Disclaimer:

- Output Handwritten pani hunxa, print & paste compulsory hoina. (Source : Sir Himself)
- 1 - 10 ko Procedure lekhne tarika mileko xaina, ideally Practical No.14 (Pg: 30 in PDF) ma vaye jasari garnu parne ho, aauxa vane tei tarika le aafei garnu. (maile half jati lekhesake paxi maatra tha paaye, @ Kiran Gajmer bata)
- Question No.14 ko source code maa ali kati logical issue xa, correct output aaudeina, print garne huda ali milaayera output linu. (maile incorrect answer pani 'as it is' saardeko xu)
- Source Code ko handwriting nabujhiyema: https://github.com/aNMOL123961/Shared/tree/main/Discrete%20Structure

Contents:

1. WAP to find the ceiling and floor value of a given float number.

- Theory:-
  - Floor ~~value~~ function:- It is a function that gives integer number value which is less then the actual value.
  - Ceiling function :- It is a function that gives integer number value which is more than the actual value.

  These functions are included in the <math.h> or <cmath> library function and can be used by using floor(x) and ceil() keywords.

- Procedure :-
  i) Take a float value as input.
  ii) store it in a variable.
  iii) use floor(variable) and ceil(variable) to and display the output.

- Source code:-

```
#include <iostream>
#include <cmath>
 using namespace std;
 int main(){
 float a;
 cout<<"Enter any float number:"<<endl;
 cin>>a;
 cout<<"Floor value is: "<<floor(a)<<endl;
 cout<<"Ceiling Value is: "<<ceil(a);
 return 0;
}
```

- Output :-

```
Enter any float number:
6.9
Floor value is: 6
Ceiling value is: 7
```

- Conclusion :-

~~Using the program~~ source

- The floor value of number 6.9 was found to be 6 and ceiling value was found to be 7.

2- WAP to find the cartesian product of two given sets.

- Theory:-
  - Cartesian Product:- The cartesian product of two sets X and Y denoted by $X \times Y$ is the set of all ordered pairs (x,y), where x is an element of X and y is an element of Y.

  i.e.

  $$X \times Y = \{(x,y) \mid x \in X \text{ and } y \in Y\}$$

- Procedure:-
  i) Create an array and take input for two sets.
  ii) Use loops and functions to do cross product on both elements of different sets.
  iii) Display the result.

- Source code:-

```
#include <iostream>
  using namespace std;
  int main(){
  int a[35], b[35], i, j, m, n;
  cout<<" Enter no. of elements in P: ";
  cin >> m;
  cout<<" Enter elements of P: "<<endl;
  for (i=1; i<=m; i++){
  cin >> a[i];
  }
  cout<< "Enter no. of elements in Q: ";
  cin >> n;
  cout <<" Enter elements of Q: "<<endl;
  for (j=1; j<=n; j++){
  cin >> b[j];
  }
```

```
cout<<"Cartesian Product is: {";
for(i=1;i<=m;i++){
    for(j=1;j<=n;j++){
        cout<<"("<<a[i]<<","<<b[j]<<")";
        if((i*j)<m*n-1){
            cout<<",";}
        }
    }
}
cout<<"}";
}
```

- Output :-

```
D:\Class\DS\2.exe
Enter no. of element in P:3
Enter elements of P:
1       2       3
Enter no. of element in Q:3
Enter elements of Q:
4       5       6
Cartesian Product is: {(1,4),(1,5),(1,6),(2,4),(2,5),(2,6),(3,4),(3,5),(3,6)}
```

- Conclusion :-

The Cartesian product of sets :-
{1, 2, 3} and {4, 5, 6}
were found out to be :-
{(1,4), (1,5), (1,6), (2,4), (2,5), (2,6), (3,4) (3,5), (3,6)}

3. WAP to find the permutation by asking the value of n and r from user

- Theory:-
  - Permutation:- It is an arrangement of objects in a definite order. The members of sets are arranged in a sequence or linear order.
  - Formula of Permutation:-

$$P(n,r) = n!/(n-r)!$$

$$= n(n-1)(n-2)\ldots \text{upto } r \text{ factors.}$$

- Procedure:-
  1. Take the value of 'n' and 'r' and store it in a variable.
  2. Interpret ~~Use~~ permutation formula in a source code and, ~~to~~ calculate the answer.
  3. Display the result.

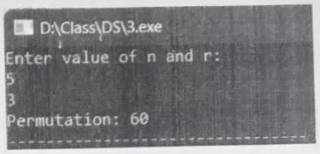
- Source code:-

```
#include <iostream>
  using namespace std;
  int main(){
      long int n,r,i,m=1,o=1,per;
      cout<<" Enter value of n and r : " endl;
      cin >>n>>r;
      for(i=n;i>=1;i--){
      m=m*i;
      }
```

Cont...

```
for(i=n-r; i>=1; i--){
o=o*i;
}
per=m/o;
cout<<"Permutation: "<<per;
}
```

- Output:-



- Conclusion:-
  - The Permutation of 5 and 3 was found out to be 60.
    i.e
    $P(5,3) = 60$

4. WAP to find the combination by asking the value of n and r from user.

- Theory:-
  - Combination:- It is a way of selecting items from a collection where the order of selection doesn't matter.
  - Formula:-

$$C(n,r) = \frac{n!}{r!(n-r)!} \quad \text{where } n > r.$$

- Procedure:-
  1. Take input of n and r from user and store it in variables.
  2. Interpret in C++ code to formula to calculate combination.
  3. Display the result.

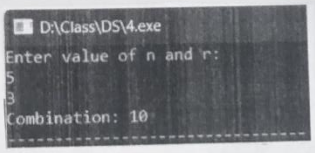
- Source code:-

```
#include <iostream>
using namespace std;
int main(){
    long int n, r, i, j, k, m=1, o=1, p=1, combi;
    cout<<" Enter value of n and r ; "<<endl;
    cin >> n >> r;
    for (i=n; i>=1; i--){
    m=m*i;
    }
    for (j=r ; j>=1 ;j--){
    p=p*j;
    }
```

cont...

```
for (k= n-r; K>=1 ; k--){
o=o*k;
}
combi= m/(o*p);
cout << