sorted Array: ");
    for(i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\nNumber of comparisons = %d\n",comp);
    printf("Number of swaps= %d\n",swaps);

}
```

Sample Input/Output

```
Sample Input1 :
5
23 43 56 78 91

Sample Output1 :
23 43 56 78 91
Number of comparisons : 15
Number of swaps : 0
```

```
Sample input 2:
5
91 78 56 43 23

Sample Output 2:
Sorted Array: 23 43 56 78 91
Number of comparisons = 15
Number of swaps= 2


Sample input 3:
5
43 78 56 91 23

Sample output 3:
Sorted Array: 23 43 56 78 91
Number of comparisons = 15
Number of swaps= 3
```

<u>Experiment 3</u>
<u>Implementation of Insertion Sort Algorithm</u>

Date: 06-08-2020

Aim: To sort a given set of elements using the insertion sort algorithm

Data Structures Used: Arrays
Operations Used: Comparison and Swapping

Algorithm:

Input : An Unsorted Array A[L...U] //L and U are the lower and upper bounds respectively

Output: Sorted Array

Steps:
Step 1 : Start
Step 2 : i ← L+1
Step 3 : while i<=U
    Step 1 : temp ← A[i]
    Step 2 : j ← i-1
    Step 3 : while j>=0
        Step 1 : if A[j]<temp
            Step 1 : End While
        Step 2 : End if
        Step 3 : A[j+1] ← A[j]
        Step 4 : j ← j-1
    Step 4 : End while
    Step 5 : A[j+1] ← temp
    Step 6 : I ← i+1
Step 4 : End while
Step 5 : Stop

Description of the Algorithm:
The insertion sort as the name suggests uses the insertion algorithm for an element in an array to sort the array. Initially the algorithm inserts the second element in the array in the proper position by moving elements before it to the right, just like the insertion algorithm for a sorted array, then we get the first two elements are sorted and the rest of the array is unsorted. Then the outer for loop moves on to the third element and the inner for loop inserts the element in the proper position of the sorted sub-array. This continuous till the last element resulting in the array being completely sorted.
The worst case complexity is $O(n^2)$ and the best case complexity is O(n).

Result : The Program was successfully complied and the required output was obtained.

Program:

```
#include<stdio.h>

void printarr(int *a, int n){
    for(int i = 0;i<n;i++){
        printf("%d ",*(a+i));
    }
}

void enterValues(int *a, int n){
    printf("Enter the elements of the array: ");
    for(int i =0; i<n;i++){
        scanf("%d%*c",a+i);
    }
}

void main(){
    int i,j;
    int n,comp=0,swaps=0;
    int arr[100];
    int temp;
    char c;

    printf("Enter the number of elements in the array: ");
    scanf("%d%*c",&n);
    enterValues(arr,n);
    printf("\n\n");

    for(i=1;i<n;i++){
        temp=arr[i];
        for(j=i-1;j>=0;j--){
            comp++;
            if(arr[j]<temp)break;
            arr[j+1]=arr[j];
            swaps++;
        }
        if(j!=i-1){
            swaps++;
            arr[j+1]=temp;
        }
        printf("The array after %d step is : ",i);
        printarr(arr,n);printf("\n");

    }

    printf("\nThe sorted array is -> ");
    printarr(arr,n);printf("\n\n");
    printf("The total number of comparisons = %d\nThe total number of swaps = %d\
n",comp,swaps);
}
```

Sample Input 1:
```
5
98 45 34 32 12
```

Sample Output 1:
```
Enter the number of elements in the array: 5
Enter the elements of the array: 98 45 34 32 12

The array after 1 step is : 45 98 34 32 12
The array after 2 step is : 34 45 98 32 12
The array after 3 step is : 32 34 45 98 12
The array after 4 step is : 12 32 34 45 98

The sorted array is -> 12 32 34 45 98

The total number of comparisons = 10
The total number of swaps = 14
```

Sample Input 2:
```
5
12 32 34 45 98
```

Sample Output 2:
```
Enter the number of elements in the array: 5
Enter the elements of the array: 12 32 34 45 98

The array after 1 step is : 12 32 34 45 98
The array after 2 step is : 12 32 34 45 98
The array after 3 step is : 12 32 34 45 98
The array after 4 step is : 12 32 34 45 98

The sorted array is -> 12 32 34 45 98

The total number of comparisons = 4
The total number of swaps = 0
```

Sample Input 3:
```
5
45 32 12 98 34
```

Sample Output 3:
```
Enter the number of elements in the array: 5
Enter the elements of the array: 45 32 12 98 34

The array after 1 step is : 32 45 12 98 34
The array after 2 step is : 12 32 45 98 34
The array after 3 step is : 12 32 45 98 34
The array after 4 step is : 12 32 34 45 98

The sorted array is -> 12 32 34 45 98

The total number of comparisons = 7
The total number of swaps = 8
```

Experiment 4
Implementation of Linear Search Algorithm

Date: 06-09-2020

Aim: To find an element in a give array using the linear search algorithm

Data Structures Used: Arrays
Operations Used : Comparison

Algorithm:

Input: An integer Array A[L...U] //L and U are the lower and upper bounds of the array. An integer 'q' which is to be searched

Output: An integer value KEY which is the index of the element 'q', -1 if the element doesn't exist in the array

Steps:
Step 1 : Start
Step 2 : i ← L
Step 3 : while i <= U
    Step 1: if(A[i] = q)
        Step 1: End while
    Step 2: End if
Step 4 : End while
Step 5: KEY ← i
Step 5 : if KEY=n
    Step 1: Print "The element q is not in the array"
Step 6 : else
    Step 1: Print "The element q is at position KEY+1 and at index KEY+1"
Step 7 : End if
Step 8 : Stop

Description of the Algorithm:
The Linear search will search the array for the element q in the array starting from the first element until the element to be searched is found or the last element is reached. If the control variable reaches the last element and it is not the required element then the program will print an error message. If the element is found then the program will return the posion and the index value of the element in the array
The Best case time complexity is O(1)
The Worst case time complexity is O(n)

Result: The Program was successfully compiled and the required output was obtained

Program:

```
/*Implementation of linear Search*/

#include<stdio.h>

/*Linear search funtion:
 * Takes in an array A and a value query to search for in the array
 * Returns the position of the element or prints an error message if the element
is not found
 */
int linearSearch(int* A,int n,int query){
    int KEY;
    int i;
    for(i=0;i<n;i++){
        if(A[i]==query){
            break;
        }
    }
    if(i==n){
        return -1;
    }
    else{
        return i;
    }
}

void getValues(int* A, int n){
    for(int i = 0; i<n; i++){
        scanf("%d%*c",A+i);
    }
}

void main(){
    int n;
    int arr[100];
    int elem;
    printf("Enter the size of the array: ");
    scanf("%d%*c",&n);
    printf("Enter the elements of the array: ");
    getValues(arr,n);
    printf("Enter the element to be searched: ");
    scanf("%d%*c",&elem);

    int KEY = linearSearch(arr,n,elem);
    if(KEY<0){
        printf("%d doesn't exist in the array\n",elem);
    }
    else{
        printf("First occurrence of %d is in position %d and index %d\
n",elem,KEY+1,KEY);
    }

}
```

Sample Input 1:
```
5
59 57 41 32 81
32
```

Sample Output 1:
```
Enter the size of the array: 5
Enter the elements of the array: 59 57 41 32 81
Enter the element to be searched: 32
First occurrence of 32 is in position 4 and index 3
```

Sample Input 2:
```
5
59 57 41 32 81
69
```

Sample Output 2:
```
Enter the size of the array: 5
Enter the elements of the array: 59 57 41 32 81
Enter the element to be searched: 69
69 doesn't exist in the array
```

Experiment 5
Implementation of Binary Search Algorithm

Date: 06-09-2020

Aim: To find an element in a give array using the Binary search algorithm

Data Structures Used: Arrays
Operations Used : Comparison

Algorithm:

Input: A sorted integer Array A[L...U] //L and U are the lower and upper bounds of the array. An integer ‘q’ which is to be searched

Output: An integer value KEY which is the index of the element ‘q’, -1 if the element doesn’t exist in the array

Steps:
Step 1 : Start
Step 2 : beg ← L // Variable initially pointing to the first index
Step 3 : last ← U // Variable initially pointing to the second index
Step 4 : KEY ← -1
Step 5 : while beg<=last
    Step 1 : mid ← (last+beg)/2
    Step 2 : if A[mid] > q
        Step 1: last ← mid-1
    Step 3 : else if A[mid] < q
        Step 1: beg ← mid+1
    Step 4 : else
        Step 1: KEY ← mid
    Step 5 :End If
Step 6 : if KEY = -1
    Step 1 : Print “The element is not in the array”
Step 7 : else
    Step 1 : Print “The element is at index KEY and position KEY+1”
Step 8 : End If
Step 9 : Stop

Description of the Algorithm:
The Binary search checks if the middle element is the element to be searched (q), if it is not then it checks if the middle element is greater than q if it is then it searches only the lower half of the array, hence it works only on sorted arrays. If the middle element is smaller than q it checks the upper half of the array.
The Best case time complexity is O(1). When the element to be searched is the middle element.
The Worst case time complexity is O(log(n)).

Result: The Program was successfully compiled and the required output was obtained.

Program:

```
/* Implementation of binary search
 */
#include<stdio.h>

int binarySearch(int *A,int n, int elem){
    int beg = 0;
    int last = n-1;
    int mid;
    while(beg<=last){
        mid = (last+beg)/2;
        if(A[mid]>elem){
            last=mid-1;
        }
        else if(A[mid]<elem){
            beg = mid+1;
        }
        else{
            return mid;
        }
    }
    return -1;
}

void inputArray(int *A, int n){
    for(int i = 0;i<n;i++){
        scanf("%d%*c",A+i);
    }
}

void main(){
    int n,elem;
    int arr[100];
    printf("Enter the number of elements in the array: ");
    scanf("%d%*c",&n);
    printf("Enter the elements of the array: ");
    inputArray(arr,n);

    printf("Enter the element to search for: ");
    scanf("%d%*c",&elem);

    int KEY = binarySearch(arr,n,elem);
    if(KEY<0){
        printf("%d is not in the array\n",elem);
    }
    else{
        printf("%d is found at position %d and at index %d in the array\
n",elem,KEY+1,KEY);
    }
}
```

Sample Input 1:

```
5
32 41 57 59 81
81
```

Sample Output 1:

```
Enter the number of elements in the array: 5
Enter the elements of the array: 32 41 57 59 81
Enter the element to search for: 81
81 is found at position 5 and at index 4 in the array
```

Sample Input 2:

```
5
32 41 57 59 81
23
```

Sample Output 2:

```
Enter the number of elements in the array: 5
Enter the elements of the array: 32 41 57 59 81
Enter the element to search for: 23
23 is not in the array
```

Experiment 6
Addition Of Two Polynomials

Date: 21-09-2020

Aim: To receive two polynomials and print their sum

Data Structure Used: Arrays

Operation Used: Comparisons

Algorithm:

Input: Two polynomial, A and B in tuple format and ‘a’ denoting the number of tems in polynomial A and ‘b’ denoting the number of terms in polynomial ‘B’
Output: Sum of the polynomial ‘C’

Step 1 : Start
Step 2 : Receive two polynomial in tuple format
Step 3 : i ← 0 //Pointer to the polynomial A
Step 4 : j ← 0 //Pointer to the polynomial B
Step 5 : while i<a and j<b //a and b are the number of terms in A and B respectively
    Step 1 : if A[i][0] = B[j][0]
        Step 1: C[k][0] ← A[i][0]
        Step 2: C[k][1] ← A[i][1] +B[j][1]
        Step 3: i++
        Step 4: j++
        Step 5: k++
    Step 2: else if A[i][0] <B[j][0]
        Step 1: C[k][0] ← B[j][0]
        Step 2: C[k][1] ← B[j][1]
        Step 3: j++
        Step 6: k++
    Step 3: else if A[i][0] >B[j][0]
        Step 1: C[k][0] ← A[i][0]
        Step 2: C[k][1] ← A[i][1]
        Step 3: i++
        Step 4: k++
    Step 4: Endif
Step 6 : EndWhile
Step 7 : while i<a
    Step 1: C[k][0] ← A[i][0]
    Step 2: C[k][1] ← A[i][1]
    Step 3: i++
    Step 4: k++
Step 8: EndWhile
Step 9: while j<b
    Step 1: C[k][0] ← B[j][0]
    Step 2: C[k][1] ← B[j][1]
    Step 3: j++
    Step 6: k++
Step 10 : EndWhile
Step 11 : Stop

Description of the Algorithm:

The two polynomials are stored as two different 2-D arrays with the first column containing the powers of the polynomial (in descending order) and the second row containing the corresponding coefficients of the polynomial. Two pointers pointing to the two polynomials are created, if the powers pointed by the two polynomials are same then the coefficients are added and the result is pushed in the sum array, else the coefficient of the greater power is pushed into the sum array.

Result: the Program is successfully compiled and the desired output is obtained.

Program/ Source Code:

```
#include<stdio.h>
#include<stdlib.h>

/*  Input : 2 polynomials of the form
 *          a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0
 *  Output: First polynomial the second polynomial and there sum
 */
/* Funtion to print the polynomials*/
void printPoly(int** a){
    int iterCount = a[0][0];
    int i;
    for(i = 1;i<iterCount;i++)
        printf("%d*X^%d + ",a[i][1],a[i][0]);
    printf("%d*X^%d\n",a[i][1],a[i][0]);
}

/* Funtion to convert the polynomial into tuple*/
int** createPolyFromString(char* s){
    int** a;
    int i,j;
    int maxPolySize = 10;

    int count = 0;
    int numberStack[10];
    int numberStackTop = -1;

    int number = 0;
    int negative = 0;

    //parsing the string

    a = (int**) malloc(maxPolySize*sizeof(int*));
    for(i = 0;i<maxPolySize;i++){
        a[i] = (int*)malloc(2*sizeof(int));
    }

    for(i = 0; s[i]!='\0'; i++){
        if(s[i] == '-'){
            negative = 1;
```

```
            i++;
        }

        if(s[i]>='0'&&s[i]<='9'){
            while((s[i]!= 'X'||s[i]!='x'||s[i]!=' '||s[i]!='^') &&
(s[i]>='0'&&s[i]<='9')){
                 // here s[i] will only be numbers
                number = number*10+(s[i]-'0');
                i++;
            }
            if(negative) numberStack[++numberStackTop] = -1*number;
            else numberStack[++numberStackTop] = number;


            negative = 0;
            number = 0;
        }
        if(s[i]=='+'||s[i]=='\0'){
                    count++;
                    a[count][0] = numberStack[numberStackTop--];
                    a[count][1] = numberStack[numberStackTop--];
        }

    }
    a[0][0] = count;

    return a;
}

/*Funtion to find the sum of the polynomials*/
int** sumOfPoly(int** a, int** b){
    int totalSize = a[0][0] + b[0][0]+2;
    int **c;
    int count = 0;
    c = (int**) malloc(totalSize*sizeof(int*));

    int i,j;
    for(i = 0;i<totalSize;i++){
        c[i] = (int*) malloc(2*sizeof(int));
    }

    i=1,j=1;
    while(i<=a[0][0]&&j<=b[0][0]){
        //If the powers are same then add the coefficients
        if(a[i][0]==b[j][0]){
            if(a[i][1]+b[j][1]==0){
                i++;j++;
                continue;
            }
            else{
                count++;
                c[count][0] = a[i][0];
                c[count][1] = a[i][1]+b[j][1];
                i++;j++;
            }
        }
```

```
        //If the powers arent same then push the one with the highest power into
polynomial c
        else if(a[i][0]<b[j][0]){
            count++;
            c[count][0] = b[j][0];
            c[count][1] = b[j][1];
            j++;
        }
        else if(b[j][0]<a[i][0]){
            count++;
            c[count][0] = a[i][0];
            c[count][1] = a[i][1];
            i++;
        }
    }

    /* If the while loop abve terminates prematurely i.e. after the elements of the
shorter of the two
       polynomial is added to the c polynomial*/

    while(i<=a[0][0]){
        count++;
        c[count][0] = a[i][0];
        c[count][1] = a[i][1];
        i++;
    }

    while(j<=b[0][0]){
        count++;
        c[count][0] = b[j][0];
        c[count][1] = b[j][1];
        j++;
    }

    c[0][0] = count;
    return c;
}
void main(){
    int** a;
    int** b;
    int** c;
    int strLength = 100;
    char* polyString = (char*) malloc(strLength*sizeof(char));

    /*Read the polynomials*/
        flush(stdin);
        printf("Enter polynomial 1 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 .....
an*X^0 ");
        scanf("%[^\n]",polyString);
        scanf("%*c"); //remove the \n charecter from the input stream
        a  = createPolyFromString(polyString);
        free(polyString);

        flush(stdin);
        flush(stdout);
```

```
        polyString = (char*) malloc(strLength*sizeof(char));

        printf("Enter polynomial 2 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 .....
an*X^0 ");
        scanf("%[^\n]",polyString);
        b  = createPolyFromString(polyString);
        free(polyString);
    /*Finish reading Polynomials*/

    printf("\nPolynomial 1 is: ");
    printPoly(a);
    printf("\nPolynomial 2 is: ");
    printPoly(b);

    c = sumOfPoly(a,b); //Find the sum of the polynomials
    printf("\nSum is ");
    printPoly(c);
    free(a);
    free(b);
    free(c);
}
```

Sample Input/Output

Sample input 1:

```
100*X^10 + 29*X^5 + 10*X^0
21*X^9 + 1*X^5 + 3*X^3 + 2X^1
```

Sample output 1:

```
Enter polynomial 1 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 -->
100*X^10 + 29*X^5 + 10*X^0
Enter polynomial 2 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 -->
21*X^9 + 1*X^5 + 3*X^3 + 2X^1

Polynomial 1 is: 100*X^10 + 29*X^5 + 10*X^0

Polynomial 2 is: 21*X^9 + 1*X^5 + 3*X^3 + 2*X^1

Sum is 100*X^10 + 21*X^9 + 30*X^5 + 3*X^3 + 2*X^1 + 10*X^0
```

Sample input 2:

```
12*X^100 +12*X^1
13*X^101 + -12*X^100 + 1*X^2
```

Sample output 2:

```
Enter polynomial 1 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 -->
12*X^100 +12*X^1
Enter polynomial 2 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 -->
13*X^101 + -12*X^100 + 1*X^2

Polynomial 1 is: 12*X^100 + 12*X^1

Polynomial 2 is: 13*X^101 + -12*X^100 + 1*X^2
```

```
Sum is 13*X^101 + 1*X^2 + 12*X^1
```

Sample input 3:

```
-11*X^12 + 1*X^0
11*X^12 + 13*X^10 + 14*X^0
```

Sample output 3:

```
Enter polynomial 1 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 --> -
11*X^12 + 1*X^0
Enter polynomial 2 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 -->
11*X^12 + 13*X^10 + 14*X^0

Polynomial 1 is: -11*X^12 + 1*X^0

Polynomial 2 is: 11*X^12 + 13*X^10 + 14*X^0

Sum is 13*X^10 + 15*X^0
```

Experiment 7
Sparse Matrix

Date: 21-09-2020

Aim: To receive two sparse matrices and print their transpose and their sum

Data Structure Used: Arrays

Operation Used: Comparisons, Addition

Algorithm for reading a Sparse matrix and obtaining the tuple representation:

Input: A Sparse matrix A, containing m rows and n columns
Output: Tuple Representation of the array

```
Step 1  : Start
Step 2  : Receive the sparse matrix A
Step 3  : Initialize the array sp which will contain the tuple representation of A
Step 4  : i ← 0
Step 5  : j ← 0
Step 6 : count ← 0
Step 7  : while i<m
        Step 1 : j ← 0
        Step 2 : while j<n
                Step 3 : if A[i][j]!=0
                        Step 1  : count ++
                        Step 2  : sp[count][0] ← i
                        Step 3  : sp[count][1] ← j
                        Step 4  : sp[count][2] ← A[i][j]
                Step 4 : Endif
                Step 5: j++
        Step 3 : EndWhile
        Step 4 : i++
Step 8  : EndWhile
Step 9  : sp[0][0] ← m
Step 10 : sp[0][1] ← n
Step 11 : sp[0][2] ← k
Step 12 : Stop
```

Description of the algorithm

The elements of the sparse Matrix A is iterated one by one and the ones which are non-zero is pushed into the tuple representation array, sp.

Algorithm For Transpose:

Input: Sparse matrix, A in sequential tuple representation
Output: Sparse matrix, A_T in tuple representation of the transpose of the input Sparse matrix

```
Step 1  : Start
Step 2  : Receive Sparse Matrix in tuple format
Step 3  : Initialize the array A_T
Step 4  : m ← A[0][0]           //The number of rows in A
Step 5  : n ← A[0][1]           //The number of columns in A
Step 6  : t ← A[0][2]           //The number of non-zero elements in A
Step 7  : i ← 0                 //Control Variable to iterate through the columns in A
Step 8  : while i<n
        Step 1  : j ← 1         //Control Variable to iterate through the nonzero elements in A
```

Step 2 : while j<=t
Step 1 : if A[j][1] = i:
Step 1 : k++
Step 2: A[j][0] ← A_T[k][1]
Step 3 : A[j][1] ← A_T[k][0]
Step 4 : A[j][2] ← A_T[k][2]
Step 2: End if
Step 9 : EndWhile
Step 10 : A_T [0][0] ← n
Step 11: A_T [0][1] ← m
Step 12: A_T [0][2] ← t
Step 13: Stop

Description of the algorithm:

For finding the transpose we require two loops one to keep track of the columns and another loop inside the first to iterate through the non-zero elements of the sparse array A. When the outer loop i is 0 then the inner loop checks for non-zero elements in the first column and puts them in the array A_T. Since the elements in the array A are stored row-wise the transpose matrix will have the elements in order.

Algorithm for Adding two Sparse Array:

Input: Two sparse array (A,B) in tuple representation
Output: Sum of the two arrays in tuple representation, C

Step 1 : Start
Step 2 : Receive the two sparse array in tuple representation
Step 4 : Initialize the array C to store the sum of A and B
Step 3 : i ← 1 //Pointer to the non-zero elements of array A
Step 4 : j ← 1 //Pointer to the non-zero elements of array B
Step 5 : if (A[0][0] = B[0][0] and A[0][1] = B[0][1]) //Arrays can only be added if the rows and columns are equal
Step 1 : k ← 0 //Variable to count the number of elements in C
Step 2 : while i<=A[0][2] or j<=B[0][2]
Step 1 : if A[i][0] < B[i][0]
Step 1 : k++
Step 2 : C[k][0] ← A[i][0]
Step 3 : C[k][1] ← A[i][1]
Step 4 : C[k][2] ← A[i][2]
Step 5 : i++
Step 2 : else if A[i][0] > B[j][0]
Step 1 : k++
Step 2 : C[k][0] ← B[j][0]
Step 3 : C[k][1] ← B[j][1]
Step 4 : C[k][2] ← B[j][2]
Step 5 : j++
Step 3 : else if A[i][1] < B[j][1]
Step 1 : k++
Step 2 : C[k][0] ← A[i][0]
Step 3 : C[k][1] ← A[i][1]
Step 4 : C[k][2] ← A[i][2]
Step 5 : i++
Step 4 : else if A[i][1] > B[j][1]
Step 1 : k++
Step 2 : C[k][0] ← B[j][0]
Step 3 : C[k][1] ← B[j][1]
Step 4 : C[k][2] ← B[j][2]
Step 5 : j++
Step 5: else
Step 1 : k++
Step 2 : C[k][0] ← A[i][0]

Step 3 : C[k][1] ← A[i][1]
Step 4 : C[k][2] ← A[i][2]+B[j][2]
Step 5 : i++
Step 6 : j++
Step 6 : Endif
Step 3: EndWhile
Step 4 : C[0][0] ← A[0][0]
Step 5 : C[0][1] ← A[0][1]
Step 6 : C[0][2] ← k
Step 6 : else
Step 1 : Print "The arrays cannot be added"
Step 7 : End if
Step 8 : Stop

Description of the algorithm:

Two pointers to the sparse arrays is taken and, the smallest row value pointed by the any of the two pointers is added to the resulting array C, If the row values are the same then the column values are compared and the one with the smallest column value is add to the array, if the column values are also same then the non-zero element is added and the resulting value is add to the resultant matrix C.

Result: the Program is successfully compiled and the desired output is obtained.

Program/ Source Code:

```
#include<stdio.h>
#include<stdlib.h>
#define MAX_SIZE 10

void printSparse(int **sp){
int i;
    printf("The sparse array is \n");
    for(i=0;i<=sp[0][2];i++){
        printf("[%d] %d  %d  %d \n",i,sp[i][0],sp[i][1],sp[i][2]);fflush(stdout);
    }
}

int** readSparse(){
    int** sp = (int**) malloc(MAX_SIZE*sizeof(int*));
    int m,n;
    int i,j,k = 0;
    int A[10][10];

    for (i = 0;i<MAX_SIZE;i++)
        sp[i] = (int*)malloc(3*sizeof(int));

    //get the values for rows and columns in A
    printf("Enter the number of rows and columns-> ");
    scanf("%d%*c%d%*c",&m,&n);

    //Receive values for the sparse matrix A
    printf("Enter the values of the sparse matrix A\n");
    for(i=0;i<m;i++){
        printf("Values for row %d -> ",i+1);
        for(j=0;j<n;j++){
            scanf("%d%*c",&A[i][j]);

            //find the triplet represent of the sparse matrix
            if(A[i][j]!=0){
                k++;
                sp[k][0] = i;
                sp[k][1] = j;
                sp[k][2] = A[i][j];
            }

        }
    }

    sp[0][0] = m;
    sp[0][1] = n;
    sp[0][2] = k;
    return sp;
}

void transpose(int** sp1){
    int m = sp1[0][0];
    int n = sp1[0][1];
    int count = sp1[0][2];
```

```
    int **sp_t = (int**)malloc(MAX_SIZE*sizeof(int*));

    for(int i = 0;i<MAX_SIZE;i++){
       sp_t[i] = (int*) malloc(MAX_SIZE*sizeof(int));
    }

    if(count == 0){
        printf("The array is empty ");
        free(sp_t);
        return;
    }
    else{
        sp_t[0][0]=sp1[0][1];
        sp_t[0][1]=sp1[0][0];
        sp_t[0][2]=sp1[0][2];

        int i,j,k=1;

        for(i = 0;i<n;i++){
            for(j = 1;j<=count;j++){
                if(sp1[j][1]==i){
                    sp_t[k][0] = sp1[j][1];
                    sp_t[k][1] = sp1[j][0];
                    sp_t[k][2] = sp1[j][2];
                    k++;
                }
            }
        }
        printSparse(sp_t);
    }
}

int** add(int** A, int** B){
    int **c;
    int size = A[0][2]+B[0][2]+2;
    int i,j;
    int count = 0;

    if(A[0][0]==B[0][0]&&A[0][1]==B[0][1]){
        printf("Arrays can be added\n");fflush(stdout);

        c  = (int**) malloc(size*sizeof(int*));
        for(i=0;i<size;i++){
            c[i] = (int*) malloc(3*sizeof(int));
        }

        c[0][0] = A[0][0];c[0][1] = A[0][1];

        i=1;j=1;
        while(i<=A[0][2]&&j<=B[0][2]){
            if(A[i][0]<B[j][0]){
                count++;
                c[count][0] = A[i][0];
                c[count][1] = A[i][1];
                c[count][2] = A[i][2];
                i++;
```

```
        }
        else if(A[i][0]>B[j][0]){
            count++;
            c[count][0] = B[j][0];
            c[count][1] = B[j][1];
            c[count][2] = B[j][2];
            j++;
        }
        else{
            if(A[i][1]<B[i][1]){
                count++;
                c[count][0] = A[i][0];
                c[count][1] = A[i][1];
                c[count][2] = A[i][2];
                i++;
            }
            else if(A[i][1]>B[j][1]){
                count++;
                c[count][0] = B[j][0];
                c[count][1] = B[j][1];
                c[count][2] = B[j][2];
                j++;
            }
            else{
                count++;
                c[count][0] = B[j][0];
                c[count][1] = B[j][1];
                c[count][2] = B[j][2]+A[i][2];
                i++;j++;
            }
        }
    }

while(i<=A[0][2]){
        count++;
        c[count][0] = A[i][0];
        c[count][1] = A[i][1];
        c[count][2] = A[i][2];
        i++;
    }

while(j<=B[0][2]){
        count++;
        c[count][0] = B[j][0];
        c[count][1] = B[j][1];
        c[count][2] = B[j][2];
        j++;
    }
    c[0][2] = count;
    return c;
}

else{
    printf("Matrices cant be added");
    free(A);
    free(B);
```

```
        exit(0);
        return 0;
    }
}
void main(){
    int i,j,k=0;
    int A[10][10];
    int **sp1;
    int **sp2;
    int m,n;    //m is the number of rows and n is the number of columns

    int** sum;
    char c;
        sp1 = readSparse();
        sp2 = readSparse();
        //terminate the loop if n is entered
        printSparse(sp1);
        printSparse(sp2);

        //Print the transposes
        printf("\nTranspose of the first matrix is \n");
        transpose(sp1);
        printf("\nTranspose of the second matrix is \n");
        transpose(sp2);

        //Find the sum of the matrices
        sum = add(sp1,sp2);

        printf("Sum of both the matrices is \n");
        printSparse(sum);
        free(sp1);
        free(sp2);
        free(sum);
}
```

Sample Input/Output

Sample input 1:

```
4 5
0 1 0 0 0
0 0 2 1 0
0 0 0 0 0
0 3 0 0 1
4 5
1 0 1 0 0
0 0 0 0 2
0 0 3 0 0
1 0 0 1 0
```

Sample output 1:

```
Enter the number of rows and columns-> 4 5
Enter the values of the sparse matrix A
Values for row 1 -> 0 1 0 0 0
```

```
Values for row 2 -> 0 0 2 1 0
Values for row 3 -> 0 0 0 0 0
Values for row 4 -> 0 3 0 0 1
Enter the number of rows and columns-> 4 5
Enter the values of the sparse matrix A
Values for row 1 -> 1 0 1 0 0
Values for row 2 -> 0 0 0 0 2
Values for row 3 -> 0 0 3 0 0
Values for row 4 -> 1 0 0 1 0
The sparse array is
[0] 4  5  5
[1] 0  1  1
[2] 1  2  2
[3] 1  3  1
[4] 3  1  3
[5] 3  4  1
The sparse array is
[0] 4  5  6
[1] 0  0  1
[2] 0  2  1
[3] 1  4  2
[4] 2  2  3
[5] 3  0  1
[6] 3  3  1

Transpose of the first matrix is
The sparse array is
[0] 5  4  5
[1] 1  0  1
[2] 1  3  3
[3] 2  1  2
[4] 3  1  1
[5] 4  3  1

Transpose of the second matrix is
The sparse array is
[0] 5  4  6
[1] 0  0  1
[2] 0  3  1
[3] 2  0  1
[4] 2  2  3
[5] 3  3  1
[6] 4  1  2
Arrays can be added
Sum of both the matrices is
The sparse array is
[0] 4  5  9
[1] 0  0  1
[2] 0  2  2
[3] 1  4  4
[4] 1  3  1
[5] 2  2  3
[6] 3  1  3
[7] 3  0  1
[8] 3  3  1
[9] 3  4  1
```

Sample input 2:

```
3 2
0 0
0 1
-2 0
2 3
1 0 0
0 0 3
```

Sample output 2:

```
Enter the number of rows and columns-> 3 2
Enter the values of the sparse matrix A
Values for row 1 -> 0 0
Values for row 2 -> 0 1
Values for row 3 -> -2 0
Enter the number of rows and columns-> 2 3
Enter the values of the sparse matrix A
Values for row 1 -> 1 0 0
Values for row 2 -> 0 0 3
The sparse array is
[0] 3  2  2
[1] 1  1  1
[2] 2  0  -2
The sparse array is
[0] 2  3  2
[1] 0  0  1
[2] 1  2  3

Transpose of the first matrix is
The sparse array is
[0] 2  3  2
[1] 0  2  -2
[2] 1  1  1

Transpose of the second matrix is
The sparse array is
[0] 3  2  2
[1] 0  0  1
[2] 2  1  3
Matrices cant be added
```

Experiment 8
Implementation Of Stack Using Array

Date: 21-09-2020

Aim: To implement Stack data structure using array

Data Structure Used: Arrays, Stack

Operation Used: Comparisons

Algorithm:

Algorithm for isEmpty()

Input: Stack A and pointer to the top most element, Top
Output: True if stack is empty, false if stack is not empty

Step 1 : Start
Step 2 : If top<0
    Step 1 : return true
Step 3: else
    Step 1: return false
Step 4 : Stop

Description of the Algorithm:

Returns a true value it the stack is empty false if otherwise

Algorithm for isFull()

Input: Stack A and pointer to the top most element, Top
Output: True if stack is full, false if stack is not full

Step 1 : Start
Step 2 : If top>=size //size is the predefined size of the array A
    Step 1 : return false
Step 3: else
    Step 1: return true
Step 4 : Stop

Description of the Algorithm:

Returns a true value it the stack is full false if otherwise

Algorithm for push function:

Input: Stack A and pointer to the top most element, Top and element to be inserted e
Output: Stack with the element e added on the top

Step 1 : Start
Step 2 : If isFull()
    Step 1 : Print "Overflow, The stack is Full"
Step 3: else
    Step 1: A[++top] = e
Step 4 : Stop

Description of the Algorithm:

Takes an input e and adds it to the top of the stack if it is not full

Algorithm for pop function:

Input: Stack A and pointer to the top most element, Top

Output: Stack with the top element removed

Step 1 : Start
Step 2 : If isEmpty()
    Step 1 : Print “Underflow, There is no element in the stack”
Step 3: else
    Step 1: top--
Step 4 : Stop

Description of the Algorithm:

Removes the top element of the array by decrementing it

Algorithm for seek function:

Input: Stack A and pointer to the top most element, Top and the index of the element from the top

Output: Element e at position index from the top

Step 1 : Start
Step 2 : i = top-index+1
Step 3 : If i<0
    Step 1 : Print “There is no element at position index from top”
    Step 2: return 0
Step 4: else
    Step 1: return A[i]
Step 5 : Stop

Description of the Algorithm:

Returns the element a position index from the top. That is if index is 1 then it will return top. If the value of top is less than index-1 then error is shown.

Algorithm for seekTop function:

Input: Stack A and pointer to the top most element, Top.

Output: Element at the top of the stack

Step 1 : Start
Step 2 : If isEmpty()
    Step 1 : Print “Underflow There is no element in the array”
    Step 2: return 0
Step 3: else
    Step 1: return A[top]
Step 4 : Stop

Description of the Algorithm:

Element at the top of the array is returned

Result: The program is successfully compiled and the desired output is obtained.

Program/ Source Code:

```
#include<stdio.h>
#include<stdlib.h>

#define SIZE 50


int A[SIZE];
int top = -1;

int  isEmpty(){
    if(top<0){
        return 1;
    }
    else{
        return 0;
    }
}

int isFull(){
    if(top<SIZE)
        return 0;
    else
        return 1;
}

int peek(int index){
    int i = top-index +1;
    if(i<0){
        printf("Underflow there is no element in the array \n");
        return 0;
    }
    else{
        return A[i];
    }
}

int stackTop(){
    if(isEmpty()){
        printf("The Stack is empty no element in stack\n");
        return 0;
    }
    else{
        return A[top];
    }
}

void push(int a){
    if(isFull()){
        printf("Stack is Full: Overflow");
    }
    else{
        top = top+1;
        A[top] = a;
```

```
    }
}

int pop(){
    int a;
    if(isEmpty()){
        printf("Underflow Stack is empty no element to pop");
    }
    else{
        a =  A[top];
        top--;
    }
    return a; //garbage or error is returned if underflow occurs
}

void main(){
    int c;
    int i;
    int e;
    int RUN = 1;
    while(RUN){
        printf("\n");
        printf("========================\n");
        printf("Menu\n");
        printf("1.push\n2.pop\n3.Check if empty\n4.Check if full\n5.Element at top\
n6.peek\n7.Exit\n");
        printf("========================\n");
        printf("\nEnter Choice ---> ");
        scanf("%d%*c",&c);
        switch(c){
            case 1: printf("\nEnter an element to push into the array --> ");
                    scanf("%d%*c",&e);
                    push(e);
                    break;
            case 2: e =pop() ;
                    printf("\nElement poped is %d\n",e);
                    break;
            case 3: if(isEmpty()){
                        printf("Stack is empty\n");
                    }
                    else{
                        printf("Stack is not empty\n");
                    }
                    break;
            case 4: if(isFull()){
                        printf("Stack is full\n");
                    }
                    else{
                        printf("Stack is not full\n");
                    }
                    break;
            case 5: e = stackTop();
                    printf("The Element at top is %d\n", e);
                    break;
            case 6: printf("Enter the value of the index--> ");
                    scanf("%d%*c",&i);
```

```
                    e = peek(i);
                    printf("\nThe %dth element in the stack is %d\n",i,e);
                    break;
            case 7: RUN = 0;
                    printf("\nExiting!!!!!!!!!!\n");
                    break;
            default: printf("Enter a proper value!!!!!!!!!!!!!!! \n");
        }
    }
}
```

Sample Input/Output

Sample input:

```
1
32
1
-41
1
12
2
5
6
2
2
2
7
```

Sample Output:

```
=========================
Menu
1.push
2.pop
3.Check if empty
4.Check if full
5.Element at top
6.peek
7.Exit
=========================

Enter Choice ---> 1

Enter an element to push into the array --> 32

=========================
Menu
1.push
2.pop
3.Check if empty
4.Check if full
5.Element at top
6.peek
7.Exit
=========================
```

```
Enter Choice ---> 1

Enter an element to push into the array --> -41

=========================
Menu
1.push
2.pop
3.Check if empty
4.Check if full
5.Element at top
6.peek
7.Exit
=========================

Enter Choice ---> 1

Enter an element to push into the array --> 12

=========================
Menu
1.push
2.pop
3.Check if empty
4.Check if full
5.Element at top
6.peek
7.Exit
=========================

Enter Choice ---> 2

Element poped is 12

=========================
Menu
1.push
2.pop
3.Check if empty
4.Check if full
5.Element at top
6.peek
7.Exit
=========================

Enter Choice ---> 5
The Element at top is -41

=========================
Menu
1.push
2.pop
3.Check if empty
4.Check if full
5.Element at top
6.peek
```

```
7.Exit
=========================

Enter Choice ---> 6
Enter the value of the index--> 2

The 2th element in the stack is 32

=========================
Menu
1.push
2.pop
3.Check if empty
4.Check if full
5.Element at top
6.peek
7.Exit
=========================

Enter Choice ---> 2

Element poped is -41

=========================
Menu
1.push
2.pop
3.Check if empty
4.Check if full
5.Element at top
6.peek
7.Exit
=========================

Enter Choice ---> 2

Element poped is 32

=========================
Menu
1.push
2.pop
3.Check if empty
4.Check if full
5.Element at top
6.peek
7.Exit
=========================

Enter Choice ---> 7

Exiting!!!!!!!!!
```

Experiment 9
Infix To Postfix Conversion and Evaluation

Date: 18-10-2020

Aim: To receive and infix expression and find the corresponding postfix expression and evaluate it.

Data Structure Used : Stack, Arrays

Algorithm for Conversion from infix to postfix

Input: Arithmetic expression E in infix notation with a right parenthesis at the end of the expression. A Stack with an opening parenthesis at the top, In-Stack precedence and incoming precedence of operators used.

Output: Corresponding postfix expression

Data Structure: Stacks

Operations Used : symbol() to a read the symbol from the expression

Steps:

```
Step 1: Top = -1, Push('(')
Step 2:while(Top>-1) do
        Step 1: item = E.symbol()
        Step 2: x = pop()
        Step 3:   case item=operand:
                          Step 1: Push(x)
                          Step2 : output(item)
                  case item=')':
                          Step 1:while x !=')' do
                                  Step 1: output(x)
                                  Step 2: x = Pop()
                          Step 2 :end while
                  case isp(x)>=icp(item):       //If the operator in the stack has a higher precedenc
                          Step 1: while isp(x)>=icp(item) do  //pop items from the stack until
                                  Step 1: output(x)            //an item with lower percedence
                                  Step 2: x = Pop()            // occurs
                          Step 2: end while
                          Step 3: Push(x)
                          Step 4: Push(item)          //Push the item into the stack
                  case isp(x) < icp(item):            //If the operator in the stack has a lower
                          Step 1 : Push(x)            //precedence, push the item into the
                          Step 2 : Push(item)         //stack
                  default :
                          Step 1 : Print("invalid expression")
Step 3 : EndWhile
Step 4 :Stop
```

Description of the Algorithm:

The algorithm converts infix expression to the corresponding postfix expression, using the stack data structure. When a operand is encountered then it is outputted but on encountering an operator all the operators with precedence higher than it is popped out of the stack and outputted to the postfix expression and on encountering an operator with a lower precedence then the scanned operator is pushed into the array. On encountering an open parenthesis it is push it into the stack and on encountering a closing parenthesis, all elements are popped out of the stack until an opening parenthesis is found.

Algorithm for Evaluation of the postfix Expression

Input: Postfix expression E
Output : Result after the evaluation of the postfix expression
Data Structure: Stacks
Operations used : *symbol()* to read a symbol from the expression, *performOperation(operand1,operand2, operator)*: performs the required mathematical operation denoted by operator

Steps:

```
Step 1 : Start
Step 2 : Top = -1
Step 3 : item = E,symbol()
Step 4 : Push(item)
Step 5 : item = E.symbol()
Step 6 : Push(item)
Step 7 : while(Top > 0)
        Step 1: item = E.symbol()
        Step 2: if(item.isOperator() ) then
                Step 1 : operand2 = Pop()
                Step 2 : operand1 = Pop()
                Step 3 : result = performOperation(operand1,operand2,item)
                Step 4: Push(result)
        Step 3: else:
                Step 1: Push(item)
        Step 4: End While
Step 8 : result = Pop()
Step 9 : print result
```

Description of the algorithm

Elements are read form the postfix expression, if an operand is encountered it is pushed into the stack and if an operator is encountered, two elements are popped form the stack and the corresponding operation is performed. Then the result is pushed into the stack. If there is only one element in the stack then that is the result of the expression

Program Code

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

#define SIZE 50

//START: Defnition of structures

//Character Stack
typedef struct stacks
{
    int top;
    char stk[SIZE];
}stack;

void push(stack *s,char ch)
{
    if(s->top>=SIZE)
    {
        printf("Stack Overflow Error.EXITING\n");
        exit(0);
    }
    else{
        s->stk[++(s->top)] = ch;
    }
    return;
}

char pop(stack *s)
{
    if(s->top<0)
    {
        printf("Error has occurred. Stack is empty\n");
        exit(0);
        return 0;
    }
    else
    {
        char a = s->stk[s->top];
        (s->top)--;
        return a;
    }
}

//Integer Stack
typedef struct stacksInt
{
    int top;
    int stk[SIZE];
}intStack;

void intPush(intStack *s,int a)
{
    if(s->top>=SIZE)
    {
        printf("Stack Overflow Error.EXITING\n");
        exit(1);
    }
    else{
        s->stk[++(s->top)] = a;
    }
    return;
}

int intPop(intStack *s)
{
    if(s->top<0)
```

```
    {
        printf("Error has occurred. Stack is empty\n");
        exit(1);
        return 0;
    }
    else
    {
        int a = s->stk[s->top];
        (s->top)--;
        return a;
    }
}


//START: Utility funtions definition

/* the utility function used are:
 * 1. int verifyExpression(char* exp) --> Checks if the expression is valid, i.e there are correct
number of operators and operands
 *
 * 2. void printExpression(char* exp) --> Prints the expression
 *
 * 3. int findPrecedence(char a) -->returns the instack precedence of the operator passed
 *
 * 4. void getValues(char *exp, int **values) -->since the expression contains only chrecter variables
this funciton
 *                                                asks the user for values for each of the charecter
 *
 * 5. int getValue(char c, int** values) -->this funtion is used to find the value of the charecter c
from the
 *                                           array of values user has entered
 *
 * 6. int evaluate(int a, int b, char c) -->performs the operation c with a as the first operand and b
as the second operand
 *
 * */


int verifyExpression(char* exp) //Verifies wheather the given expression contains only mathematical
operations and letters
{
    int i = 0;
    //char *cleared = (char*) malloc(50*(sizeof(char)));
    int operands = 0;
    int operators =0;
    int paranthsis = 0;
    for(i = 0;exp[i]!='\0';i++){
        if(!isalpha(exp[i])){
            if(!(exp[i]=='+'||exp[i]=='-'||exp[i]=='*'||exp[i]=='/'||exp[i]=='^'||exp[i]==')'||
exp[i]=='('))
            {
                printf("The Expression must contain only alphbets and mathematical operators \n\n");
                return 0;
            }
            else{
                if(exp[i]=='*'|| exp[i]=='/'|| exp[i]=='-'|| exp[i]=='+'||exp[i]=='^' )
                {
                    operators ++;
                }else if(exp[i]=='('||exp[i]==')'){
                    paranthsis++;
                }
            }
        }
        else{
            operands++;
        }
    }
    if(operators+1 > operands)
    {
        printf("The number of operators for %d operand(s) shoud be: %d but %d found\
```

```
n",operands,operands-1,operators);
        exit(1);
    }
    else if(operators+1< operands){
        printf("The number of operands for %d operation(s) must be %d, but %d found\
n",operators,operators+1,operands);
        exit(1);
    }

    if (paranthsis%2!=0)
    {
        printf("ERROR!!!!! The Expression contains an incomplete paranthesis\n");
        exit(1);
    }
    return 1;
}

void printExpression(char* exp) //Prints the given experssion
{
    for(int i = 0;exp[i]!='\0';i++)
        printf("%c ",exp[i]);
    printf("\n");
}

int findPrecedence(char a)
{
    switch(a)
    {
        case '+':
        case '-':return 2;
                 break;
        case '*':
        case '/':return 4;
                 break;
        case '^':return 5;
                 break;
        case '(':return 0;
                 break;
        default: printf("Invalid Expression \n");
                 exit(0);
                 return -1;
    }

}

void getValues(char *exp, int **values)
{
    int i = 0;int j = 0;
    int alreadyScanned = 0;


    for(i = 0; exp[i]!='\0';i++)
    {
        alreadyScanned = 0;
        if(isalpha(exp[i])){
            for(j =0;values[j][0]!=0;j++){
                if(exp[i]== values[j][0]){
                    alreadyScanned = 1;
                }
            }
            if(!alreadyScanned){
                values[j][0] = exp[i];
            }
        }
    }

    printf("Enter the values of ");
    for(j = 0; values[j][0]!=0;j++){
        printf("%c, ",(char)values[j][0]);
    }
```

```
        printf("\b\b: ");

    fflush(stdout);
    fflush(stdin);

    for(j=0;values[j][0]!=0;j++)
    {
        scanf("%d%*c",&values[j][1]);
    }

    /*
    printf("\nThe values entered are :\n");
    for(j = 0; values[j][0]!=0;j++){
        printf("%c - %d\n",(char)values[j][0],values[j][1]);
    }*/
}

int getValue(char c, int** values)
{
    int i=0;
    while(values[i][0]!=c)
    {
        i++;
    }
    return values[i][1];
}

int evaluate(int a, int b, char c)
{
    int i =0;
    int res=1;
    switch(c)
    {
        case '+':return a+b;
        case '-':return a-b;
        case '*':return a*b;
        case '/':return a/b;
        case '^':while(i++<b) res *=a;
                 return res;
        default : printf("Such an charecter is not found Exiting\n");
                  exit(1);

    }
}
//END: Defenition of utility funtions


//START: Convertion to postfix Algorithm
char* convertToPostFix(char* str)
{
    int i=0,j=0;
    int isp,icp;
    int operand;

    char stkItem;

    stack *s = (stack*) malloc(sizeof(stack));
    s->top = -1;
    push(s,'(');

    for(;str[i]!='\0';i++);
    str[i] = ')';
    str[i+1] = '\0';

    // printExpression(str);   //for debugging purposes

    char *postfixExp = (char*) malloc(i*sizeof(char));
    for(i=0;str[i]!='\0';i++)
    {
```

```
        /*printf("Iterarion %d\nItem Read = %c ",i+1,str[i]);*/ //for debugging purposes
        operand = 0;
        switch(str[i])
        {
            case '+':;
            case '-':icp = 1;
                    break;

            case '*':
            case '/':icp = 3;
                    break;

            case '^':icp = 6;
                    break;

            case '(':icp = 9;
                    break;

            case ')':icp = 0;
                    break;

            default :postfixExp[j] = str[i];
                     j++;
                     operand = 1;
                     break;
        }
        if(!operand)
        {
            stkItem = pop(s);
            //printf("stkItem = %c",stkItem);  //for debugging purposes

            if(str[i]!=')')
            {
                isp = findPrecedence(stkItem);
                while(isp>=icp)
                {
                    if(s->top == -1)
                    {
                        printf("Invalid Expression\n");
                        exit(1);
                    }
                    postfixExp[j]=stkItem;
                    j++;
                    stkItem = pop(s);
                    isp = findPrecedence(stkItem);
                }
                push(s,stkItem);
                push(s,str[i]);
            }
            else
            {
                while(stkItem!='(')
                {
                    postfixExp[j] = stkItem;
                    j++;
                    stkItem = pop(s);
                }
            }
        }
        /*printf("\n-----------------------------------"); //for debugging purposes
        printf("\n");*/
    }
    postfixExp[j] = '\0';
    return postfixExp;

}
//END: Convertion to postfix Algorithm

//STRAT: Evatuation algorithm
```

```
int evaluatePostfix(char *exp)
{
    int **values = (int**) malloc(SIZE*sizeof(int*));
    intStack *s = (intStack*)malloc(sizeof(intStack));
    s->top = -1;
    int operand1,operand2;

    int i;
    for(i = 0; i<SIZE;i++)
    {
        values[i] = (int*) calloc(2,sizeof(int));
    }

    getValues(exp,values);

    for(i=0;exp[i]!='\0';i++)
    {
        switch(exp[i])
        {
            case '+':
            case '-':
            case '*':
            case '/':
            case '^':
                     operand2 = intPop(s);
                     operand1 = intPop(s);
                     intPush(s,evaluate(operand1,operand2,exp[i]));
                     break;

            default: intPush(s,getValue(exp[i],values));
                     break;
        }
    }
    free(values);
    return intPop(s);
}

//END : Evaluation algorithm


int main()
{
    char *exp = (char*) malloc(50*(sizeof(char)));
    char *pexp;
    char c;

       printf("Infix to postfix conversion and evaluation\n");
       printf("==========================================\n");
       printf("Enter the infix expression for \"20+30\" as \"a+b\" without any spaces\n");
       printf("between the characters and then later enter the values of a and b when asked\n\n");

    printf("Enter the Infix Expression : ");
    scanf("%[^\n]",exp);
    while((c = getchar()) != '\n' && c != EOF);
    if(verifyExpression(exp))
    {
        //printExpression(exp);
        pexp = convertToPostFix(exp);
        printf("\nPostfix Expression is -->  ");
        printExpression(pexp);
        int result = evaluatePostfix(pexp);
        printf("The result is = %d\n",result);
        free(pexp);

    }
    free(exp);
       return 0;
}
```

Sample input 1

```
((A+((B^C)-D))*(E-(A/C)))
3 5 -1 9 20
```

Sample output 1

```
Infix to postfix conversion and evaluation
==========================================
Enter the infix expression for "20+30" as "a+b" without any spaces
between the characters and then later enter the values of a and b when asked

Enter the Infix Expression : ((A+((B^C)-D))*(E-(A/C)))

Postfix Expression is -->  A B C ^ D - + E A C / - *
Enter the values of A, B, C, D, E: 3 5 -1 9 20
The result is = -115
```

Sample input 2

```
A^b^D
-2 2 3
```

Sample output 2

```
Infix to postfix conversion and evaluation
==========================================
Enter the infix expression for "20+30" as "a+b" without any spaces
between the characters and then later enter the values of a and b when asked

Enter the Infix Expression : A^b^D

Postfix Expression is -->  A b D ^ ^
Enter the values of A, b, D: -2 2 3
The result is = 256
```

Sample input 3

```
a-(b+c*c)+e*f
90 -20 4 1 6
```

Sample output 3

```
Infix to postfix conversion and evaluation
==========================================
Enter the infix expression for "20+30" as "a+b" without any spaces
between the characters and then later enter the values of a and b when asked

Enter the Infix Expression : a-(b+c*c)+e*f

Postfix Expression is -->  a b c c * + - e f * +
Enter the values of a, b, c, e, f: 90 -20 4 1 6
The result is = 100
```

Experiment 10
Queue Implementation Using Array

Date : 02-10-2020

Aim: To implement a Queue using array

Data Structure used : Queue, Array

Algorithms

1. Algorithm for enqueue

Input: An Array implementation of Queue (Q[SIZE]), with front pointing to the first element and rear pointing to the last element in and an element E to be inserted into the queue.

Output: The Queue with the element E inserted at the rear

Data Structure: Queue

Steps:

Step 1: if(rear == SIZE) then
    Step 1: print("The queue is full insertion not possible")
    Step 2: exit(1)
Step 2: else
    Step 1: if(rear == -1) then
        Step 1: front ++
    Step 2: EndIf
    Step 3: Q[++rear] = E
Step 3: EndIf

2. Algorithm for dequeue

Input: An Array implementation of Queue (Q[SIZE]), with front pointing to the first element and rear pointing to the last element in the queue.

Output: The element E which is removed form the front of the queue

Steps

Step 1: if(front == -1) then
    Step 1: print("The Queue is empty")
    Step 2: exit(1)
Step 2: else
    Step 1: E = Q[front]
    Step 2: if(front == rear) then
        Step 1: front =-1
        Step 2: rear =-1
    Step 3: else
        Step 1: front--
    Step 4: endif
Step 3: endif

Program code:

```
/* Queue implemetation using dynamic array
 *  Done By : Rohit Karuankaran
 * */


#include <stdlib.h>
#include <stdio.h>
//#define SIZE 50


typedef struct queue_structure_datatype
{
    int *Q;
    int size;
    int front;
    int rear;
}queue;

void initQueue(queue *q)
{
    q->size = 16;
    q->Q = (int*) malloc(q->size*sizeof(int));
    q->front = -1;
    q->rear = -1;
}

void delQueue(queue *q)
{
    free(q->Q);
}

void incrSize(queue *q)
{
    q->size = 2*(q->size);
    int *tmp = (int*) realloc (q->Q,q->size*sizeof(int));
    if(tmp==NULL)
    {
        printf("Heap is full memory not available");
    }
    else
    {
        q->Q = tmp;
    }
}


void enQueue(queue *q,int elem)
{
    if(q->rear>=q->size)
    {
       // printf("The Queue is full Inseriton not possible\n");
        incrSize(q);
```

```c
    }
    else
    {
        if(q->front==-1)
        {
            q->front=q->front+1;
        }
        q->rear = q->rear+1;
        q->Q[q->rear] = elem;
        return;
    }
}

int deQueue(queue *q)
{
    if(q->front == -1)
    {
        printf("QUEUE IS EMPTY THERE IS NO ELEMENT TO DELETE\n");
        return -1;
    }

    else
    {
        int elem = q->Q[q->front];

        if(q->front==q->rear)
        {
            q->front = -1;
            q->rear = -1;
        }

        else
            q->front=q->front+1;
        return elem;
    }
}

void displayQueue(queue *q)
{
    int i = q->front;
    if(q->front)
    {
        printf("EMPTY");
        return;
    }
    while(i>=0&&i<=q->rear)
    {
        printf("%d ",q->Q[i]);
        i++;
    }

}

int main()
{
    queue *myQueue = (queue*) malloc(sizeof(queue));
```

```
    int RUN = 1;
    int elem;
    int choice;
    initQueue(myQueue);
    while(RUN)
    {
        printf("=======================\n");
        printf("        Menu\n");
        printf("=======================\n\n");
        printf("1.Enter into the queue\n");
        printf("2.Remove from the queue\n");
        printf("3.Display the queue\n");
        printf("4.Exit\n");
        printf("Enter your choice : ");

        scanf("%d%*c",&choice);
        switch(choice)
        {
            case 1: printf("Enter the element you want to enter into the Queue :
");
                    scanf("%d%*c",&elem);
                    enQueue(myQueue,elem);
                    break;

            case 2: elem = deQueue(myQueue);
                    printf("The element remove is :%d\n",elem);
                    break;

            case 3: printf("The Queue is: ");
                    displayQueue(myQueue);
                    printf("\n");
                    break;
            case 4: RUN = 0;
                    break;
            default: printf("Enter a valid input\n\n");
        }
    }

    /*
    insert(myQueue,32);
    insert(myQueue,21);
    displayQueue(myQueue);
    */
    delQueue(myQueue);
    printf("\nExiting.....\n");

}
```

Sample input/Output:

```
┌─rohit@iris ~/Programing/C/CSL201/2020-10-26
└─➤ gcc -Wall queue.c -o queue.o
┌─rohit@iris ~/Programing/C/CSL201/2020-10-26
└─➤ ./queue.o
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 1
Enter the element you want to enter into the Queue : 23
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 1
Enter the element you want to enter into the Queue : 65
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 1
Enter the element you want to enter into the Queue : 93
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 3
The Queue is: 23 65 93
```

```
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 2
The element remove is :23
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 2
The element remove is :65
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 2
The element remove is :93
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 3
The Queue is: EMPTY
```

```
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 2
QUEUE IS EMPTY THERE IS NO ELEMENT TO DELETE
The element remove is :-1
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 1
Enter the element you want to enter into the Queue : 12
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 3
The Queue is: 12
======================
        Menu
======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 4

Exiting.....
┌─rohit@iris ~/Programing/C/CSL201/2020-10-26
└─➤ ▌
```

Result: the Program compiled successfully and the desired output was obtained.

Experiment 11
Circular Queue Implementation Using Array

Date : 05-10-2020

Aim: To implement a circular queue using array

Data Structure used : Queue, Array

Algorithms

1. Algorithm for enqueue

Input: An Array implementation of Circular Queue (C_Q[SIZE]), with front pointing to the first element and rear pointing to the last element in and an element E to be inserted into the queue.

Output: The Circular Queue with the element E inserted at the front

Data Structure: Circular Queue

Steps:

Step 1: if((rear+1)%SIZE == front) then
    Step 1: print("The queue is full insertion not possible")
    Step 2: exit(1)
Step 2: else
    Step 1: if(rear == -1) then
        Step 1: front ++
    Step 2: EndIf
    Step 3: rear = (rear+1)%SIZE
    Step 4: C_Q[rear] = E
Step 3: EndIf

2. Algorithm for dequeue

Input: An Array implementation of Circular Queue (C_Q[SIZE]), with front pointing to the first element and rear pointing to the last element in the queue.

Output: The element E which is removed form the circular queue

Steps:

Step 1: if(front == -1) then
    Step 1: print("The Queue is empty")
    Step 2: exit(1)
Step 2: else
    Step 1: E = Q[front]
    Step 2: if(front == rear) then
        Step 1: front =-1
        Step 2: rear =-1
    Step 3: else
        Step 1: front = (front+1)%SIZE
    Step 4: endif
Step 3: endif

Program code:

```
#include<stdio.h>
#include<stdlib.h>

//Create a struct for our queue
typedef struct CQueue{
    int* Q;
    int front;
    int rear;
    int size;
} CQueue;


CQueue* initializeQueue(){
    int size = 2;

    //Create a pointer to stack
    CQueue *a = (CQueue*) malloc (sizeof(CQueue));
    if(a == NULL){
        printf("An Overflow error has ocurred while creating the CircularQueue\n");
        exit(1);
    }

    //create the array that will contain our stack
    a->Q = (int*)malloc(size*sizeof(int));
    if(a->Q == NULL){
        printf("An Overflow error has ocurred while creating the Circular Queue 
array\n");
        exit(1);
    }

    a->front = -1;
    a->rear = -1;
    a->size = size;
    return a;
}

void deleteQueue(CQueue *a){
    free(a->Q);
    free(a);
}

void enQueue(CQueue *a,int item){
    if((a->rear+1)%(a->size) == a->front){

        a->size = a->size*2;
        a->Q = realloc(a->Q,a->size);
        //printf("CircularQueue is Full \n");
        //fflush(stdout);
        if(a->Q == NULL){
            printf("An Overflow Error has occured while reallocating the array\
nEXITING!!!!!!!\n");
            exit(1);
```

```
        }
    }

    if(a->front == -1){
        a->front = 0;
    }

    a->rear = (a->rear +1)%a->size;
    a->Q[a->rear] = item;
}

int deQueue(CQueue *a){
   if(a->front == -1){
        printf("You have made a grave mistake, the CQueue was empty\n\n");
        deleteQueue(a);
        exit(1);
        return -1;

   }
   else{
        int item = a->Q[a->front];
        if(a->front == a->rear){
            a->front = -1;
            a->rear = -1;
        }
        else{
            a->front = (a->front+1)%(a->size);
        }
        return item;
   }
}

void displayQueue(CQueue *a){
    int i = a->front;

    while(i!=(a->rear+1)%(a->size)){
        printf("%d ",a->Q[i]);
        i = (i+1)%a->size;
    }
    printf("\n");
}


int menu(CQueue *a){
        int RUN=1;
        int c;      //For the corresponding choice
        int item;   //To receive the item to push or pop from the array
        while (RUN){
            printf("\n");
            printf("-----------------------------------------------\n");
            printf("Circular Queue Implementation using structure\n");
            printf("-----------------------------------------------\n");
            printf("1.Insert\n");
            printf("2.Delete\n");
            printf("3.Print the queue\n");
            printf("4.Exit\n");
```

```
            printf("Enter the required choice --> ");
            scanf("%d%*c",&c);

            switch(c){
                case 1:printf("Enter the element to be inserted into the queue -->
");
                       scanf("%d%*c",&item);
                       enQueue(a,item);
                       break;

                case 2:item = deQueue(a);
                       printf("Item removed is  is --> %d\n",item);
                       break;

                case 3:printf("The Circular Queue is --> ");
                       displayQueue(a);
                       break;

                case 4: RUN=0;
                        break;

                default:printf("Entered command is unknown");
            }

        }
        deleteQueue(a);

        printf("Finished excecuting the code ALL DONE\n");
        return RUN;
}

int main(){
    CQueue *a;
    a = initializeQueue();
    return menu(a);
}
```

Result: The Program was compiled successfully and the desired output was obtained.

Sample input/Output:

```
┌─rohit@iris ~/Programing/C/CSL201/2020-11-05
└─➤ gcc -Wall circ_queue.c -o circ_queue.o
┌─rohit@iris ~/Programing/C/CSL201/2020-11-05
└─➤ ./circ_queue.o

-----------------------------------------------
Circular Queue Implementation using structure
-----------------------------------------------
1.Insert
2.Delete
3.Print the queue
4.Exit
Enter the required choice --> 1
Enter the element to be inserted into the queue --> 12

-----------------------------------------------
Circular Queue Implementation using structure
-----------------------------------------------
1.Insert
2.Delete
3.Print the queue
4.Exit
Enter the required choice --> 1
Enter the element to be inserted into the queue --> 54

-----------------------------------------------
Circular Queue Implementation using structure
-----------------------------------------------
1.Insert
2.Delete
3.Print the queue
4.Exit
Enter the required choice --> 1
Enter the element to be inserted into the queue --> 73

-----------------------------------------------
Circular Queue Implementation using structure
-----------------------------------------------
1.Insert
2.Delete
3.Print the queue
4.Exit
Enter the required choice --> 2
Item removed is  is --> 12
```

```
-----------------------------------------------
Circular Queue Implementation using structure
-----------------------------------------------
1.Insert
2.Delete
3.Print the queue
4.Exit
Enter the required choice --> 2
Item removed is  is --> 54

-----------------------------------------------
Circular Queue Implementation using structure
-----------------------------------------------
1.Insert
2.Delete
3.Print the queue
4.Exit
Enter the required choice --> 3
The Queue is --> 73

-----------------------------------------------
Circular Queue Implementation using structure
-----------------------------------------------
1.Insert
2.Delete
3.Print the queue
4.Exit
Enter the required choice --> 2
Item removed is  is --> 73

-----------------------------------------------
Circular Queue Implementation using structure
-----------------------------------------------
1.Insert
2.Delete
3.Print the queue
4.Exit
Enter the required choice --> 2
You have made a grave mistake, the Queue was empty

┌─rohit@iris ~/Programing/C/CSL201/2020-11-05
└─➤ █
```

Experiment 12
Priority Queue Implementation Using Array

Date : 05-11-2020

Aim: To implement a priority queue using array

Data Structure used : Priority Queue, Array

Algorithms

1. Algorithm for enqueue

Input: An Array implementation of Priority Queue (P_Q[SIZE]), with front pointing to the first element and rear pointing to the last element in and an element E to be inserted into the queue, with a priority P

Output: The Priority Queue with the element E inserted at the end

Data Structure: Priority Queue

Steps:

```
Step 1: if(rear == SIZE) then
        Step 1: print("The queue is full insertion not possible")
        Step 2: exit(1)
Step 2: else
        Step 1: if(rear == -1) then
                Step 1: front ++
        Step 2: EndIf
        Step 3: ++rear
        Step 4: Q[rear].elem = E
        Step 5: Q[rear].priority = P
Step 3: EndIf
```

2. Algorithm for dequeue

Input: An Array implementation of Queue (Q[SIZE]), with front pointing to the first element and rear pointing to the last element in the queue.

Output: The element E which has the lowest priority is removed form the priority queue

Steps

```
Step 1: if(front == -1) then
        Step 1: print("The Queue is empty")
        Step 2: exit(1)
Step 2: else
        Step 1: ptr = front
        Step 2: lowestPriority = Q[front].priority
        Step 2: while(ptr<=rear)
                Step 1: if(Q[ptr].priority<lowestPriority) then
                        Step 1:  lowestPriority =  Q[ptr].priority
                        Step 2: pos = ptr
                Step 2: endif
                Step 3: ptr++
        Step 3: endWhile
        Step 4: E = Q[pos].elem
        Step 5: While(pos>front) do
                Step 1: pos--
                Step 2: Q[pos+1] = Q[pos]
```

Step 6: EndWhile
Step 7:if(front==rear) then
Step 1: front=-1
Step 2: rear = -1
Step 8:else
fornt = front +1
Step 9: endif
Step 3: endif

Description of the Algorithm:

In this algorithm the time complexity of insertion is O(1) while deletion is O(n).

Program code:

```
/* Priority Queue implemetation using dynamic array
 *  Done By : Rohit Karuankaran
 * */


#include <stdlib.h>
#include <stdio.h>
#define SIZE 32


typedef struct priority_queue
{
    int **Q;
    int size;
    int front;
    int rear;
}pqueue;

void initQueue(pqueue *q)
{
    q->size = SIZE;
    q->Q = (int**) malloc(q->size*sizeof(int*));

    for (int i = 0;i<q->size;i++)
        q->Q[i] = (int*)malloc(2*sizeof(int));

    q->front = -1;
    q->rear = -1;
}

void delQueue(pqueue *q)
{
    for(int i =0;i<q->size;i++)
        free(q->Q[i]);
    free(q->Q);
}

void enQueue(pqueue *q,int elem,int p)
{
    if(q->rear>=q->size)
```

```
    {
        printf("The Queue is full Inseriton not possible\n");
        delQueue(q);
        exit(1);

    }
    else
    {
        if(q->front==-1)
        {
            q->front=q->front+1;
        }
        q->rear = q->rear+1;
        q->Q[q->rear][0] = elem;
        q->Q[q->rear][1] = p;
        return;
    }
}

int deQueue(pqueue *q)
{
    if(q->front == -1)
    {
        printf("QUEUE IS EMPTY THERE IS NO ELEMENT TO DELETE\n");
        return -1;
    }

    else
    {
        int ptr = q->front;
        int pos =ptr;
        int priority = q->Q[q->front][1];
        while(ptr<=q->rear)
        {
            if(q->Q[ptr][1]<priority)
            {
                priority = q->Q[ptr][1];
                pos = ptr;
            }
            ptr++;
        }

        int elem = q->Q[pos][0];

        if(pos !=q->front)
        {
            while(pos>q->front)
            {
                pos--;
                q->Q[pos+1][0] =q->Q[pos][0];
                q->Q[pos+1][1] =q->Q[pos][1];
            }
        }

        if(q->front==q->rear)
        {
```

```
            q->rear =-1;
            q->front =-1;
        }
        else{
            q->front +=1;
        }
        return elem;
    }
}

void displayQueue(pqueue *q)
{
    int i = q->front;
    if(q->front==-1)
    {
        printf("EMPTY");
        return;
    }
    while(i>=0&&i<=q->rear)
    {
        printf("%d ",q->Q[i][0]);
        i++;
    }

}

int main()
{
    pqueue *myQueue = (pqueue*) malloc(sizeof(pqueue));

    int RUN = 1;
    int elem;
    int priority;
    int choice;
    initQueue(myQueue);
    while(RUN)
    {
        printf("======================\n");
        printf("        Menu\n");
        printf("======================\n\n");
        printf("1.Enter into the queue\n");
        printf("2.Remove from the queue\n");
        printf("3.Display the queue\n");
        printf("4.Exit\n");
        printf("Enter your choice : ");

        scanf("%d%*c",&choice);
        switch(choice)
        {
            case 1: printf("Enter the element you want to enter into the Queue :
");
                    scanf("%d%*c",&elem);
                    printf("Enter the priority of the element : ");
                    scanf("%d%*c",&priority);
                    enQueue(myQueue,elem,priority);
                    break;
```

```c
            case 2: elem = deQueue(myQueue);
                    printf("The element remove is :%d\n",elem);
                    break;

            case 3: printf("The Queue is: ");
                    displayQueue(myQueue);
                    printf("\n");
                    break;
            case 4: RUN = 0;
                    break;
            default: printf("Enter a valid input\n\n");
        }
    }

    /*
    insert(myQueue,32);
    insert(myQueue,21);
    displayQueue(myQueue);
    */
    delQueue(myQueue);
    printf("\nExiting.....\n");

}
```

Result: The Program compiled successfully and the desired output was obtained.

<u>Sample input/Output:</u>

```
┌─rohit@iris ~/Programing/C/CSL201/2020-11-05
└─➤ gcc -Wall priority_queue.c -o priority_queue.o
┌─rohit@iris ~/Programing/C/CSL201/2020-11-05
└─➤ ./priority_queue.o
=======================
        Menu
=======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 1
Enter the element you want to enter into the Queue : 12
Enter the priority of the element : 4
=======================
        Menu
=======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 1
Enter the element you want to enter into the Queue : 0
Enter the priority of the element : 0
=======================
        Menu
=======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 2
The element remove is :0
=======================
        Menu
=======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 1
Enter the element you want to enter into the Queue : 34
Enter the priority of the element : 1
```

```
=======================
        Menu
=======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 2
The element remove is :34
=======================
        Menu
=======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 2
The element remove is :12
=======================
        Menu
=======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 2
QUEUE IS EMPTY THERE IS NO ELEMENT TO DELETE
The element remove is :-1
=======================
        Menu
=======================

1.Enter into the queue
2.Remove from the queue
3.Display the queue
4.Exit
Enter your choice : 4

Exiting.....
┌─rohit@iris ~/Programing/C/CSL201/2020-11-05
└─➤ █
```

<u>Experiment 13</u>
<u>Deque Implementation Using Array</u>

Date : 05-10-2020

Aim: To implement a Deque using array

Data Structure used : Deque, Array

Algorithms

1. Algorithm for insertion in Front

Input: An Array implementation of Deque (DQ[SIZE]), with front pointing to the first element and rear pointing to the last element in and an element E to be inserted into the queue.

Output: The Deque with the element E inserted at the front

Data Structure: Deque

Steps:

```
Step 1: if(front == 0) then
        Step 1: print(“The insertion at front is not possible”)
        Step 2: exit(1)
Step 2: else
        Step 1: if(rear == -1) then          //If there is no element in the deque then insetion at
                Step 1: front ++             //front is possible
        Step 2: else
                Step 1: front - -
        Step 3: EndIf
        Step 4: DQ[front] = E
Step 3: EndIf
```

2. Algorithm for insertion in Rear

Input: An Array implementation of Deque (DQ[SIZE]), with front pointing to the first element and rear pointing to the last element in and an element E to be inserted into the queue.

Output: The Deque with the element E inserted at the rear

Data Structure: Deque

Steps:

```
Step 1: if(rear == SIZE) then
        Step 1: print(“The queue is full insertion not possible”)
        Step 2: exit(1)
Step 2: else
        Step 1: if(rear == -1) then
                Step 1: front ++
        Step 2: EndIf
        Step 3: DQ[++rear] = E
```

Step 3: EndIf

3. Algorithm for removing from front

Input: An Array implementation of Deque (DQ[SIZE]), with front pointing to the first element and rear pointing to the last element in the queue.

Output: The element E which is removed form the front of the deque

Data Structure: Deque

Steps

Step 1: if(front == -1) then
    Step 1: print("The Deque is empty")
    Step 2: exit(1)
Step 2: else
    Step 1: E = DQ[front]
    Step 2: if(front == rear) then
        Step 1: front =-1
        Step 2: rear =-1
    Step 3: else
        Step 1: front--
    Step 4: endif
Step 3: endif

4. Algorithm for removing from the rear

Input: An Array implementation of Deque (DQ[SIZE]), with front pointing to the first element and rear pointing to the last element in the queue.

Output: The element E which is removed form the rear of the deque

Data Structure: Deque

Steps

Step 1: if(rear == -1) then
    Step 1: print("The Deque is empty")
    Step 2: exit(1)
Step 2: else
    Step 1: E = DQ[rear]
    Step 2: if(front == rear) then
        Step 1: front =-1
        Step 2: rear =-1
    Step 3: else
        Step 1: rear - -
    Step 4: endif
Step 3: endif

Program code:

```
/* Deque implemetation using dynamic array
 *  Done By : Rohit Karuankaran
 * */


#include <stdlib.h>
#include <stdio.h>
#define SIZE 50


typedef struct deque_structure_datatype
{
    int *Q;
    int size;
    int front;
    int rear;
}deque;

void initQueue(deque *dq)
{
    dq->size = SIZE;
    dq->Q = (int*) malloc(dq->size*sizeof(int));
    dq->front = -1;
    dq->rear = -1;
}

void delQueue(deque *dq)
{
    free(dq->Q);
}

void insertRear(deque *dq,int elem)
{
    if(dq->rear>=dq->size)
    {
        printf("The Queue is full Inseriton not possible\n");
        //incrSize(dq);
    }
    else
    {
        if(dq->front==-1)
        {
            dq->front=dq->front+1;
        }
        dq->rear = dq->rear+1;
        dq->Q[dq->rear] = elem;
        return;
    }
}

void insertFront(deque *dq,int elem)
{
    if(dq->front==0)
    {
```

```
        //This is the condition  if there is somthin inserted
        printf("Insertion at front not possible\n");
    }
    else
    {
        if(dq->rear == -1)
        {
            dq->rear= dq->rear+1;
        }
        if(dq->front == -1)
        {
            dq->front=dq->front+1;
        }
        else
        {
            dq->front = dq->front-1;
        }
        dq->Q[dq->front] =  elem;
        return;
    }
}

int deleteFront(deque *dq)
{
    if(dq->front == -1)
    {
        printf("QUEUE IS EMPTY THERE IS NO ELEMENT TO DELETE\n");
        return -1;
    }

    else
    {
        int elem = dq->Q[dq->front];

        if(dq->front==dq->rear)
        {
            dq->front = -1;
            dq->rear = -1;
        }

        else
            dq->front=dq->front+1;
        return elem;
    }
}

int deleteRear(deque *dq)
{
    if(dq->rear ==-1)
    {
        printf("QUEUE IS EMPTY THERE IS NO ELEMENT TO DELETE\n");
        return -1;
    }
    else
    {
        int elem = dq->Q[dq->rear];
```

```
        if(dq->front==dq->rear)
        {
            dq->front = -1;
            dq->rear = -1;
        }
        else
        {
            dq->rear = dq->rear-1;
        }
        return elem;
    }
}

void displayQueue(deque *dq)
{
    int i = dq->front;
    if(dq->front)
    {
        printf("EMPTY");
        return;
    }
    while(i>=0&&i<=dq->rear)
    {
        printf("%d ",dq->Q[i]);
        i++;
    }
}

int main()
{
    deque *myDeque = (deque*) malloc(sizeof(deque));
    int RUN = 1;
    int elem;
    int choice;
    initQueue(myDeque);
    while(RUN)
    {
        printf("\n======================\n");
        printf("        Menu\n");
        printf("======================\n");
        printf("1.Enter into the front\n");
        printf("2.Enter into the rear\n");
        printf("3.Remove from the front\n");
        printf("4.Remove from the rear\n");
        printf("5.Display the deque\n");
        printf("6.Exit\n");
        printf("Enter your choice : ");

        scanf("%d%*c",&choice);
        switch(choice)
        {
            case 1: printf("Enter the element you want to enter into the front :
");
                    scanf("%d%*c",&elem);
                    insertFront(myDeque,elem);
```

```
                    break;

            case 2: printf("Enter the element you want to enter into the rear: ");
                    scanf("%d%*c",&elem);
                    insertRear(myDeque,elem);
                    break;

            case 3: elem = deleteFront(myDeque);
                    printf("The element remove is :%d\n",elem);
                    break;

            case 4: elem = deleteRear(myDeque);
                    printf("The element remove is :%d\n",elem);
                    break;

            case 5: printf("The Queue is: ");
                    displayQueue(myDeque);
                    printf("\n");
                    break;

            case 6: RUN = 0;
                    break;
            default: printf("Enter a valid input\n\n");
        }
    }
    /*
    insert(myDeque,32);
    insert(myDeque,21);
    displayQueue(myDeque);
    */
    delQueue(myDeque);
    printf("\nExiting.....\n");
}
```

Sample input and output:

```
┌rohit@iris ~/Programing/C/CSL201/2020-11-05
└➤ ./a.out

======================
        Menu
======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 1
Enter the element you want to enter into the front : 12

======================
        Menu
======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 1
Enter the element you want to enter into the front : 2
Insertion at front not possible

======================
        Menu
======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 2
Enter the element you want to enter into the rear: 54

======================
        Menu
======================
1.Enter into the front
2.Enter into the rear
```

```
=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 5
The Queue is: 12 54

=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 2
Enter the element you want to enter into the rear: 93

=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 3
The element remove is :12

=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 1
```

```
=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 4
The element remove is :93

=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 4
The element remove is :54

=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 5
The Queue is: 12
```

```
=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 5
The Queue is: 12

=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 1
Enter the element you want to enter into the front : 23
Insertion at front not possible

=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 3
The element remove is :12
```

```
=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 3
QUEUE IS EMPTY THERE IS NO ELEMENT TO DELETE
The element remove is :-1

=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 4
QUEUE IS EMPTY THERE IS NO ELEMENT TO DELETE
The element remove is :-1

=======================
        Menu
=======================
1.Enter into the front
2.Enter into the rear
3.Remove from the front
4.Remove from the rear
5.Display the deque
6.Exit
Enter your choice : 6

Exiting.....
┌rohit@iris ~/Programing/C/CSL201/2020-11-05
└➤ █
```

<u>Result:</u> the Program compiled successfully and the desired output was obtained.

Experiment 14
Implementation Of Linked List

Date: 10-11-2020

Aim: Implementation of linked list

Data Structure Used: Linked List

Operation Used: Comparisons

Algorithm:

Algorithm for InsertFront

Input: Header Node of a linked list (LL) and the ITEM to be inserted
Output: Linked List with the new node inserted after the Header node
Data Structure: Linked List

Step 1 : Start
Step 2: new = GetNode(Node)
Step 3: if(new==NULL) then
    Step 1: Print("No Memory space available")
    Step 2: Stop
Step 4: else
    Step 1: new→data = ITEM
    Step 2: new→link = Header→link
    Step 3: Header→link = new
Step 5: endif
Step 6: Stop

Description of the Algorithm: This Algorithm inserts a node just after the header node

Algorithm for InsertBack

Input: Header Node of a linked list (LL) and the ITEM to be inserted
Output: Linked List with the new node inserted at the end of the List
Data Structure: Linked List

Step 1 : Start
Step 2: new = GetNode(Node)
Step 3: if(new==NULL) then
    Step 1: Print("No Memory space available")
    Step 2: Stop
Step 4: else
    Step 1;ptr = Header
    Step 2: while(ptr→ link !=NULL) do
        Step 1: ptr=ptr→link
    Step 3: endWhile
    Step 4:new→data = ITEM
    Step 5: new→link = NULL
    Step 6: ptr→link = new
Step 5: endif
Step 6: Stop

Description of the Algorithm: This algorithm goes to the end of the List and inserts a node after the last node

Algorithm for InsertFront

Input: Header Node of a linked list (LL), the ITEM to be inserted and the position (POS)

Output: Linked List with the new node inserted at the corresponding position
Data Structure: Linked List

Step 1 : Start
Step 2: new = GetNode(Node)
Step 3: if(new==NULL) then
    Step 1: Print("No Memory space available")
    Step 2: Stop
Step 4: else
    Step 1: i=-1
    Step 2: ptr = Header
    Step 3: while(i<pos-1 and ptr!=NULL) then
        Step 1: i++
        Step 2: ptr=ptr→link
    Step 4: endwhile
    Step 5: if(ptr!=NULL) then
        Step 1: new→data = ITEM
        Step 2: new→link = ptr→link
        Step 3: ptr→link = new
    Step 6: else
        Step 1: print("Given position is not found")
        Step 2: Stop
    Step 7: endif
Step 5: endif
Step 6: Stop

Description of the Algorithm: This algorithm traversed the List, on reaching the node at the index position passed it inserts a new node at that position. Eg: if the List is "34 21 56 12" and assume the elements are indexed from 0 (even though it is a linked list and indexing of elements don't make any sense) if I want to insert 23 at position 2. The resulting Linked list will be
"34 21 23 56 12".

Algorithm for DeleteFront

Input: Header Node of a linked list (LL)
Output: The item removed from the list
Data Structure: Linked List

Step 1 : Start
Step 2: if(Header→link ==NULL) then
    Step 1: print("Linked List is empty")
    Step 2: Stop
Step 3: else
    Step 1: ptr = Header→link
    Step 2: Header→link = ptr→ link
    Step 3: ITEM = ptr→data
    Step 4: ReturnNode(ptr)
    Step 5: return ITEM
Step 4: endif
Step 5: Stop

Description of the Algorithm: This algorithm deletes the node just after the header node

Algorithm for DeleteRear

Input: Header Node of a linked list (LL)
Output: The item removed from the end of the list
Data Structure: Linked List

Step 1 : Start
Step 2: if(Header→link ==NULL) then
    Step 1: print("Linked List is empty")
    Step 2: Stop
Step 3: else
    Step 1: ptr = Header →link
    Step 2: ptr1 = Header
    Step 3: while(ptr→link!=NULL) do
        Step 1: ptr1=ptr
        Step 2: ptr = ptr→link
    Step 4: EndWhile
    Step 5: ITEM = ptr→data
    Step 6: ptr1→link = ptr→link
    Step 7: ReturnNode(ptr)
    Step 8: return ITEM
Step 4: EndIf
Step 5 : Stop

Description of the Algorithm: This algorithm deletes the Node at the end of the linked list

Algorithm for Delete from a position

Input: Header Node of a linked list (LL) and the position of the node to be removed
Output: The item removed from the specified position of the list
Data Structure: Linked List

Step 1 : Start
Step 2: if(Header→link == NULL)
    Step 1: Print("The List Is Empty")
    Step 2: Stop
Step 3: else
    Step 1: i=-1
    Step 2: ptr = Header
    Step 3: while(i<pos-1 and ptr!=NULL) then
        Step 1: i++
        Step 2: ptr=ptr→link
    Step 4: endwhile
    Step 5:if(ptr→link ==NULL)
        Step 1: ITEM = ptr->link→ data
        Step 2: ptr1 = ptr→link
        Step 3: ptr→link = ptr1→link
        Step 4: ReturnNode(ptr1)
        Step 5:return(ITEM)
    Step 6: else
        Step 1: Print("Index Out Of Bounds")
        Step 2: Stop
    Step 7:endif
Step 4: endif
Step 5: Stop

Description of the Algorithm: Just like the insertion at any position algorithm passing the position of the element to be deleted will remove the element. It takes a pointer (ptr) to the element right before the one to be deleted and then links the link part of ptr to the link of the element to be deleted.

Program Code:

```
/*******************************
 * Linked List Implementation
 * Done By: Rohit Karunakaran
 * *****************************/

#include<stdio.h>
#include<stdlib.h>

typedef struct Linked_List_Node
{
    struct Linked_List_Node *link;
    int data;
}Node;

void initList(Node* Header)
{
    //Header = (Node*) malloc (sizeof(Node));
    Header->link = NULL;
    Header->data = 0;
}


//Insertion Algorithms
void insertStart(Node *Header,int val)
{
    Node *new_node = (Node*) malloc(sizeof(Node));

    if(new_node!=NULL)
    {
        new_node->data = val;
        new_node->link = NULL;
        Node* ptr = Header->link;
        Header->link = new_node;
        new_node->link=ptr;
    }
    else
    {
        printf("Insertion Not Possible\n");
        exit(1);
    }
    return ;
}

void insertAt(Node *Header,int val,int pos) //Insert at a specified position from
the header node
{
    Node *new_node = (Node*) malloc(sizeof(Node));

    if(new_node!=NULL)
    {
        Node* ptr = Header;
        int index = -1;
        while(index<pos-1 && ptr!=NULL)
        {
```

```
            ptr=ptr->link;
            index ++;
        }
        if(ptr !=NULL)
        {
            new_node->link = ptr->link;
            new_node->data = val;
            ptr->link =new_node;
        }
        else
        {
            printf("Given position is not found \nExiting......\n");
            exit(1);
        }
    }
    else
    {
        printf("Insertion Not Possible");
        exit(1);
    }
    return ;

}

void insertEnd(Node *Header,int val)
{
    Node *new_node = (Node*) malloc(sizeof(Node));

    if(new_node!=NULL)
    {
        new_node->data = val;
        new_node->link = NULL;
        Node* ptr=Header;

        while(ptr->link != NULL)
        {
            ptr = ptr->link;
        }

        ptr->link = new_node;
    }

    else
    {
        printf("Insertion not possible");
        exit(1);
    }

    return;
}

//Deletion Algorithms
int deletionBegin(Node *Header)
{
    if(Header->link == NULL)
    {
```

```
        printf("Deletion not possible. The list is empty");
        exit(0);
        return 0;
    }
    else
    {
        Node* ptr = Header->link;
        Header->link = ptr->link;
        int elem = ptr->data;
        free(ptr);
        return elem;
    }
}

int deletionAt(Node* Header, int pos)
{
    if(Header->link == NULL)
    {
        printf("Deletion not possible. The list is empty");
        exit(0);
        return 0;
    }
    else
    {
        int index = -1;
        Node* ptr = Header;
        while(index<pos-1&&ptr!=NULL)
        {
            ptr=ptr->link;
            index++;
        }
        if(ptr->link!=NULL)
        {
            int elem = ptr->link->data;
            Node* red = ptr->link;
            ptr->link = ptr->link->link;
            free(red);
            return elem;
        }
        else
        {
            printf("Index Is out of Bounds \n");
            exit(1);
            return 0;
        }
    }
}

int deletionEnd(Node* Header)
{
    if(Header->link == NULL)
    {
        printf("Deletion not possible. The list is empty");
        exit(0);
        return 0;
    }
```

```
    else
    {
        Node* ptr=Header->link;
        Node* ptr1=Header;
        while(ptr->link!=NULL)
        {
            ptr1=ptr;
            ptr=ptr->link;
        }
        int elem = ptr->data;
        ptr1->link = NULL;
        free(ptr);
        return elem;
    }
}

void displayList(Node* Header)
{
    Node* ptr = Header->link;
    if(ptr!=NULL)
    {
        printf("The List is : ");
        while(ptr!=NULL)
        {
            printf("%d ",ptr->data);
            ptr=ptr->link;
        }
        printf("\n");
    }
    else
    {
        printf("The Linked list is empty\n");
    }
}

int menu(Node* Header)
{
    int RUN = 1;
    while(RUN)
    {
        printf("\n");
        printf("=============================\n");
        printf("            MENU             \n");
        printf("=============================\n");
        printf("1.Insert At Begining\n");
        printf("2.Insert At End\n");
        printf("3.Insert At Position\n");

        printf("4.Delete From Begining\n");
        printf("5.Delete From End\n");
        printf("6.Delete From Position\n");
        printf("7.Display the linked List\n");
        printf("8.Exit\n");
        printf("Enter Choice: ");
        int choice;
        int elem;
```

```
    int pos;
    scanf("%d%*c",&choice);

    switch(choice)
    {
        case 1: printf("Enter the element to be inserted: ");
                scanf("%d%*c",&elem);
                insertStart(Header,elem);
                printf("\n");
                break;

        case 2: printf("Enter the element to be inserted: ");
                scanf("%d%*c",&elem);
                insertEnd(Header,elem);
                printf("\n");
                break;

        case 3: printf("Enter the element to be inserted: ");
                scanf("%d%*c",&elem);
                printf("Enter the postion to insert %d : ",elem);
                scanf("%d%*c",&pos);
                insertAt(Header,elem,pos);
                printf("\n");
                break;

        case 4: elem = deletionBegin(Header);
                printf("The Element removed is %d",elem);
                printf("\n");
                break;

        case 5: elem = deletionEnd(Header);
                printf("The Element removed is %d",elem);
                printf("\n");
                break;


        case 6: printf("Enter the postion of the element to be deleted : ");
                scanf("%d%*c",&pos);
                elem = deletionAt(Header,pos);
                printf("The Element removed is %d",elem);
                printf("\n");
                break;

        case 7: displayList(Header);
                break;

        case 8: RUN=0;
                break;
        default: printf("Enter a valid choice\n");
                printf("\n");
                 break;

    }

}
printf("Exiting........\n");
```

```
    return RUN;
}

int main()
{
    Node *Header = (Node*)malloc(sizeof(Node));
    initList(Header);
    return menu(Header);
}
```

Result: The Program is successfully compiled and the desired result is obtained

Sample Input and output

```
============================
            MENU
============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 1
Enter the element to be inserted: 39


============================
            MENU
============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 2
Enter the element to be inserted: 72


============================
            MENU
============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 3
Enter the element to be inserted: 93
Enter the postion to insert 93 : 1
```

```
=============================
            MENU
=============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 7
The List is : 39 93 72

=============================
            MENU
=============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 6
Enter the postion of the element to be deleted2
The Element removed is 72

=============================
            MENU
=============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 5
The Element removed is 93
```

```
=============================
            MENU
=============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 5
The Element removed is 93

=============================
            MENU
=============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 4
The Element removed is 39

=============================
            MENU
=============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 7
The Linked list is empty
```

```
==============================
            MENU
==============================
1.Insert At Begining
2.Insert At End
3.Insert At Position
4.Delete From Begining
5.Delete From End
6.Delete From Position
7.Display the linked List
8.Exit
Enter Choice: 8
Exiting........%
```

Experiment 15
Implementation Of Stack Using Linked List

Date: 10-11-2020

Aim: Implementation of Stack using Linked List

Data Structure Used: Stack

Operation Used: Comparisons

Algorithm:

Algorithm for Push

Input: The Stack (S) implemented using Linked List, the pointer to the element at the top (TOP), ITEM to be inserted

Output: The Stack (S) with ITEM inserted at the top.

Data Structure: Stack and linked list

Steps:

Step 1: Start
Step 2: new = GetNode(Node)
Step 3: if(new!=NULL) then
    Step 1: new→data = ITEM
    Step 2: new→link = NULL
    Step 3: if(Top!=NULL) then
        Step 1: new→link = Top→Link
    Step 4: endif
    Step 5: Top = new
Step 4: else
    Step 1: print("Insertion not possible")
    Step 2: exit(1)
Step 5: endif
Step 6: Stop

Description of the algorithm

This algorithm places a new Node 'new' with the value of ITEM and the link part pointing to the previous Top element in the Stack (S) making it the new Top element

Algorithm for Pop

Input: The Stack (S) implemented using Linked List, the pointer to the element at the top (TOP)

Output: The Stack (S) with , ITEM to be removed and the ITEM

Data Structure: Stack and Linked list

Steps:

Step 1:Start
Step 2: if(Top == NULL)
    Step 1: print("The Stack is empty")
    Step 2: exit
Step 3: else
    Step 1: ITEM = Top→data
    Step 2: remove = Top
    Step 3: Top= Top→link
    Step 4: RetrunNode(remove)
    Step 5: return ITEM

Step 4:endif
Step 5: Stop

Description of the algorithm:

This algorithm stores the value of the current Top item in a variable, and stores the value in a variable remove. Then it assigns Top to Top→Link and returns the remove variable to the memory.

Program Code:

```
/*****************************************
 * Stack Implementation using a Linked List
 * Done By: Rohit Karunakaran
 * ****************************************/

#include<stdio.h>
#include<stdlib.h>

typedef struct Linked_List_Node
{
    struct Linked_List_Node *link;
    int data;
}Node;

typedef struct Linked_Stack
{
    Node *Top;
}Stack;

Stack* initStack()
{
    Stack *s = (Stack*) malloc (sizeof(Stack));
    s->Top = NULL;
    return s;
}


//Insertion Algorithms
void push(Stack *s,int val)
{
    Node *new_node = (Node*) malloc(sizeof(Node));

    if(new_node!=NULL)
    {
        new_node->data = val;
        new_node->link = s->Top;
        s->Top = new_node;
    }
    else
    {
        printf("Stack Is Full");
        exit(1);
    }
    return ;
}
```

```
//Deletion Algorithms
int pop(Stack *s)
{
    if(s->Top == NULL)
    {
        printf("Stack Is Empty");
        exit(0);
        return 0;
    }
    else
    {
        Node* ptr = s->Top;
        s->Top = s->Top->link;
        int elem = ptr->data;
        free(ptr);
        return elem;
    }
}

void displayStack(Stack *s)
{
    Node* ptr = s->Top;
    if(ptr!=NULL)
    {
        printf("The Stack is: Top -> ");
        while(ptr!=NULL)
        {
            if(ptr==s->Top){
                printf("%d\n",ptr->data);
            }
            else{
                printf("                     %d\n",ptr->data);
            }
            ptr=ptr->link;
        }
        printf("\n");
    }
    else
    {
        printf("The Stack is empty\n");
    }
}

int menu(Stack* s)
{
    int RUN = 1;
    while(RUN)
    {
        printf("\n");
        printf("=============================\n");
        printf("            MENU             \n");
        printf("=============================\n");
        printf("1.Push\n");
        printf("2.Pop\n");
        printf("3.Display the stack\n");
```

```
        printf("4.Exit\n");
        printf("Enter Choice: ");
        int choice;
        int elem;
        scanf("%d%*c",&choice);

        switch(choice)
        {
            case 1: printf("Enter the element to be inserted: ");
                    scanf("%d%*c",&elem);
                    push(s,elem);
                    printf("\n");
                    break;
            case 2: elem = pop(s);
                    printf("The Element removed is %d",elem);
                    printf("\n");
                    break;
            case 3: displayStack(s);
                    break;
            case 4: RUN=0;
                    break;
            default: printf("Enter a valid choice\n");
                     printf("\n");
                     break;

        }

    }
    printf("Exiting........");
    return RUN;
}


int main()
{
    Stack *s = initStack();
    return menu(s);
}
```

Result: The Program is successfully compiled and the desired result is obtained

Sample Input/Output

```
rohit@iris ~/Programing/C/CSL201/2020-11-10
 ./LinkedStack.o

=============================
            MENU
=============================
1.Push
2.Pop
3.Display the stack
4.Exit
Enter Choice: 1
Enter the element to be inserted: 32


=============================
            MENU
=============================
1.Push
2.Pop
3.Display the stack
4.Exit
Enter Choice: 1
Enter the element to be inserted: 75


=============================
            MENU
=============================
1.Push
2.Pop
3.Display the stack
4.Exit
Enter Choice: 3
The Stack is: Top -> 75
                     32


=============================
            MENU
=============================
1.Push
2.Pop
3.Display the stack
4.Exit
Enter Choice: 2
The Element removed is 75
```

```
=============================
            MENU
=============================
1.Push
2.Pop
3.Display the stack
4.Exit
Enter Choice: 3
The Stack is: Top -> 32


=============================
            MENU
=============================
1.Push
2.Pop
3.Display the stack
4.Exit
Enter Choice: 2
The Element removed is 32

=============================
            MENU
=============================
1.Push
2.Pop
3.Display the stack
4.Exit
Enter Choice: 4
Exiting........%
rohit@iris ~/Programing/C/CSL201/2020-11-10
```

<u>Experiment 16</u>
<u>Queue Implementation Using Linked List</u>

Date : 12-11-2020

Aim: To implement a Queue using Linked List

Data Structure used : Queue, Linked List

Algorithms

1. Algorithm for Enqueue

Input: An Array implementation of Queue (Q), with Front pointing to the first element and Rear pointing to the last element in and an element ITEM to be inserted into the queue.

Output: The Queue with the element ITEM inserted at the rear

Data Structure: Queue, Linked List

Steps:

Step 1: Start
Step 2: new = GetNode(Node)
Step 3: if(new == NULL)
    Step 1: Print("Can nont Insert a new node")
    Step 2: Exit(1)
Step 4: else
    Step 1: new→data = ITEM
    Step 2: new→ Link = NULL
    Step 3: if(Front==NULL) then
        Step 1: Front = new
    Step 4: else
        Step 1: Rear→ link = new
    Step 5: endif
    Step 6: Rear = new
Step 5: endif
Step 6: Stop

2. Algorithm for dequeue

Input: An Array implementation of Queue (Q), with Front pointing to the first element and Rear pointing to the last element in the queue.

Output: The element ITEMwhich is removed form the Front of the queue

Steps

Step 1: if(front == NULL) then
    Step 1: print("The Queue is empty")
    Step 2: exit(1)
Step 2: else
    Step 1: ITEM = Front→data
    Step 2: rem = Front
    Step 3: if(Front==Rear)then
        Step 1:Rear =NULL
        Step 2: Front = NULL
    Step 4:else
        Step 1: Front = Front→link

Step 5:endif
Step 6: ReturnNode(rem)
Step 7: return ITEM
Step 3: endif
Step 4: Stop

Result: the Program compiled successfully and the desired output was obtained.

Program code:

```
/****************************************
 * Queue Implementation Using Linked List
 * Done By: Rohit Karunakaran
 * ***************************************/


#include<stdio.h>
#include<stdlib.h>

typedef struct Linked_List_Node
{
    struct Linked_List_Node *link;
    int data;
}Node;

typedef struct Linked_Queue
{
    Node* Front;
    Node* Rear;
}Queue;

Queue* initQueue()
{
    Queue *q = (Queue*) malloc (sizeof(Queue));
    q->Front = NULL;
    q->Rear = NULL;
    return q;
}


//Insertion Algorithm
void enQueue(Queue *q,int val)
{
    Node *new_node = (Node*) malloc(sizeof(Node));

    if(new_node!=NULL)
    {
        new_node->link=NULL;
        new_node->data = val;
        if(q->Rear == NULL)
        {
            q->Front = new_node;
        }
        else
```

```
        {
             q->Rear->link = new_node;
        }
        q->Rear = new_node;
    }
    else
    {
        printf("Queue Is Full");
        exit(1);
    }
    return ;
}

//Deletion Algorithm
int deQueue(Queue *q){
    if(q->Front == NULL)
    {
        printf("Queue Is Empty");
        exit(0);
        return 0;
    }
    else
    {
        Node* ptr = q->Front;
        q->Front = q->Front->link;
        int elem = ptr->data;
        free(ptr);
        return elem;
    }
}

void displayQueue(Queue *q){
    Node* ptr = q->Front;
    if(ptr!=NULL)
    {
        printf("The Queue is: ");
        while(ptr!=NULL)
        {
            printf("%d",ptr->data);
            ptr=ptr->link;
        }
        printf("\n");
    }
    else
    {
        printf("The Queue is empty\n");
    }
}

int menu(Queue* q){
    int RUN = 1;
    while(RUN)
    {
        printf("\n");
        printf("============================\n");
        printf("            MENU             \n");
```

```
        printf("=============================\n");
        printf("1.Enqueue\n");
        printf("2.Dequeue\n");
        printf("3.Display the Queue\n");
        printf("4.Exit\n");
        printf("Enter Choice: ");
        int choice;
        int elem;
        scanf("%d%*c",&choice);

        switch(choice)
        {
            case 1: printf("Enter the element to be inserted: ");
                    scanf("%d%*c",&elem);
                    enQueue(q,elem);
                    printf("\n");
                    break;
            case 2: elem = deQueue(q);
                    printf("The Element removed is %d",elem);
                    printf("\n");
                    break;
            case 3: displayQueue(q);
                    break;
            case 4: RUN=0;
                    break;
            default: printf("Enter a valid choice\n");
                     printf("\n");
                     break;

        }

    }
    printf("Exiting........");
    return RUN;
}

int main(){
    Queue *q = initQueue();
    return menu(q);
}
```

Sample Input/Output

```
rohit@iris ~/Programing/C/CSL201/2020-11-12
➤ gcc -Wall -g LinkedQueue.c -o LinkedQueue.o
rohit@iris ~/Programing/C/CSL201/2020-11-12
➤ ./LinkedQueue.o

============================
            MENU
============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 1
Enter the element to be inserted: 34


============================
            MENU
============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 1
Enter the element to be inserted: 82


============================
            MENU
============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 3
The Queue is: 34 -> 82

============================
            MENU
============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 2
The Element removed is 34
```

```
=============================
            MENU
=============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 1
Enter the element to be inserted: 56


=============================
            MENU
=============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 3
The Queue is: 82 -> 56

=============================
            MENU
=============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 1
Enter the element to be inserted: 78


=============================
            MENU
=============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 3
The Queue is: 82 -> 56 -> 78

=============================
            MENU
=============================
1.Enqueue
2.Dequeue
```

```
=============================
            MENU
=============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 2
The Element removed is 82

=============================
            MENU
=============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 2
The Element removed is 56

=============================
            MENU
=============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 2
The Element removed is 78

=============================
            MENU
=============================
1.Enqueue
2.Dequeue
3.Display the Queue
4.Exit
Enter Choice: 2
Queue Is Empty%
rohit@iris ~/Programing/C/CSL201/2020-11-12
➤
```

Experiment 17
Polynomials using Linked List

Date: 19-11-2020

Aim: To receive two polynomials and print their sum and product

Data Structure Used: Linked List

Operation Used: Comparisons, addition, multiplication

Algorithm for Addition (ADD_POLY):

Input: Two polynomial, A and B with the terms as the nodes of a linked list and 'a' denoting the number of terms in polynomial A and 'b' denoting the number of terms in polynomial 'B'

Output: Sum of the polynomial 'C'

Data Structure Used : Linked List

Step 1 : Start
Step 2 : Receive two polynomial in linked list
Step 3 : i = A → Header //Pointer to the header of polynomial A
Step 4 : j = B → Header //Pointer to the polynomial B
Step 5 : while i =! NULL and j=! NULL
  Step 1: new=GetNode(Node)
  Step 1 : if i→ pow == j→ pow
    Step 1: new→ pow = i→ pow
    Step 2: new → coeff = i → coeff+j → coeff
    Step 3: C.addNode(new)
    Step 4: i=i→link
    Step 5: j=j->link
  Step 2: else if i→pow < j→pow
    Step 1: new → pow = j→pow
    Step 2: new→ coeff=j→ coeff
    Step 3: C.addNode(new)
    Step 4: j=j→ link
  Step 3: else if i → pow > j → pow
    Step 1: new→coeff = i->coeff
    Step 2: new→pow = i→ pow
    Step 3: i=i → link
    Step 4: C.addNode(new)
  Step 4: Endif
Step 6 : EndWhile

Step 7 : while i!=NULL
  Step 1: new→coeff = i → coeff
  Step 2: new→pow = i → pow
  Step 3: i = i → link
  Step 4: C.addNode(new)
Step 8: EndWhile
Step 9: while j!=NULL
  Step 1: new = GetNode(Node)
  Step 2: new → pow = j→pow
  Step 3: new→ coeff=j→ coeff
  Step 4: C.addNode(new)
  Step 5: j=j→ link
Step 10 : EndWhile
Step 11 : return c
Step 12 : Stop

Description of the Algorithm:

In this algorithm the polynomials' terms are the nodes of a linked list and there are 2 pointers i and j, which points to the nodes of A and B respectively. If the powers of a term in A and B are equal then the coefficeint are added and the sum is put into a new node (new). Which is then added to the end of the resultant polynomial C. If the coefficientof the term in A is greater than the term in B then the term is added to the end of B. Likewise for B also.

Algorithm for Multiplication(MUL_POLY):

Input: A and B, two polynomials with the terms as nodes of a linked list with pow being the power of the term and coeff being the coefficient

Output: Polynomial C, with

Data Structure used: Linked list

Steps:

Step 1:Start
Step 2: receive two polynomials
Step 3:i = A→ head
Step 4: j = B→ head
Step 5: initialize C as a polynomial with 0 as the only term
Step 6: k = 0
Step 7: while(k<B→numberOfTerms)
    Step 1: j = B → head
    Step 2: while(j!=NULL)
        Step 1: new = GetNode(Node)
        Step 2; new→pow = i→pow+j→ pow
        Step 3: new → coeff = i→ coeff * i→ coeff
        Step 4: temp.addNode(new)
        Step 5: j = j→link
    Step 3: End While
    Step 4: C= ADD_POLY(C,temp)
    Step 5: i=i→ link
    Step 6: k++
Step 8: EndWhile
Step 9: return C
Step 10: Stop

Description of the Algorithm:

The polynomial product of $(6X^2+1)*(7X^2+3X+1)$ can be expressed as, $0+(6X^2+1)*(7X^2)+(6X^2+1)*(3X+(6X^2+1)*1$. Here we just need to multiply the first polynomial with one of the terms from the second and feed the result obtained before and the result obtained now to the addition function and then after the algorithm has been executed number of times as there are number of terms in B. We get the product of the polynomial.

Result: the Program is successfully compiled and the desired output is obtained.

Program/ Source Code:

```
/************************************
 * Sum And Product of a Polynomial
 * Done By Rohit Karunakaran
 ************************************/

#include<stdio.h>
#include<stdlib.h>

/*  Input : 2 polynomials of the form
 *          a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an
 *  Output: First polynomial the second polynomial and there sum
 */

typedef struct Node
{
   int coeff;
   int pow;
   struct Node* link;
}PolyNode;

typedef struct Polynomial
{
    int numberOfTerms;
    PolyNode* Head; //Header contians the first polynomial, so it has to be printed
    PolyNode* Trail;
}Poly;

//UTILITY FUNCTIONS START

void initPoly(Poly **a)
{
    *a = (Poly*)malloc(sizeof(Poly));
    (*a)->Head = NULL;
    (*a)->Trail= NULL;
    (*a)->numberOfTerms=0;
}

void addNode(Poly *a,int pow, int coeff)
{
    PolyNode* n = (PolyNode*) malloc(sizeof(PolyNode));
    if(n!=NULL){
        n->coeff = coeff; n->pow = pow; n->link=NULL;
        if(a->Trail ==NULL)
        {
            a->Head = n;
        }
        else
        {
            a->Trail->link = n;
        }
        a->Trail = n;
    }
    else
```

```
    {
        return;
    }
}

void deleteNode(Poly *a,PolyNode *b)
{
    PolyNode *ptr=a->Head;
    if(ptr==NULL)return;

    while(ptr->link!=b&&ptr!=NULL){ptr=ptr->link;} //Traverse till you find the node
b

    if(ptr==NULL){return;} //If there is no such node then, return

    else
    {
        if(ptr->link->link==NULL)
        {
            free(ptr->link);
            ptr->link=NULL;
        }
        else
        {
            PolyNode *tmp = ptr->link;
            ptr->link = tmp->link;
            free(tmp);
        }
    }
}

void freePoly(Poly **poly)
{
    if(*poly !=NULL)
    {
        PolyNode *i,*tmp;
        i=(*poly)->Head;
        while(i!=NULL)
        {
            tmp=i;
            i=i->link;
            free(tmp);
        }
        free (*poly);
    }
    return;
}

//UTILITY FUNCTIONS END

/* Funtion to print the polynomials*/
void printPoly(Poly* a){
    /* Input: Polynomial stored in the structure Polynomial
     * Ouput: prints the polynomial
     */
```

```
    //int iterCount = a->numberOfTerms;
    //int i;
    PolyNode *ptr=a->Head;
    while(ptr!=a->Trail){
        printf("%d*X^%d + ",ptr->coeff,ptr->pow);
        ptr = ptr->link;
    }
    printf("%d*X^%d",ptr->coeff,ptr->pow);
}

/* Funtion to convert the polynomial into tuple*/
Poly* createPolyFromString(char* s){
    /* Input: String of charecters
     *
     * Output: the Head node of the linked list contating the polynomial
     * */

    Poly* a=NULL;initPoly(&a);
    int i;

    int count = 0;
    int numberStack[2];
    int numberStackTop = -1;

    int number = 0,pow,coeff;
    int negative = 0;

    //parsing the string
    for(i = 0; s[i]!='\0'; i++){
        if(s[i] == '-'){
            negative = 1;
        }

        if(s[i]>='0'&&s[i]<='9'){
            while((s[i]!= 'X'||s[i]!='x'||s[i]!=' '||s[i]!='^') &&
(s[i]>='0'&&s[i]<='9')){
                 // here s[i] will only be numbers
                number = number*10+(s[i]-'0');
                i++;
            }

            if(negative) numberStack[++numberStackTop] = -1*number;
            else numberStack[++numberStackTop] = number;

            i--;
            negative = 0;
            number = 0;
        }

        if(i!=0&&(s[i]=='-'||s[i]=='+'||s[i]=='\0')){//&&s[i-1]!='^'){
            if(numberStackTop==0)
            {
                if(s[i-1]=='X')
                    numberStack[++numberStackTop] = 1;
                else
                    numberStack[++numberStackTop] = 0;
```

```
            }

                count++;
                pow = numberStack[numberStackTop--];
                coeff = numberStack[numberStackTop--];
                addNode(a,pow,coeff);
        }
    }

    if(numberStackTop==0)
    {
        if(s[i-1]=='X')
            numberStack[++numberStackTop] = 1;
        else
            numberStack[++numberStackTop] = 0;
    }
    count++;
    pow = numberStack[numberStackTop--];
    coeff = numberStack[numberStackTop--];
    addNode(a,pow,coeff);

    a->numberOfTerms = count;

    return a;
}

/*Funtion to find the sum of the polynomials*/
Poly* sumOfPoly(Poly* a, Poly* b)
{
    Poly* c = (Poly*)malloc(sizeof(Poly));
    initPoly(&c);

    PolyNode *i=a->Head;
    PolyNode *j=b->Head;

    while(i!=NULL&&j!=NULL)
    {

        if(i->pow==j->pow)
        {
            if(i->coeff+j->coeff!=0)
                addNode(c,i->pow,i->coeff+j->coeff);

            i=i->link;
            j=j->link;
        }
        else if(i->pow>j->pow)
        {
            addNode(c,i->pow,i->coeff);
            i=i->link;
        }
        else if(i->pow<j->pow)
        {
            addNode(c,j->pow,j->coeff);
            j=j->link;
        }
```

```
        c->numberOfTerms++;
    }

    while(i!=NULL)
    {
        addNode(c,i->pow,i->coeff);
        i=i->link;
        c->numberOfTerms++;
    }
    while(j!=NULL)
    {
        addNode(c,j->pow,j->coeff);
        j=j->link;
        c->numberOfTerms++;
    }

    return c;
}

Poly* productOfPolynomials(Poly* a,Poly*b)
{
    Poly *c=NULL;
    Poly *temp=NULL;
    //intiPoly(Temp);

    int k = 0;
    PolyNode *i = a->Head;
    PolyNode *j = b->Head;

    while(k<a->numberOfTerms)
    {
        //i=a->Head;
        j=b->Head;
        if(c==NULL)
        {
            initPoly(&c);
            while(j!=NULL)
            {
                addNode(c,i->pow+j->pow,i->coeff*j->coeff);
                j=j->link;
            }
        }
        else
        {
            initPoly(&temp);
            while(j!=NULL)
            {
                addNode(temp,i->pow+j->pow,i->coeff*j->coeff);
                j=j->link;
            }
            c=sumOfPoly(c,temp);
        }
        i=i->link;
        freePoly(&temp);
        k++;
    }
```

```
    return c;
}

int main(){
    Poly* a;
    Poly* b;
    Poly* c;
    int strLength = 100;
    char* polyString = (char*) malloc(strLength*sizeof(char));

    /*Read the polynomials*/
        fflush(stdin);
        printf("Enter polynomial 1 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 .....
an*X^0 --> ");
        scanf("%[^\n]",polyString);
        scanf("%*c"); //remove the \n charecter from the input stream
        a  = createPolyFromString(polyString);
        free(polyString);

        fflush(stdin);
        fflush(stdout);

        polyString = (char*) malloc(strLength*sizeof(char));

        printf("Enter polynomial 2 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 .....
an*X^0 --> ");
        scanf("%[^\n]",polyString);
        b  = createPolyFromString(polyString);
        free(polyString);
    /*Finish reading Polynomials*/

    printf("\nPolynomial 1 is: ");
    printPoly(a);
    printf("\nPolynomial 2 is: ");
    printPoly(b);

    c = sumOfPoly(a,b); //Find the sum of the polynomials

    printf("\nSum is ");
    printPoly(c);

    c = productOfPolynomials(a,b);
    printf("\nProduct is ");
    printPoly(c);
    printf("\n");

    freePoly(&a);
    freePoly(&b);
    freePoly(&c);
    return 0;
}
```

Sample Input/Output:

```
..ograming/C/CSL201/2020-11-16❯ ./polynomial.o
Enter polynomial 1 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 --> 4X^2+5X+1
Enter polynomial 2 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 --> 5X+4

Polynomial 1 is: 4*X^2 + 5*X^1 + 1*X^0
Polynomial 2 is: 5*X^1 + 4*X^0
Sum is 4*X^2 + 10*X^1 + 5*X^0
Product is 20*X^3 + 41*X^2 + 25*X^1 + 4*X^0
..ograming/C/CSL201/2020-11-16❯
```

```
..ograming/C/CSL201/2020-11-16❯ ./polynomial.o
Enter polynomial 1 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 --> 12X^100+1
Enter polynomial 2 in the form : a0*X^n + a1*X^n-1 + a2*X^n-2 ..... an*X^0 --> 7X

Polynomial 1 is: 12*X^100 + 1*X^0
Polynomial 2 is: 7*X^1
Sum is 12*X^100 + 7*X^1 + 1*X^0
Product is 84*X^101 + 7*X^1
..ograming/C/CSL201/2020-11-16❯
```

Experiment 18
Student Linked List

Date: 16-11-2020

Aim: The details of students are to be stored in a linked list.

Data Structures used: Linked List

Algorithm for Searching

Input: Roll no (RN) of the student to be searched, and the Header node of the linked list
Output: A pointer to the corresponding student, if the number exists in the linked list, NULL in all other cases
Data Structure: Linked List
Steps

```
Step 1: Start
Step 2: ptr = Header→link          //points to the first node in the list
Step 3: if(ptr==NULL)
        Step 1: The linked List is empty
        Step 2: return NULL
Step 4: else
        Step 1: while(ptr!=NULL) do
                Step 1: if(ptr→rollNo == RN) the
                        Step 1: EndWhile
                Step 2: endif
        Step 2: endwhile
        Step 3: if(ptr==NULL) then
                Step 1: return NULL
        Step 4: else
                Step 1: return ptr
        Step 5: endif
Step 5: Stop
```

Algorithm for Sorting

Input: The Header Node of the Linked list to be sorted
Output : The Header node of the sorted Linked list
Data Structure : Linked List

Steps

```
Step 1: Start
Step 2: if(Header→link == NULL) then
    Step 1: print("The List is empty")
Step 3: else
    Step 1: temp = getNode(Node)
    Step 2: ptr = Header→link
    Step 3: while(Header→link!=NULL) do
            Step 1: ptr = Header→link
            Step 2: Header→link = ptr→link
            Step 3: if(Header→link == NULL) then
                    Step 1: Header→link = ptr
                    Step 2: ptr→link = NULL
            Step 4: else
                    Step 1: ptr2 = temp→link
                    Step 2: ptr1 = temp
                    while(ptr2!=NULL and ptr2→rollNo<=ptr→rollno) do
```

17. Step 1: ptr2 = ptr2 $\rightarrow$ link
18. Step 2: ptr1 = ptr1 $\rightarrow$ link
19. Step 4: endwhile
20. Step 5: ptr1 $\rightarrow$ link = ptr
21. Step 6: ptr $\rightarrow$ link = ptr2
22. Step 5: endif
23. Step 4: EndWhile
24. Step 5: Header $\rightarrow$ link = temp $\rightarrow$ link
25. Step 6: returnNode(temp)
26. Step 4: endif
27. Step 5:return Header
28. Step 6: Stop

<u>Program Code</u>

```
/*******************************
 * Linked List Implementation
 * Done By: Rohit Karunakaran
 * *****************************/

#include<stdio.h>
#include<stdlib.h>

typedef struct Linked_List_Node
{
    struct Linked_List_Node *link;
    int rollNo;
    double mark;
    char name[40];

}Student;

void initList(Student* Header)
{
    //Header = (Student*) malloc (sizeof(Student));
    Header->link = NULL;
}

void clearList(Student **List)
{
    Student* ptr = *List;
    Student *eat = ptr;
    ptr = ptr->link;
    if(ptr!=NULL)
    {
        free(eat);
        ptr = ptr->link;
    }
}

void getStudentData(Student* node)
{
    printf("\nEnter the name of the student: ");
    scanf("%[^\n]%*c",node->name);
    printf("Enter the roll no: ");
    scanf("%d",&node->rollNo);
```

```
    printf("Enter the marks: ");
    scanf("%lf",&node->mark);
    printf("\n");
}

//Searching Algorithm
Student* searchFor(Student* Header, int rollNo)
{
    Student* ptr = Header;
    if(Header->link == NULL){
        printf("The List is Empty\n");
        return NULL;
    }
    else
    {
        while(ptr!=NULL)
        {
            if(ptr->rollNo == rollNo)
            {
                return ptr;
            }
            ptr = ptr->link;
        }
        return NULL;
    }
}

//Sorting algorithm
void sortStudentList(Student** Header)
{
    if((*Header)->link==NULL)
    {
        printf("The List is empty]\n");
    }
    else
    {
        Student *temp =(Student*) malloc(sizeof(Student));
        Student *ptr=NULL;
        temp->link=(*Header)->link;
        (*Header)->link = NULL;

        while(temp->link!=NULL)
        {
            ptr = temp->link;
            temp->link = ptr->link;
            if((*Header)->link ==NULL)
            {
                (*Header)->link = ptr;
                ptr->link = NULL;
            }
            else
            {
                Student *ptr2=(*Header)->link;
                Student *ptr1 =(*Header);
                while(ptr2!=NULL && ptr2->rollNo<=ptr->rollNo)
                {
```

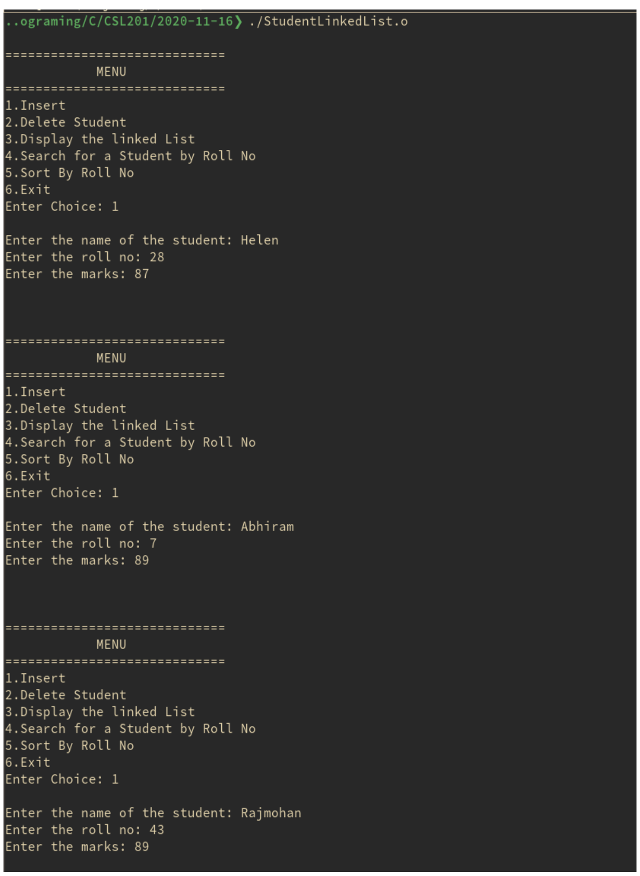
```
                    ptr2=ptr2->link;
                    ptr1=ptr1->link;
                }
                ptr1->link=ptr;
                ptr->link = ptr2;

            }

        }
        free(temp);
    }
}

void dispStudent(Student* ptr)
{
    printf("\nName: %s",ptr->name);
    printf("\nRoll No: %d",ptr->rollNo);
    printf("\nMarks: %lf",ptr->mark);
}

//Insertion Algorithms
void insertStart(Student *Header)
{
    Student *new_node = (Student*) malloc(sizeof(Student));

    if(new_node!=NULL)
    {
        getStudentData(new_node);
        new_node->link = NULL;
        Student* ptr = Header->link;
        Header->link = new_node;
        new_node->link=ptr;
    }
    else
    {
        printf("Insertion Not Possible\n");
        exit(1);
    }
    return ;
}


void deletionAt(Student* Header, int rollNo)
{
    if(Header->link == NULL)
    {
        printf("Deletion not possible. The list is empty\n");
    }
    else
    {
        Student* ptr = Header;
        while(ptr->link!=NULL)
        {
            if(ptr->link->rollNo==rollNo)
                break;
            ptr=ptr->link;
```

```
        }
        if(ptr->link!=NULL)
        {
            Student* red = ptr->link;
            ptr->link = ptr->link->link;
            printf("The Student to be deleted is :\n");
            dispStudent(red);
            free(red);
        }
        else
        {
            printf("The Given RollNo is not found \n");
        }
    }
}



void displayList(Student* Header)
{
    Student* ptr = Header->link;
    if(ptr!=NULL)
    {

        while(ptr!=NULL)
        {
                  printf("\n");
            dispStudent(ptr);
            printf("\n");
            ptr=ptr->link;
        }
        printf("\n");
    }
    else
    {
        printf("The Linked list is empty\n");
    }
}

int menu(Student* Header)
{
    int RUN = 1;
    while(RUN)
    {
        printf("\n");
        printf("=============================\n");
        printf("            MENU            \n");
        printf("=============================\n");
        printf("1.Insert\n");
        printf("2.Delete Student\n");
        printf("3.Display the linked List\n");
        printf("4.Search for a Student by Roll No\n");
        printf("5.Sort By Roll No\n");
        printf("6.Exit\n");
        printf("Enter Choice: ");
        int choice;
```

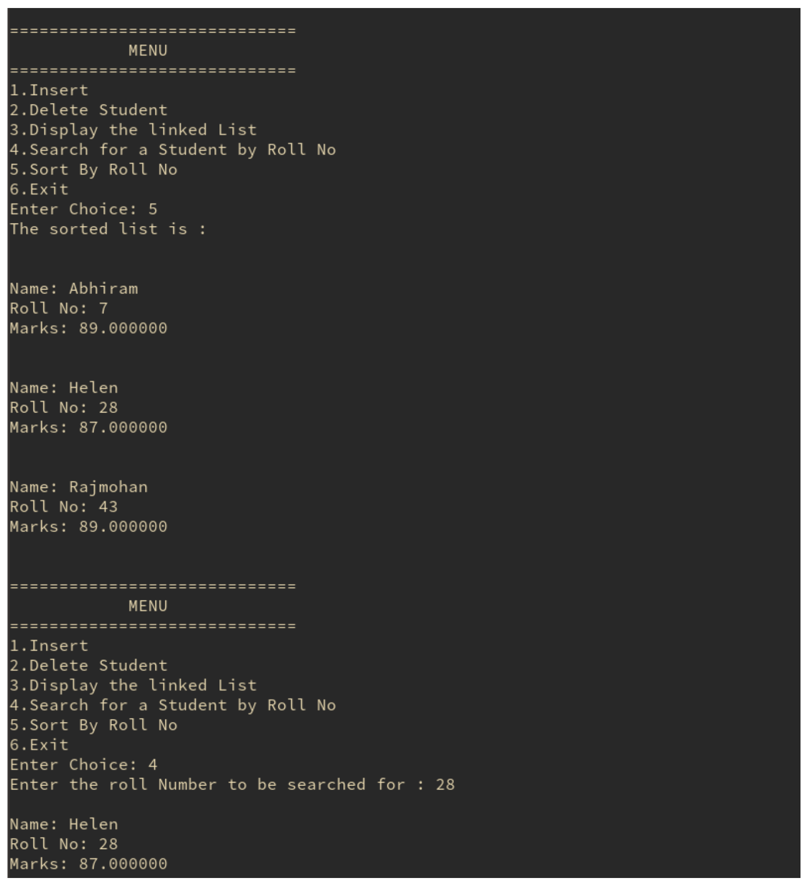
```
int pos;
scanf("%d%*c",&choice);

switch(choice)
{
    case 1:
            insertStart(Header);
            printf("\n");
            break;

    case 2: printf("Enter the roll no of the student to be deleted : ");
            scanf("%d%*c",&pos);
            deletionAt(Header,pos);
            printf("\n");
            break;

    case 3: printf("\nThe Student List is : ");
                      displayList(Header);
            break;

    case 4: printf("Enter the roll Number to be searched for : ");
            scanf("%d%*c",&pos);
            Student* res = searchFor(Header,pos);
            if(res == NULL)
            {
                printf("The given roll number is invalid !!!\n");
            }
            else
            {
                dispStudent(res);
            }
            break;

    case 5: sortStudentList(&Header);
                      printf("The sorted list is :\n");
                      displayList(Header);
            break;

    case 6: RUN=0;
            break;

    default: printf("Enter a valid choice\n");
```

```
                    printf("\n");
                    break;

            }
        }
        printf("Exiting........\n");
        clearList(&Header);
        return RUN;
}


int main(){
    Student *Header = (Student*)malloc(sizeof(Student));
    initList(Header);
    return menu(Header);
}
```

Sample input output



```
..ograming/C/CSL201/2020-11-16❯ ./StudentLinkedList.o

==============================
           MENU
==============================
1.Insert
2.Delete Student
3.Display the linked List
4.Search for a Student by Roll No
5.Sort By Roll No
6.Exit
Enter Choice: 1

Enter the name of the student: Helen
Enter the roll no: 28
Enter the marks: 87



==============================
           MENU
==============================
1.Insert
2.Delete Student
3.Display the linked List
4.Search for a Student by Roll No
5.Sort By Roll No
6.Exit
Enter Choice: 1

Enter the name of the student: Abhiram
Enter the roll no: 7
Enter the marks: 89



==============================
           MENU
==============================
1.Insert
2.Delete Student
3.Display the linked List
4.Search for a Student by Roll No
5.Sort By Roll No
6.Exit
Enter Choice: 1

Enter the name of the student: Rajmohan
Enter the roll no: 43
Enter the marks: 89
```



```
==============================
           MENU
==============================
1.Insert
2.Delete Student
3.Display the linked List
4.Search for a Student by Roll No
5.Sort By Roll No
6.Exit
Enter Choice: 5
The sorted list is :


Name: Abhiram
Roll No: 7
Marks: 89.000000


Name: Helen
Roll No: 28
Marks: 87.000000


Name: Rajmohan
Roll No: 43
Marks: 89.000000


==============================
           MENU
==============================
1.Insert
2.Delete Student
3.Display the linked List
4.Search for a Student by Roll No
5.Sort By Roll No
6.Exit
Enter Choice: 4
Enter the roll Number to be searched for : 28

Name: Helen
Roll No: 28
Marks: 87.000000
```

```
=============================
            MENU
=============================
1.Insert
2.Delete Student
3.Display the linked List
4.Search for a Student by Roll No
5.Sort By Roll No
6.Exit
Enter Choice: 2
Enter the roll no of the student to be deleted : 7
The Student to be deleted is :

Name: Abhiram
Roll No: 7
Marks: 89.000000

=============================
            MENU
=============================
1.Insert
2.Delete Student
3.Display the linked List
4.Search for a Student by Roll No
5.Sort By Roll No
6.Exit
Enter Choice: 3

The Student List is :

Name: Helen
Roll No: 28
Marks: 87.000000


Name: Rajmohan
Roll No: 43
Marks: 89.000000


=============================
            MENU
=============================
1.Insert
2.Delete Student
3.Display the linked List
4.Search for a Student by Roll No
5.Sort By Roll No
6.Exit
Enter Choice: 6
Exiting........
```

Experiment 19
Doubly Linked List

Date : 19-11-2020

Aim: To implement a Doubly Linked List and check whether the given string is palindrome

Data Structure used : Linked List

Algorithms

1. Algorithm for checking palindrome

Input: A Doubly Linked List with the Head pointing to the first element of the string and the Tail pointing to the last
Output: 1 if the string is palindrome 0 if otherwise
Data Structure: Doubly Linked List

Steps:

```
Step 1: if(Head==NULL)
        Step 1: print(The list is empty)
        Step 2: return 0
Step 2: else
        Step 1: i = Header→rlink
        Step 2: j = Tail → llink
        Step 3: while(i!=Head and j!=Tail) do
                Step 1: if(i→data!=j→data ) then
                        Step 1: endWhile
                Step 2: endif
        Step 4: EndWhile
        Step 5: if(i==Head and j==Tail) do
                Step 1: return 1
        Step 6: else
                Step 1: return 0
        Step 7: endif
Step 3: endif
Step 4: Sop
```

Result: the Program compiled successfully and the desired output was obtained.

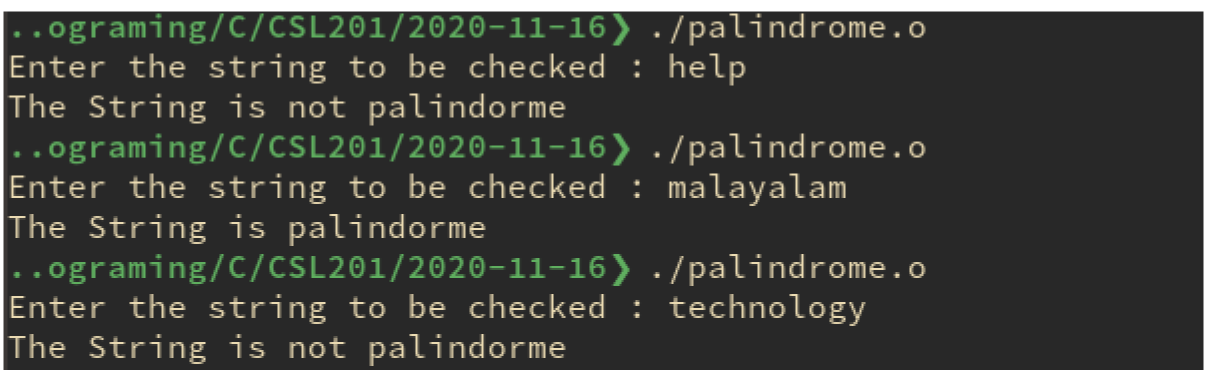
Program code:

```
/***************************************
 * Program to check whether the given
 * string is palindrome using doubly linked list
 * Done By: Rohit Karunaran
 * **************************************/
#include<stdio.h>
#include<stdlib.h>

typedef struct char_doubly_linked_list
{
    struct char_doubly_linked_list *next;
    struct char_doubly_linked_list *prev;
    char data;
```

```c
} ddchar;

void initString(ddchar **Header)
{
    *Header = (ddchar*)malloc(sizeof(ddchar));
    (*Header)->next = NULL;
    (*Header)->prev = NULL;
}

void insert(ddchar *Header,char ch)
{
    ddchar *newNode = (ddchar*)malloc(sizeof(ddchar));

    if(newNode!=NULL)
    {
        ddchar *Tail = Header;
        newNode->data = ch;
        if(Header->next == NULL) //That is the string is empty
        {
            Tail = NULL;
            Header->next = newNode;
            newNode->prev = Header;
            newNode->next=NULL;
        }
        else
        {
            while(Tail->next!=NULL) Tail = Tail->next;
            Tail->next = newNode;
            newNode->prev = Tail;
            newNode->next=NULL;
        }

    }
}
 void stringToList(ddchar *Header,char *s)
{
    for(int i=0;s[i]!='\0';i++)
        insert(Header,s[i]);
}

int checkPalindrome(ddchar *Header)
{
    ddchar *i,*j;
    if(Header->next!=NULL)
    {
        i=Header->next;
        j=Header;
        while(j->next!=NULL)j=j->next; //j becomes the tail pointer

        while(i!=NULL&&j!=Header)
        {
            if(i->data!=j->data)
                break;
            i=i->next;
            j=j->prev;
        }
```

```
        if(i==NULL && j==Header)
        {
            return 1;
        }
        return 0;
    }
    else{
        return 0;
    }
}

int main()
{
    ddchar *str = (ddchar*) malloc(sizeof(ddchar));
    initString(&str);
    char input[50];
    printf("Enter the string to be checked : ");
    scanf("%[^\n]%*c",input);
    stringToList(str,input);
    if(checkPalindrome(str))
    {
        printf("The String is palindorme");
    }
    else
    {
        printf("The String is not palindorme");
    }
    return 0;
}
```

Sample Input/Output



```
..ograming/C/CSL201/2020-11-16❯ ./palindrome.o
Enter the string to be checked : help
The String is not palindorme
..ograming/C/CSL201/2020-11-16❯ ./palindrome.o
Enter the string to be checked : malayalam
The String is palindorme
..ograming/C/CSL201/2020-11-16❯ ./palindrome.o
Enter the string to be checked : technology
The String is not palindorme
```

<u>Experiment 20</u>
<u>Binary Tree</u>

Date: 31-12-2020

Aim: Implement a Binary Tree

Data Structures used: Linked List, Binary Tree

Algorithm for Insertion

Input: The root node (root) and the key after which the element is to be inserted
Output : The binary tree with the node inserted
Data Structure : Binary Tree

Steps

```
1.  Step 1: Start
2.  Step 2: ptr = Srearch(root,key)
3.  Step 3: if(ptr == NULL) then
4.          Step 1: print("No element found")
5.          Step 2: exit
6.  Step 4: endif
7.  Step 5: If(ptr→lc ==NULL or ptr→rc==NULL) then
8.          Step 1: read option to insert the node left or right
9.          Step 2: if(option == l) then
10.                 Step1: if(ptr→lc == NULL)
11.                         Step1: new=GetNode(node)
12.                         Step 2: new→data = item
13.                         Step 3: new→lc=new→rc =NULL
14.                         Step 4: ptr→lc = new
15.                 Step 2: else
16.                         Step 1: print("Insertion not possible")
17.                         Step 2: exit
18.                 Step 3: endif
19.         Step 3: else if(option == r)then
20.                 Step 1: if(ptr→rc= NULL) then
21.                         Step 1: new = getNode(node)
22.                         Step 2: new→data = item
23.                         Step 3: new→lc=new→rc= NULL
24.                         Step 4: ptr→rc = new
25.                 Step 2 : else
26.                         Step 1: print("Insertion not posiible")
27.                             Step 2: exit
28.                 Step 3:endif
29.         Step 3: endif
30. Step 6: endif
31. Step 7: Stop
```

Algorithm for Deleting a node

Input: Root node of the binary tree, the element to be deleted
Output: Binary tree with the element deleted
Data Structure used: Binary tree

Steps

Step 1: Start
Step 2: getParent(root,elem)

Step 3: if(parent → rc == elem) then
Step 1: ptr = parent → rc
Step 4: else
Step 1: ptr = parent →lc
Step 5: endif
Step 6: if(ptr→rc!=NULL || ptr→lc!=NULL) then
Step 1: print(“ptr is a leaf node it cant be deleted”)
Step 7: else if(ptr==parent →rc) then
Step 1:parent →rc=NULL
Step 8:else
Step 1: parent → lc =NULL
Step 9: endif
Step 10: returnNode(ptr)

Algorithm for Inorder Traversal

Input: Root node of the binary tree
Output : All the nodes of the binary tree visited in an inorder fashion
Data Structure used: Binary trees

Steps

1. Step 1: Start
2. Step 2: if(root!=NULL) then
3. Step 1: inorder_traversal(root→lc)
4. Step 2: visit(root)
5. Step 3: inorder_traversal(root→ rc)
6. Step 3: else
7. Step 1: return
8. Step 4: endif
9. Step 5: Stop

Algorithm for Postorder Traversal

Input: Root node of the binary tree
Output : All the nodes of the binary tree visited in an postorder fashion
Data Structure used: Binary trees

Steps

10. Step 1: Start
11. Step 2: if(root!=NULL) then
12. Step 1: postorder_traversal(root→lc)
13. Step 2: postorder_traversal(root→ rc)
14. Step 3: visit(root)
15. Step 3: else
16. Step 1: return
17. Step 4: endif
18. Step 5: Stop

Algorithm for Preorder Traversal

Input: Root node of the binary tree
Output : All the nodes of the binary tree visited in an preorder fashion
Data Structure used: Binary trees

Steps

19. Step 1: Start
20. Step 2: if(root!=NULL) then
21. Step 1: visit(root)
22. Step 2: preorder_traversal(root→lc)
23. Step 3: preorder_traversal(root→ rc)
24.
25. Step 3: else
26. Step 1: return
27. Step 4: endif
28. Step 5: Stop

Algorithm for Searching

Input: Root node (root) and the value to be searched(key)

Output: A pointer to the corresponding node, if the key is present in the binary tree else null

Data Structure: Linked List, Binary Tree

Steps

1. Step 1: Start
2. Step 2: ptr=root
3. Step 3: if(ptr→data!=key) then
4. Step 1: if(ptr→lc!=NULL) then
5. Step 1: Search(root→lc,key)
6. Step2: endif
7. Step3: if(ptr→rc!=NULL) then
8. Step 1: Search(root→rc,key)
9. Step4: endif
10. Step 5: return (NULL)
11. Step 4: else
12. Step 1 : return ptr //base case
13. Step 5: endif

Program Code

```
/******************************
 * Binary tree
 * Done By: Rohit Karunakaran
 * *****************************/
#include<stdio.h>
#include<stdlib.h>

typedef struct binary_tree_node{
    struct binary_tree_node* lc;
    struct binary_tree_node* rc;
    int value;
}node;

/*
node* init_tree(){
    root_node = (node*) malloc(sizeof(node));
}
*/

node* search_node(node* root, int value){
    node* ptr=NULL;
    if(root->value != value){
        if(root->lc==NULL && root->rc==NULL){
```

```
                return NULL;
            }
            else{
                if(root->lc!=NULL){
                    ptr = search_node(root->lc, value);
                    if(ptr!=NULL){
                        return ptr;
                    }
                }
                if(root->rc!=NULL){
                    ptr = search_node(root->rc,value) ;
                    if(ptr !=NULL){
                        return ptr;
                    }
                }
                return ptr;
            }
        }
        else{
            return root;
        }
}

node* search_parent(node* root, int value){
    node* ptr = NULL;
    if(root!=NULL){
        if(root->lc !=NULL && root->rc!=NULL){
            if(root->lc ->value == value||root->rc->value==value){
                return root;
            }else{
                ptr = search_parent(root->lc, value);
                if(ptr == NULL){
                    ptr = search_parent(root->rc, value);
                }
                return ptr;
            }
        }
        else if(root -> lc ==NULL && root ->rc ==NULL){
            return NULL;
        }
        else{
            if(root->lc == NULL){
                if(root->rc->value==value){
                    return root;
                }
                else{
                    ptr = search_parent(root->rc,value);
                    return ptr;
                }
            }
            else{
                if(root->lc->value==value){
                    return root;
                }
                else{
                    ptr = search_parent(root->lc,value);
```

```
                return ptr;
            }

        }
    }
}
else{
    return NULL;
}
}

void insert_node(node* root,int value){
    node* ptr = search_node(root,value);
    char c;
    if(ptr!=NULL){
        fflush(stdin);
        printf("Insert Node as Left child or as a right child: ");
        scanf("\n%c",&c);
        if(c == 'l'){
            if(ptr->lc == NULL){
                node* tmp = (node*)malloc(sizeof(node));
                printf("Enter the value to be inserted: ");
                scanf("%d",&(tmp->value));
                tmp->rc = NULL;
                tmp->lc = NULL;
                ptr->lc = tmp;
            }
            else{
                printf("Insertion at the left node of %d is not possible\n",ptr-
>value);
            }
        }

        else if(c =='r'){
            if(ptr->rc == NULL){
                node* tmp = (node*)malloc(sizeof(node));
                printf("Enter the value to be inserted: ");
                scanf("%d",&(tmp->value));
                tmp->rc = NULL;
                tmp->lc = NULL;
                ptr->rc = tmp;
            }
            else{
                printf("Insertion at the right node of %d is not possible\n",ptr-
>value);
            }
        }
        else{
            printf("Proper option was not chosen\n");
        }
    }
    else{
        printf("Value %d not found!!!!\nInsertion not possible\n",value);
    }
}
```

```
void inorder_traversal(node* root){
    if(root!=NULL){
        inorder_traversal(root->lc);
        printf("%d ",root->value);
        inorder_traversal(root->rc);

    }
    else{
        return;
    }

}
void postorder_traversal(node* root){
    if(root!=NULL){
        printf("%d ",root->value);
        postorder_traversal(root->lc);
        postorder_traversal(root->rc);
    }
    else{
        return;
    }
}
void preorder_traversal(node* root){
    if(root!=NULL){
        preorder_traversal(root->lc);
        preorder_traversal(root->rc);
        printf("%d ",root->value);
    }
    else{
        return;
    }
}
void delete_node(node** root, int value)
{
    node* parent = search_parent(*root, value);
    if(parent == NULL){
        if((*root)->value == value&&(*root)->rc==NULL&&(*root)->lc==NULL){
            free(*root);
            *root = NULL;

        }
        else if((*root)->value == value){
            printf("Deletion not possible\n");
        }
        else{
            printf("The value %d not found in the tree\n\n",value);
        }
    }
    else{
        if(parent->rc !=NULL&&parent->rc->value==value){
            if(parent->rc->rc==NULL && parent->rc->lc==NULL){
                free(parent->rc);
                parent->rc =NULL;
            }
            else{
                printf("Deletion not possible\n");
```

```c
                }
            }
            else{
                if(parent->lc->lc==NULL && parent->lc->rc==NULL){
                    free(parent->lc);
                    parent->lc =NULL;
                }
                else{
                    printf("Deletion not possible\n");
                }
            }
        }
    }

int menu(node* root){
    printf("Binary Tree implementation\n");
    int RUN=1;
    int choice;
    int elem;
    while(RUN){
        printf("\nMenu\n");
        printf("1.Insert\n");
        printf("2.Inorder traversal\n");
        printf("3.Preorder traversal\n");
        printf("4.Postorder traversal\n");
        printf("5.Delete Node\n");
        printf("6. Exit\n");
        printf("Enter Choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: if(root==NULL){
                        root = (node*)malloc(sizeof(node));
                        printf("Enter the value to be inserted: ");
                        scanf("%d",&elem);
                        root->value = elem;root->lc = NULL;root->rc = NULL;
                    }
                    else{
                        printf("Enter the value to be searched for : ");
                        scanf("%d",&elem);
                        insert_node(root,elem);
                    }
                    break;
            case 2: if(root!=NULL){
                        printf("\nInorder Traversal : ");
                        inorder_traversal(root);
                    }
                    else
                        printf("The tree is Empty!!!!\n");
                    break;
            case 3: if(root!=NULL){
                        printf("\nProerder Traversal : ");
                        preorder_traversal(root);
                    }
                    else
                        printf("The tree is Empty!!!!\n");
                    break;
```

```
            case 4: if(root!=NULL){
                        printf("\nPostorder Traversal : ");
                        postorder_traversal(root);
                    }
                    else
                        printf("The tree is Empty!!!!\n");
                    break;
            case 5: printf("Enter the value to be deleted: ");
                    scanf("%d",&elem);
                    delete_node(&root,elem);
                    break;
            case 6: RUN=0;
                    break;
        }
    }
    return RUN;
}

int main(){
    node* root = NULL;
    return menu(root);
}
```

Sample Input and Output

```
..ograming/C/CSL201/2020-12-31❯ ./binaryTree.o
Binary Tree implementation

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 1
Enter the value to be inserted: 12

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 1
Enter the value to be searched for : 12
Insert Node as Left child or as a right child: r
Enter the value to be inserted: 15

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 1
Enter the value to be searched for : 12
Insert Node as Left child or as a right child: l
Enter the value to be inserted: 23

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 1
Enter the value to be searched for : 23
Insert Node as Left child or as a right child: r
Enter the value to be inserted: 63

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 2
```

```
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 2

Inorder Traversal : 23 63 12 15
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 3

Proerder Traversal : 63 23 15 12
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 4

Postorder Traversal : 12 23 63 15
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 5
Enter the value to be deleted: 12
Deletion not possible

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 5
Enter the value to be deleted: 63
```

```
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 2

Inorder Traversal : 23 12 15
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 3

Proerder Traversal : 23 15 12
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6. Exit
Enter Choice: 6
```

<u>Experiment 21</u>
<u>Binary Search Tree</u>

Date: 31-12-2020

Aim: Implement a Binary Search Tree

Data Structures used: Linked List, Binary Tree

Algorithm for Insertion

Input: The root node (root) and the key, element to be inserted
Output : The binary search tree with the node inserted
Data Structure : Binary Search Tree

Steps

1. Step 1: Start
2. Step 2: ptr = root
3. Step 3: while(ptr!=NULL and flag==true) do
4. Step 1: case: item<=ptr → data
5. Step 1: ptr1 = ptr
6. Step 2: ptr=ptr→ lc
7. Step 2: case: item>ptr → data
8. Step 1: ptr1=ptr
9. Step 2: ptr = ptr→ rc
10. Step 3: endCase
11. Step 4: endWhile
12. Step 5: if(ptr==NULL) then
13. Step 1: new = getNode(node)
14. Step 2: new → data = item
15. Step 3: new→ rc = new→lc = NULL
16. Step 4: if(ptr→ dara <= item) then
17. Step 1: ptr1→ rc = new
18. Step 5: else
19. Step 1: ptr1→lc = new
20. Step 6: endIf
21. Step 6: endif
22. Step 7: Stop

Algorithm for Deleting a node

Input: Root node of the binary search tree, the element to be deleted
Output: Binary tree with the element deleted
Data Structure used: Binary search tree

Steps

1. Step 1: Start
2. Step 2: ptr = root
3. Step 3: flage = false
4. Step 4: while(ptr!=NULL) then
5. Step 1: case: item < ptr→ data
6. Step 1: parent = ptr
7. Step 2: ptr = ptr→lc
8. Step 2: case item > ptr→data
9. Step 1: parent = ptr

10. Step 2: ptr = ptr→rc
11. Step 3: case item=ptr→data
12. Step 1: flage = true
13. Step 4: endcase
14. Step 5: endWhile
15. Step 6: if(flag = false) then
16. Step 1: printf(“There is no item in the binary tree”)
17. Step 2: exit
18. Step 7: endIf
19. Step 8: If(ptr→lc==NULL and ptr→ rc ==NULL) then //case 1
20. Step 1: if(parent → lc == ptr) then
21. Step 1: parent→lc =NULL
22. Step 2: else
23. Step 1: parent →rc =NULL
24. Step 3: endIf
25. Step 4: returnNode(ptr)
26. Step 9: else if(ptr→lc !=NULL and ptr-.rc !=NULL) then //case 3
27. Step 1: ptr1 = ptr →rc
28. Step 2: while(ptr1→lc!=NULL) do
29. Step 1: ptr1= ptr1 → lc
30. Step 3: endWhile
31. Step 4: item = ptr1→data
32. Step 5: delete_node(ptr1)
33. Step 6: ptr → data = item
34. Step 10: else //case 2
35. Step 1: if(parent→lc == ptr) then
36. Step 1: if (ptr→lc ==NULL) then
37. Step 1: parent →lc = ptr→rc
38. Step 2: else
39. Step 1: parent →lc = ptr → lc
40. Step 3: endIf
41. Step 2: else
42. Step 1: if(ptr → lc ==NULL) then
43. Step 1: parent →rc = ptr→rc
44. Step 2: else
45. Step 1: parent → rc = ptr → lc
46. Step 3: endif
47. Step 3: EndIf
48. Step 11: endif
49. Step 12: Stop

Algorithm for Inorder Traversal

Input: Root node of the binary tree
Output : All the nodes of the binary tree visited in an inorder fashion
Data Structure used: Binary trees

Steps

1. Step 1: Start
2. Step 2: if(root!=NULL) then
3. Step 1: inorder_traversal(root→lc)
4. Step 2: visit(root)
5. Step 3: inorder_traversal(root→ rc)
6. Step 3: else
7. Step 1: return
8. Step 4: endif
9. Step 5: Stop

Algorithm for Postorder Traversal

Input: Root node of the binary tree
Output : All the nodes of the binary tree visited in an postorder fashion
Data Structure used: Binary trees

Steps

10. Step 1: Start
11. Step 2: if(root!=NULL) then
12. Step 1: postorder_traversal(root→lc)
13. Step 2: postorder_traversal(root→ rc)
14. Step 3: visit(root)
15. Step 3: else
16. Step 1: return
17. Step 4: endif
18. Step 5: Stop

Algorithm for Preorder Traversal

Input: Root node of the binary tree
Output : All the nodes of the binary tree visited in an preorder fashion
Data Structure used: Binary trees

Steps

19. Step 1: Start
20. Step 2: if(root!=NULL) then
21. Step 1: visit(root)
22. Step 2: preorder_traversal(root→lc)
23. Step 3: preorder_traversal(root→ rc)
24.
25. Step 3: else
26. Step 1: return
27. Step 4: endif
28. Step 5: Stop

Program Code

```
/*********************
 * Binary Search Tree
 * Done By Rohit Karunakaran
 * ********************/


#include<stdio.h>
#include<stdlib.h>

typedef struct binary_search_tree_node{
    struct binary_search_tree_node* lc;
    struct binary_search_tree_node* rc;
    int value;
}node;

node* search_node(node* root, int value){
    if(root!=NULL){
        if(root->value!=value){
            if(root->value>value){
```

```
                return search_node(root->lc,value);
            }
            else{
                return search_node(root->rc,value);
            }
        }
        else{
            return root;
        }
    }
    else{
        return NULL;
    }
}


void insert_node(node** root,int value){
    int flag=1;
    node* ptr=*root;
    if(ptr!=NULL){
        while(ptr!=NULL&&flag){
            if(ptr->value<value){
                if(ptr->rc==NULL){
                    ptr->rc = (node*)malloc(sizeof(node));
                    ptr->rc->lc = ptr->rc->rc =NULL;
                    ptr->rc->value = value;
                    flag=0;
                }
                else{
                    ptr= ptr->rc;
                }
            }
            else{
                if(ptr->lc==NULL){
                    ptr->lc = (node*)malloc(sizeof(node));
                    ptr->lc->lc = ptr->lc->rc =NULL;
                    ptr->lc->value = value;
                    flag=0;
                }
                else{
                    ptr = ptr->lc;
                }
            }
        }
    }
    else{
    //Root is empty
        *root = (node*)malloc(sizeof(node));
        (*root) ->lc = (*root)->rc = NULL;
        (*root)->value = value;
    }
}

void delete_node(node** root, int value,node* par){
    node* ptr = *root;
    node* parent =par;
```

```
int flag = 1;
if(ptr!=NULL){
    while(ptr!=NULL&&flag){
        if(ptr->value<value){
            parent = ptr;
            ptr = ptr->rc;
        }
        else if(ptr->value>value){
            parent = ptr;
            ptr = ptr->lc;
        }
        else{
            flag = 0;
        }
    }
    if(flag == 1){
        printf("Item not found\n");
        return;
    }
    if(ptr ->lc ==NULL && ptr->rc==NULL){
        if(parent!=NULL){
            if(parent -> rc ==ptr){
                parent ->rc =NULL;
            }
            else {
                parent ->lc =NULL;
            }
        }
        else{
            *root = NULL;
        }
        free(ptr);

    }
    else if(ptr->lc!=NULL && ptr->rc!=NULL){
        node* ptr1=ptr->rc;
        while(ptr1->lc!=NULL) ptr1=ptr1->lc; //Find the successor node
        int item = ptr1->value;
        delete_node(&ptr1,item,ptr);
        ptr->value = item;
    }
    else{
        if(parent!=NULL){
            if(parent ->rc ==ptr){
               if(ptr->rc!=NULL){
                    parent ->rc = ptr->rc;
               }
               else{
                    parent->rc = ptr->lc;
               }
            }
            else{
               if(ptr->rc!=NULL){
                    parent ->lc = ptr->rc;
               }
               else{
```

```
                        parent->lc = ptr->lc;
                    }

                }
            }
            else{
                //If the parent is null then the node is root and has one child
                if(ptr->rc!=NULL){
                    *root = ptr->rc;
                }
                else{
                    *root = ptr->lc;
                }
            }
            free(ptr);
        }
    }
    else{
        printf("There is no item in the binary tree\n");
    }
}

void inorder_traversal(node* root){
    if(root!=NULL){
        inorder_traversal(root->lc);
        printf("%d ",root->value);
        inorder_traversal(root->rc);

    }
    else{
        return;
    }

}

void postorder_traversal(node* root){
    if(root!=NULL){
        postorder_traversal(root->lc);
        postorder_traversal(root->rc);
        printf("%d ",root->value);
    }
    else{
        return;
    }
}

void preorder_traversal(node* root){
    if(root!=NULL){
        printf("%d ",root->value);
        preorder_traversal(root->lc);
        preorder_traversal(root->rc);
    }
    else{
        return;
    }
}
```

```
void leaf_nodes(node* root,int* count){
    if(root!=NULL){
        leaf_nodes(root->lc,count);
        if(root->lc==NULL&&root->rc==NULL)(*count)++;
        leaf_nodes(root->rc,count);

    }
    else{
        return;
    }

}


int menu(node* root){
    printf("Binary Tree implementation\n");
    int RUN=1;
    int choice;
    int elem;
    while(RUN){
        printf("\nMenu\n");
        printf("1.Insert\n");
        printf("2.Inorder traversal\n");
        printf("3.Preorder traversal\n");
        printf("4.Postorder traversal\n");
        printf("5.Delete Node\n");
        printf("6.Number of leaf nodes\n");
        printf("7. Exit\n");
        printf("Enter Choice: ");
        scanf("%d",&choice);
        switch(choice){
            case 1: printf("Enter the value to be inserted : ");
                    scanf("%d",&elem);
                    insert_node(&root,elem);
                    break;
            case 2: if(root!=NULL){
                        printf("Inorder Traversal: ");
                        inorder_traversal(root);
                    }
                    else
                        printf("The tree is Empty!!!!\n");
                    break;
            case 3: if(root!=NULL){
                        printf("Preorder Traversal: ");
                        preorder_traversal(root);
                    }
                    else
                        printf("The tree is Empty!!!!\n");
                    break;
            case 4: if(root!=NULL){
                        printf("Postorder Traversal: ");
                        postorder_traversal(root);
                    }
                    else
                        printf("The tree is Empty!!!!\n");
```

```
                break;
            case 5: printf("Enter the value to be deleted: ");
                scanf("%d",&elem);
                delete_node(&root,elem,NULL);
                break;

            case 6: if(root!=NULL){
                    elem = 0;
                    leaf_nodes(root,&elem);
                    printf("Number of leafnodes = %d\n",elem)
                }
                else{
                    printf("The tree is empty there is no leaf nodes\n");
                }
            case 7: RUN=0;
                break;
            default:printf("Wrong value entered try again\n\n");
                break;
        }

    }

    return RUN;
}

int main(){
    node* root = NULL;
    return menu(root);
}
```

Result: The program compiled successfully and required output was obtained

Sample input and output

```
..ograming/C/CSL201/2020-12-31❯ ./binarySearchTree.o
Binary Tree implementation

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 1
Enter the value to be inserted : 12

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 1
Enter the value to be inserted : 11

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 1
Enter the value to be inserted : 14

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 1
Enter the value to be inserted : 35
```

```
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 1
Enter the value to be inserted : 24

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 2
Inorder Traversal: 11 12 14 24 35
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 3
Preorder Traversal: 12 11 14 35 24
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 4
Postorder Traversal: 11 24 35 14 12
```

```
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 5
Enter the value to be deleted: 12

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 5
Enter the value to be deleted: 11

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 2
Inorder Traversal: 14 24 35
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 5
Enter the value to be deleted: 24
```

```
Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 6
Number of leafnodes = 1

Menu
1.Insert
2.Inorder traversal
3.Preorder traversal
4.Postorder traversal
5.Delete Node
6.Number of leaf nodes
7. Exit
Enter Choice: 7
..ograming/C/CSL201/2020-12-31❯
```

Experiment 22
BST Sort

Date: 31-12-2020

Aim: Sort a array of numbers using binary search tree

Data Structures used: Linked List, Binary Tree, Array

Algorithm for Insertion

Input: The root node (root) and the key, element to be inserted
Output : The binary search tree with the node inserted
Data Structure : Binary Search Tree

Steps

1. Step 1: Start
2. Step 2: ptr = root
3. Step 3: while(ptr!=NULL and flag==true) do
4. Step 1: case: item<=ptr → data
5. Step 1: ptr1 = ptr
6. Step 2: ptr=ptr→ lc
7. Step 2: case: item>ptr → data
8. Step 1: ptr1=ptr
9. Step 2: ptr = ptr→ rc
10. Step 3: endCase
11. Step 4: endWhile
12. Step 5: if(ptr==NULL) then
13. Step 1: new = getNode(node)
14. Step 2: new → data = item
15. Step 3: new→ rc = new→lc = NULL
16. Step 4: if(ptr→ dara <= item) then
17. Step 1: ptr1→ rc = new
18. Step 5: else
19. Step 1: ptr1→lc = new
20. Step 6: endIf
21. Step 6: endif
22. Step 7: Stop

Algorithm for Sorting

Input: Root node of the binary tree containing the elements to be sorted and a array in which elements are to be inserted in sorted order
Output : All the elements sorted
Data Structure used: Binary Search trees, array

Steps

1. Step 1: Start // i is initialized to zero
2. Step 2: if(root!=NULL) then
3. Step 1: bst_sort(root→lc,arr)
4. Step 2: arr[i] = root→ value
5. Step 3: i++
6. Step 4: bst_sort(root→ rc,arr)
7. Step 3: else

8. Step 1: return
9. Step 4: endif
10. Step 5: Stop

Program Code

```
/*********************
 *  Sorting using binary search tree
 * Done By Rohit Karunakaran
 * ********************/


#include<stdio.h>
#include<stdlib.h>

typedef struct binary_search_tree_node{
    struct binary_search_tree_node* lc;
    struct binary_search_tree_node* rc;
    int value;
}node;

void insert_node(node** root,int value){
    int flag=1;
    node* ptr=*root;
    if(ptr!=NULL){
        while(ptr!=NULL&&flag){
            if(ptr->value<value){
                if(ptr->rc==NULL){
                    ptr->rc = (node*)malloc(sizeof(node));
                    ptr->rc->lc = ptr->rc->rc =NULL;
                    ptr->rc->value = value;
                    flag=0;
                }
                else{
                    ptr= ptr->rc;
                }
            }
            else{
                if(ptr->lc==NULL){
                    ptr->lc = (node*)malloc(sizeof(node));
                    ptr->lc->lc = ptr->lc->rc =NULL;
                    ptr->lc->value = value;
                    flag=0;
                }
                else{
                    ptr = ptr->lc;
                }
            }
        }
    }
    else{
    //Root is empty
        *root = (node*)malloc(sizeof(node));
        (*root) ->lc = (*root)->rc = NULL;
        (*root)->value = value;
```

```
    }
}

int index =0;
void bstSort(node* root,int arr[]){
    if(root!=NULL){
        bstSort(root->lc,arr);
        arr[index] = root->value; index++;
        bstSort(root->rc,arr);

    }
    else{
        return;
    }

}


int main(){
    node* root = NULL;
    int n;
    printf("Enter the number of elements to be sorted :");
    scanf("%d",&n);
    int arr[n];
    printf("Enter the elements in the array : ");

    for(int i=0;i<n;i++){
        int elem;
        scanf("%d",&elem);
        insert_node(&root,elem);
    }

    bstSort(root,arr);
    printf("The Sorted array of elemets are: ");
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
    return 0;
}
```

Result: The program compiled successfully and required output was obtained

Sample input and output

```
..ograming/C/CSL201/2020-12-31❯ ./bstSort.o
Enter the number of elements to be sorted :7
Enter the elements in the array : 1 0 4 9 23 14 1
The Sorted array of elemets are: 0 1 1 4 9 14 23
..ograming/C/CSL201/2020-12-31❯
```

```
..ograming/C/CSL201/2020-12-31❯ ./bstSort.o
Enter the number of elements to be sorted :5
Enter the elements in the array : 1
1
1
1
1
The Sorted array of elemets are: 1 1 1 1 1
..ograming/C/CSL201/2020-12-31❯ ./bstSort.o
Enter the number of elements to be sorted :8
Enter the elements in the array : 8 7 6 5 4 3 2 1
The Sorted array of elemets are: 1 2 3 4 5 6 7 8
..ograming/C/CSL201/2020-12-31❯
```

Experiment 23
BFS and DFS

Date: 10-01-2021

Aim: Implementation of depth first search and breadth first search using array

Data Structures used: Graphs, Array

Algorithm for Breadth First Search (bfs)

Input: The Graph data structure (G) and the starting node S
Output : The nodes in the graph traversed in bfs order
Data Structure : Graphs, queue

Steps

1. Step 1: Start
2. Step 2: let Q be a queue
3. Step 3: Q.enque(s)
4. Step 4: visit(s)
5. Step 5: mark s as visited
6. Step 6: while(Q is not empty) do
7. Step 1: v= Q.dequeue()
8. Step 2: for all nodes w of v in the graph g do
9. Step 1: if(w is not visited) then
10. Step 1: Q.enqueue(w)
11. Step 2: visit(w)
12. Step 3: mark w as visited
13. Step 2: endif
14. Step 3: done
15. Step 7: done
16. Step 8: stop

Algorithm for Depth First Search (dfs)

Input: The graph G and the starting node A
Output : All the elements in G traversed in DFS order
Data Structure used: Graph, stack

Steps

1. Step 1: Start
2. Step 2: let S be a stack
3. Step 3: S.push(A)
4. Step 4: while S is not empty do
5. Step 1: v = S.pop()
6. Step 2: if (v is not visited) then
7. Step 1: visit(v)
8. Step 2: mark v as visited
9. Step 3: push all adjacent vertex of v into the stack
10. Step 3: endif
11. Step 5: endWhile
12. Step 6: stop

Result: The program was successfully compiled and the desired output was obtained

Program Code

```
/* Breadth first and depth first search
 * Done By: Rohit Karunakaran
 */
#include<stdlib.h>
#include<stdio.h>

int dequeue(int *q,int *f, int *b){
    int elem = q[*f];
    if((*f)==(*b)){
        (*f)=(*b)=-1;
    }
    else{
        (*f)++;
    }
    return elem;
}

void enqueue(int *q, int *f, int *b, int elem)
{
    (*b) = (*b)+1;
    q[(*b)] = elem;
    if((*f)==-1){
        (*f)++;
    }
}

void bfs(int* vert, int** a_m, int nv,int ne){
    if(nv!=0){
        int queue[2*nv];
        int f=0, b=0;
        int visited[nv];
        int vc=0;

        int i=0; //nodes accessed.
        visited[0]=i; //visited the 0th node
        queue[f] = i;

        while(f!=-1){
            int c = dequeue(queue,&f,&b);

            for(int i = 0; i<nv;i++){  //itereate through all the edges

                if(a_m[c][i]==1){     //If an edge is connected to c

                    int flag=1;
                    for(int j = 0;j<=vc;j++) //check if the edge is visited
                    {
                        flag ==1;
                        if(visited[j]==i){
                            flag = 0;
                            break;
                        }
                    }
```

```
                    if(flag){ //If the edge is not visited then visit it....
                        enqueue(queue,&f,&b,i);
                        visited[++vc] = i;
                    }
                }
            }
        }

        for(int i = 0;i<=vc;i++){
            printf("%d ",vert[visited[i]]);
        }
    }
}

void dfs(int* vert, int** a_m, int nv,int ne){
    if(nv!=0){
        int stack[2*nv];
        int top=0;
        int visited[nv];
        int vc=-1;

        int i=0; //nodes accessed.
        stack[top] = i;

        while(top!=-1){
            int c = stack[top--];
            int flag = 1;
            for(int j = 0;j<=vc;j++){ //check if the edge is visited{
                if(visited[j]==c){
                    flag = 0;
                    break;
                }
            }

            if(flag){
                visited[++vc] = c;
                for(int i = 0;i<nv;i++){
                    if(a_m[c][i]==1){
                        stack[++top] = i;
                    }
                }
            }
        }

        for(int i = 0;i<=vc;i++){
            printf("%d ",vert[visited[i]]);
        }
    }
}

void main(){
    int nv,ne;

    printf("BFS and DFS implementation\n");
    printf("Enter the number of vertices: ");
```

```
    scanf("%d%*c",&nv);

    int** adj_matrix = (int**)calloc(nv,sizeof(int*));
    for(int i = 0;i<nv;i++)
        adj_matrix[i] = (int*)calloc(nv,sizeof(int));

    int *vertices = (int*)malloc(nv*sizeof(int ));

    printf("\nEnter the vertices of the Graph: ");
    for(int i=0;i<nv;i++){
        scanf("%d",&vertices[i]);
    }

    printf("Enter the number of edges: ");
    scanf("%d",&ne);

    printf("Enter the vetices connected by the edges in the form-> start end\n");
    for(int i=0;i<ne;){
        int s,e;
        if(scanf("%d %d",&s,&e)==2){
            adj_matrix[s][e]=1;
            adj_matrix[e][s]=1;
            i++;
        }
        else{
            printf("Enter the vertices in the correct format\n");

        }
    }
    printf("The Breadth first traversl: ");
    bfs(vertices, adj_matrix,nv,ne);
    printf("\n");


    printf("The Depth first traversal: ");
    dfs(vertices,adj_matrix,nv,ne);
    printf("\n");

}
```

Sample input and output

```
..ograming/C/CSL201/2021-01-10❯ gcc bfs_dfs.c -o bfs_dfs.o
..ograming/C/CSL201/2021-01-10❯ ./bfs_dfs.o
BFS and DFS implementation
Enter the number of vertices: 8

Enter the vertices of the Graph: 11 10 9 8 2 7 4 5
Enter the number of edges: 9
Enter the vetices connected by the edges in the form-> start end
0 1
1 2
0 5
0 3
3 4
5 4
5 6
6 7
7 3
The Breadth first traversl: 11 10 8 7 9 2 5 4
The Depth first traversal: 11 7 4 5 8 2 10 9
..ograming/C/CSL201/2021-01-10❯
```

```
..ograming/C/CSL201/2021-01-10❯ ./bfs_dfs.o
BFS and DFS implementation
Enter the number of vertices: 6

Enter the vertices of the Graph: 6 7 8 9 10 11
Enter the number of edges: 6
Enter the vetices connected by the edges in the form-> start end
0 1
0 4
0 3
1 2
1 5
3 5
The Breadth first traversl: 6 7 9 10 8 11
The Depth first traversal: 6 10 9 11 7 8
..ograming/C/CSL201/2021-01-10❯
```

Experiment 24
Quick Sort and Merge Sort

Date: 07-02-2021

Aim: Implement Quick sort and Merge Sort

Data Structures used: Array

Algorithm for Quick Sort (Quicksort)

Input: The array to be sorted and the index of the first and the last element
Output : The sorted Array
Data Structure : Array

Steps

1. Step 1: Start
2. Step 2: if(first<last) then
3.   Step 1: q = Partition(arr,first, last)
4.   Step 2: Quicksort(arr,first,q-1)
5.   Step 3: Quicksort(arr,q+1,last)
6. Step 3: endif
7. Step 4: Stop

Algorithm for Partition (Partition)

Input: The array to be partitioned and the first and the last node(also known as the pivot)
Output: The correct index of the pivot in the sorted array
Data Structure used: Array

Steps

1. Step 1: Start
2. Step 2: pivot = arr[last]
3. Step 3: i = first-1
4. Step 4: j = first
5. Step 5: while j<=last-1 then
6.   Step 1: if arr[j] <= pivot then
7.   Step 1: i=i+1
8.   Step 2: swap arr[i] and arr[j]
9.   Step 2: EndIf
10. Step 6: End While
11. Step 7: swap arr[i+1] and arr[last]
12. return i+1

Algorithm for Merge Sort (merge_sort)
Input: The array and the starting and the ending index of the array to be sorted
Output : The sorted array
Data Structure used: Array

Steps

1. Step 1: Start
2. Step 2: if(first<last) then
3. Step 1: (first+last)/2
4. Step 2: merge_sort(arr,first,mid)
5. Step 3: merge_sort(arr,mid+1,last)
6. Step 4: merge(arr,first,mid,last)
7. Step 3: Endif
8. Step 4: Stop

Algorithm for Merge (merge)

Input: The array and upperbound and the lower bound and the middle element in the array
Output : The array is sorted
Data Structure used: Binary trees

Steps

1. Step 1: Start
2. Step 2: n1 = middle-lower+1
3. Step 3: n2 = upper – middle
4. Step4: let L[1...n1+1] and R[1….n2+1]
5. Step 5: for i=1 to n1 do
6. Step 1: L[i] = arr[lower+i-1]
7. Step 6: Done
8. Step 7: for j = 1 to n2 do
9. Step 1: R[j] = arr[middle+j]
10. Step 8: done
11. Step9: L[n1+1] = ∞
12. Step 10: R[n2+1]= ∞
13. Step 11: i=1
14. Step 12: j=1
15. Step 13: for k=first to last
16. Step1: if L[i]<=R[j] then
17. Step 1: A[k] = L[i]
18. Step 2: i++
19. Step 2: else
20. Step 1: A[k] = R[j]
21. Step 2: j++
22. Step 3: endif
23. Step 14: Done
24. Step 15 :Stop

Program Code

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<time.h>

#define MAX_SIZE 100

typedef struct student_structure{
    char name[101];
    float height;
    float weight;
}student;

enum prop{NAME,HEIGHT,WEIGHT};
char prop_name[][10]={"Name","Height","Weight"};

/******************************
 * Quick Sort
 * ****************************/
int partition(student *list, int first, int pivot, enum prop a){
    int i,j;

    i = first;
    j = first-1;
    while(i<pivot){
        int flag = 0;

        switch(a){
            case NAME:
                if(strcmp(list[i].name,list[pivot].name)<=0) flag = 1;
                break;
            case HEIGHT:
                if(list[i].height<=list[pivot].height) flag = 1;
                break;
            case WEIGHT:
                if(list[i].weight<=list[pivot].weight) flag = 1;
                break;
        }
        if(flag){
            j++;
            student temp = list[i];
            list[i] = list[j];
            list[j] = temp;
        }
        i++;
    }
    j++;
    if(pivot != j){
        student temp = list[pivot];
        list[pivot] = list[j];
        list[j] = temp;
    }
    return j;
```

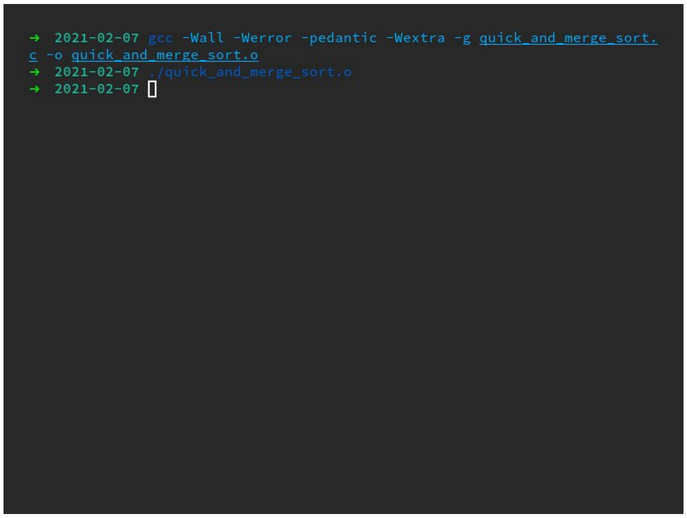
```
}

void quick_sort(student *list,int first,int last,enum prop a){
    if(first<last){
        int q = partition(list, first,last,a);
        quick_sort(list, first, q-1,a);
        quick_sort(list,q+1,last,a);
    }
}

/*******************************
 * Merge Sort
 * *****************************/
void merge(student *list,int first,int mid,int last,enum prop a){
    int n = last-first+1;
    student *temp =(student*)malloc(n*sizeof(student)) ;

    int i,j,flag,k=0;
    for(i=first,j=mid+1;i<=mid&&j<=last;)
    {
        flag = 0;
        switch(a){
            case NAME:
                if(strcmp(list[i].name,list[j].name)<0){
                    flag = 1;
                }
                break;
            case HEIGHT:
                if(list[i].height<=list[j].height){
                    flag =1;
                }
                break;
            case WEIGHT:
                if(list[i].weight<=list[j].weight){
                    flag =1;
                }
                break;
        }
        if(flag){ //if the flag is true then add i, else add j;
            strcpy(temp[k].name,list[i].name);
            temp[k].height= list[i].height;
            temp[k].weight = list[i].weight;
            i++;
        }
        else{
            strcpy(temp[k].name,list[j].name);
            temp[k].height= list[j].height;
            temp[k].weight = list[j].weight;
            j++;
        }
        k++;
    }
    while(i<=mid){
        temp[k] = list[i];
        i++;k++;
    }
```

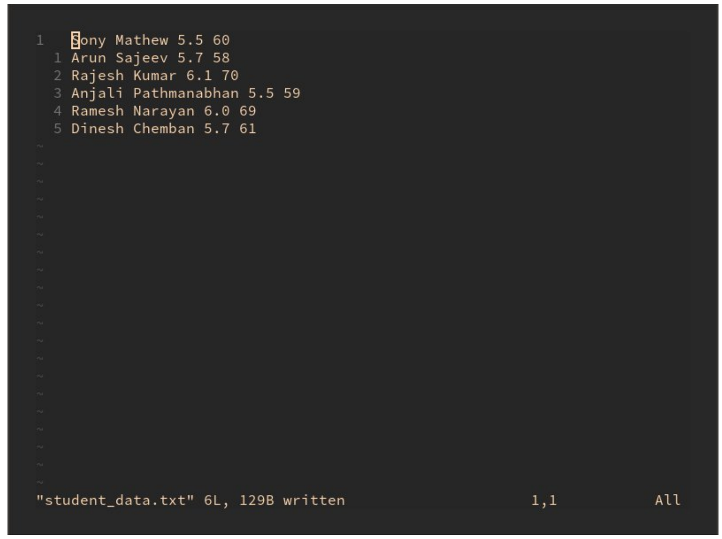
```
    while(j<=last){
        temp[k] = list[i];
        j++;k++;
    }
    k=0;
    for(i = first;i<=last;i++){
        strcpy(list[i].name,temp[k].name);
        list[i].height= temp[k].height;
        list[i].weight = temp[k].weight;
        k++;
    }
}

void merge_sort(student *list, int first, int last, enum prop a)
{
    if(first<last){
        int mid = (first+last)/2;
        merge_sort(list,first,mid,a);
        merge_sort(list,mid+1,last,a);
        merge(list,first,mid,last,a);
    }
}

void list_copy (student* l1, student* l2,int n){
    for(int i=0;i<n;i++){
        strcpy(l1[i].name,l2[i].name);
        l1[i].height = l2[i].height;
        l1[i].weight = l2[i].weight;
    }
}

int main(){

    student *student_list = (student*) malloc(MAX_SIZE*sizeof(student));
    student *temp_list = (student*) malloc(MAX_SIZE*sizeof(student));
//    FILE *file = fopen("./output.txt","w");

    char first_name[50];
    char last_name[50];
    int n = 0;
    int i;
    enum prop a = HEIGHT;

    clock_t t;
    double time_taken;

    if(freopen("./student_data.txt","r",stdin)){
        FILE *quickSortOp = fopen("./quicksortop.txt","w");
        FILE *mergeSortOp = fopen("./mergesortop.txt","w");
        while(scanf("%s %s %f %f\n", first_name,last_name,&(student_list[n].height),
                    &(student_list[n].weight))==4) {
            //conactinate the first and the last names
            strcat(student_list[n].name, first_name);
            strcat(student_list[n].name, " ");
            strcat(student_list[n].name, last_name);
            n++;
```

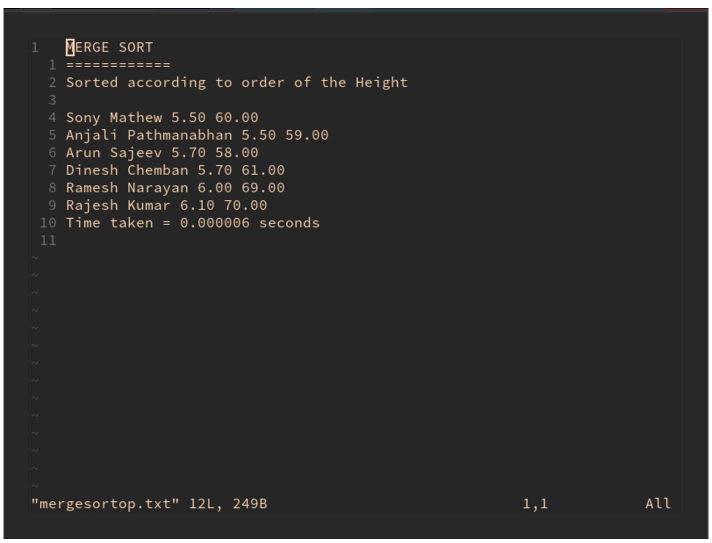
```
        }
        fprintf(quickSortOp,"QUICK SORT\n");
        fprintf(quickSortOp,"============\n");

//        for(int a=NAME;a<=WEIGHT;a++ ){ //For iterating through all the
            list_copy(temp_list,student_list,n);

            t = clock();
            quick_sort(temp_list,0,n-1,a);
            t = clock()-t;

            i=0;
            fprintf(quickSortOp,"Sorted according to order of the %s\n\
n",prop_name[a]);
            while(i<n){
                fprintf(quickSortOp,"%s %.2f %.2f\
n",temp_list[i].name,temp_list[i].height,temp_list[i].weight);
                i++;
            }

            time_taken = ((double)t)/(CLOCKS_PER_SEC);
            fprintf(quickSortOp,"Time taken = %lf seconds",time_taken);
            fprintf(quickSortOp,"\n\n");
 //       }

        fprintf(mergeSortOp,"MERGE SORT\n");
        fprintf(mergeSortOp,"============\n");

 //       for(int a = NAME; a<=WEIGHT;a++){
            list_copy(temp_list,student_list,n);

            t = clock();
            merge_sort(temp_list,0,n-1,a);
            t = clock()-t;

            i=0;
            fprintf(mergeSortOp,"Sorted according to order of the %s\n\
n",prop_name[a]);
            while(i<n){
                fprintf(mergeSortOp,"%s %.2f %.2f\
n",temp_list[i].name,temp_list[i].height,temp_list[i].weight);
                i++;
            }

            time_taken = ((double)t)/(CLOCKS_PER_SEC);
            fprintf(mergeSortOp,"Time taken = %lf seconds",time_taken);
            fprintf(mergeSortOp,"\n\n");
//        }
    }

    return 0;
}
```

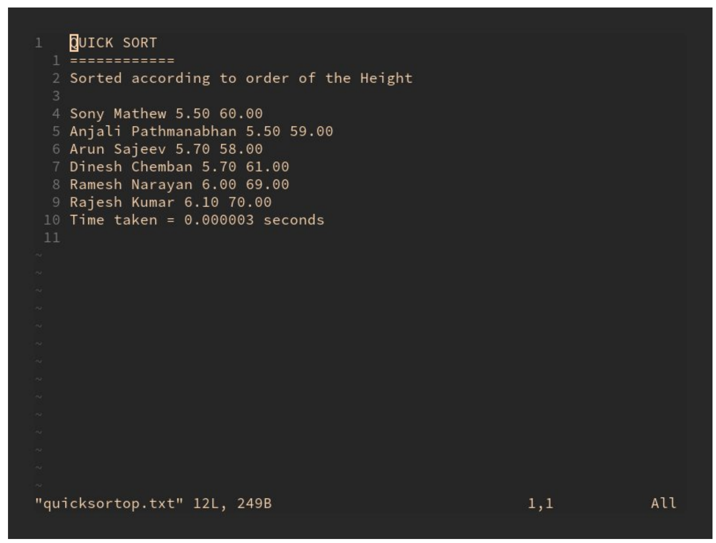
Result: The program compiled successfully and required output was obtained

Sample input and output









Experiment 25
Heap Sort

Date: 07-02-2021

Aim: Sort an array of numbers using heap sort and find an element in the array using binary search

Data Structures used: Array

Algorithm for Create Heap (create_heap)

Input: The array to be sorted and size of the array
Output : The elements of the array now follows the heap property
Data Structure : Array

Steps

1. Step 1: Start
2. Step 2: i = 1
3. Step 3: while i<=n do
4. Step 3: j = i
5. Step 4: while j>1 do
6. Step 1: if A[j] > A[j/2] then
7. Step 1: swap (A[j],A[j/2])
8. Step 2: j=j/2
9. Step 2: else
10. Step 1: j = 1
11. Step 3: endif
12. Step 5: EndWhile
13. Step 6: i = i+1
14. Step 4: endWhile
15. Step 5: Stop

Algorithm for Remove max (remove_max)

Input: The largest element in the heap and the index
Output: The largest and the element at the bottom of the heap
Data Structure used: Array

Steps
1. Step 1: Start
2. Step 2: temp = A[i]
3. Step 3: A[i] = A[1]
4. Step 4: A[1] = temp
5. Step 5: Stop

Algorithm for Rebuild Heap (rebuild_heap)
Input: The Array after the remove_max algorithm
Output: The array satisfies the heap property
Data Structure used: Array

Steps
1. Step 1: Start
2. Step 2: if(i == 1)then
3. Step 1: retun

4. Step 3: else
5. Step 1: j = 0
6. Step 2: flag = true
7. Step 3: while(flage == true) do
8. Step 1: leftchild = j*2
9. Step 2: rightchild = j*2+1
10. Step 3: largest = j
11. Step 4: if(leftchild<=i and A[largest]<A[leftchild]) then
12. Step 1: largest = leftchild
13. Step 5: endIf
14. Step 6: if(rightchild<=i and A[largest]<A[rightchild]) then
15. Step 1: largest = rightchild
16. Step 7: endIf
17. Step 8: if(largest!=j) then
18. swap(A[j], A[largest])
19. Step 9: else
20. Step 1: flag = flase
21. Step 10: endif
22. Step 4: endWhile
23. Step 4: Endif
24. Step 5: Stop

Result: The program was compiled successfully and the required output was obtained

Program Code

```
/***************************
 * Heap Sort
 * Done By: Rohit Karunakaran
 ***************************/
#include<stdio.h>
#include<stdlib.h>

void swap(int* arr, int i, int j){
    int temp = arr[i];
    arr[i] = arr[j];
    arr[j] = temp;
}

void create_heap(int *arr,  int n){
   int i = 0;
   int k,j;
   while(i<n){
        j = i;
        while(j>0){
            k = j%2==0?j/2-1:j/2;
            if(arr[j]>arr[k]){
                swap(arr,j,k);
                j = k;
            }
            else{
                j=0;
            }
        }
    i++;
   }
```

```
   //printf("Entered heap sort");
}

void heapify(int *arr, int i){
//i is the upper bound
    if(i == 0){
        return; //the array is sorted
    }
    else{
        int j=0;
        int flag = 1;
        while(flag){
            int largest = j;//initially assume the parent is the largerst which in
the first loop is'nt
            int lc = 2*j+1;
            int rc = 2*(j+1);

            if(lc<=i && arr[lc]>arr[largest])largest = lc;
            if(rc<=i && arr[rc]>arr[largest])largest = rc;

            if(j!=largest){
                swap(arr,j,largest);
            }
            else{
               // printf("swapped\n");
                flag =0; //if there is no change in the largest element then the
array is heapified
            }
        }

    }
}

void heap_sort(int *arr, int n){
    create_heap(arr,n);
    for(int i = n-1;i>=0;i--){
        swap(arr,i,0);
        heapify(arr,i-1);
    }
}

int binary_search(int *arr, int first, int last, int elem){
    if(first<=last){
        int mid = (first+last)/2;
       if(arr[mid]== elem){
            return mid;
       }
       else if(arr[mid]>elem){
            return binary_search(arr,first,mid-1,elem);
       }
       else{
            return binary_search(arr,mid+1,last,elem);
       }
    }
    else{
```

```
        return -1;
    }
}

int main(){
    int n;
    int elem;
    int* arr = (int*)malloc(20*sizeof(int));

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

    printf("Enter the elements : ");
    for(int i = 0; i<n;i++){
        scanf("%d",arr+i);
    }

    heap_sort(arr,n);
    printf("The sorted array is : ");

    for(int i = 0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\n");
    printf("Enter the element to be searched: ");
    scanf("%d", &elem);
    int index = binary_search(arr,0,n-1,elem);
    if(index!=-1){
        printf("The element is found at index %d\n",index);
    }
    else{
        printf("The element doesnt exist\n");
    }

    free(arr);

    return 0;
}
```

Sample input/output

```
➜  2021-02-07 ./heap_sort.o
Enter the number of elements: 6
Enter the elements : 12 13 18 91 2 1
The sorted array is : 1 2 12 13 18 91
Enter the element to be searched: 13
The element is found at index 3
➜  2021-02-07 ./heap_sort.o
Enter the number of elements: 6
Enter the elements : 12 0 9 3 6 12
The sorted array is : 0 3 6 9 12 12
Enter the element to be searched: 1
The element doesnt exist
```

Experiment 26
Hashing with Chaining

Date: 12-02-2021

Aim: Implementation of Hash table with chaining as the collition

Data Structure Used: Hash table

Algorithm for Inserting an element (insert)

Input: Element to be inserted, v. The hash table to which the element is inserted HT, and the hash function h(v)
Output: The element inserted in the hash table
Data Structure: Hash table

Steps:

Step 1: Start
Step 2: key = h(v)
Step 3: new = GetNode(Node)
Step 4: new→data = v
Step 5: new →next =NULL
Step 6: if(HT[key]!=NULL) then
    Step 4: HT[key] = new
Step 7: else
    Step 1: ptr = HT[key]
    Step 2: while(ptr→next != NULL) do
        Step 1: ptr = ptr→ next
    Step 3: EndWhile
    Step 4: ptr →next = new
Step 8: endif
Step 9: Stop

Program Code:

```
/***********************************
 * Implementiong of Hashing with
 * chaining
 * Done By: Rohit Karunakaran
 * ********************************/

#include<stdio.h>
#include<stdlib.h>

#define SIZE 10

typedef struct hash_node{
    int data;
    struct hash_node *next;
} node;

void insert(node ***ht, int e){
    int pos = e%SIZE;
    node **hash_table = *ht;
```

```
    node *new_node = (node*)malloc(sizeof(node));
    if(new_node !=NULL){
        new_node->data = e;
        new_node->next=NULL;
        if(hash_table[pos]==NULL){
            hash_table[pos] = new_node;
        }
        else{
            node *ptr = hash_table[pos];
            while(ptr->next!=NULL)ptr = ptr->next;
            ptr->next=new_node;
        }
    }
}

void show_the_hash_table(node **ht){
    int i;
    for(i=0;i<SIZE;i++){
        node *ptr = ht[i];
        if(ptr!=NULL){
            while(ptr->next!=NULL){
                printf("%d -> ",ptr->data);
                ptr = ptr->next;
            }
            printf("%d\n",ptr->data);
        }
    }
}

int main(){
    node **hash_table=(node**)calloc(sizeof(node),SIZE);
    int RUN=1;
    int elem;
    int c;

    while(RUN){
        printf("\nMENU\n");
        printf("1.Insert to hash table\n");
        printf("2.Display the hash table\n");
        printf("3.Exit\n");
        printf("Enter yout choice: ");

        scanf("%d",&c);

        switch(c){

            case 1:
                printf("Enter the element you want to enter : ");
                scanf("%d",&elem);
                insert(&hash_table, elem);
                break;
            case 2:
                printf("The hash table is : \n");
                show_the_hash_table(hash_table);
                break;
```

```
            case 3:
                RUN=0;
                break;
        }
    }
    return 0;
}
```

Result:
The program was successfully compiled and the required output was obtained.

<u>Sample input output:</u>

```
➜  2021-02-12 ./hashing_with_chaining.o

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 12

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 22

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 5

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 55
```

```
MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 10

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 34

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 2
The hash table is :
10
12 -> 22
34
5 -> 55

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 3
➜  2021-02-12
```

Experiment 27
Hashing with Linear Probing

Date: 12-02-2021

Aim: Implementation of Hash table with linear probing as the collition resolution method

Data Structure Used: Hash table

Algorithm for Inserting an element (insert)

Input: Element to be inserted, v. The hash table to which the element is inserted HT with the hash function h(v)
Output: The element inserted in the hash table
Data Structure: Hash table

Steps:

```
Step 1: Start
Step 2: key = h(v)
Step 3: new = GetNode(Node)
Step 4: new→value = v
Step 6: if(HT[key]!=NULL) then
        Step 4: HT[key] = new
Step 7: else
        Step 1: new_key = key+1
        Step 2: while(new_key!=key and HT[new_key]!=NULL) do
                Step 1: new_key = new_key+1
        Step 3: EndWhile
        Step 4: if(new_key == key) then
                Step 1: Print "Insertion not possible"
                Step 2: Exit
        Step 5: else
                HT[new_key] = new
        Step 6: endif
Step 8: endif
Step 9: Stop
```

Program Code:

```
/*************************************
 * Hashing with linear probing as the
 * collision resolution method
 *
 * Done By: Rohit Karunakaran
 ************************************/


#include<stdio.h>
#include<stdlib.h>

#define SIZE 10

void insert(int ***hash_table, int e,int i){
    if(i<=SIZE){
```

```
        int pos = (e%SIZE+i)%SIZE;
        int **ht = *hash_table;

        if(ht[pos]==NULL){
            int* node = (int*) malloc(sizeof(int));
            *node = e;
            ht[pos] = node;
        }
        else{
            insert(hash_table,e,i+1);
        }
    }
    else{
        printf("INSERTION NOT POSSIBLE!!!!!!\nThe Hash Table is full\n");
        return;
    }

}

void show_the_hash_table(int **ht){
    int i;
    for(i=0;i<SIZE;i++){
        int *ptr = ht[i];
        if(ptr!=NULL){
            printf("(%d) - %d\n",i,*ptr);
        }
    }
}

int main(){
    int **hash_table=(int**)calloc(sizeof(int*),SIZE);
    int RUN=1;
    int elem;
    int c;

    while(RUN){
        printf("\nMENU\n");
        printf("1.Insert to hash table\n");
        printf("2.Display the hash table\n");
        printf("3.Exit\n");
        printf("Enter yout choice: ");

        scanf("%d",&c);

        switch(c){

            case 1:
                printf("Enter the element you want to enter : ");
                scanf("%d",&elem);
                insert(&hash_table, elem,0);
                break;
            case 2:
                printf("The hash table is : \n");
                show_the_hash_table(hash_table);
                break;
            case 3:
```

```
                RUN=0;
                break;
        }

    }

    for(int i = 0; i<SIZE;i++){
        free(*(hash_table+i));
    }
    free(hash_table);

    return 0;
}
```

Sample input output:

```
➜ 2021-02-12 ./hashing_with_linear_probing.o

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 21

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 25

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 6

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 2
The hash table is :
(1) - 21
(5) - 25
(6) - 6

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 3
➜ 2021-02-12
```

```
➜ 2021-02-12 ./hashing_with_linear_probing.o

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 12

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 23

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 1
Enter the element you want to enter : 54

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 2
The hash table is :
(2) - 12
(3) - 23
(4) - 54

MENU
1.Insert to hash table
2.Display the hash table
3.Exit
Enter yout choice: 3
➜ 2021-02-12
```

Result: The program is successfully compiled and the required output is obtained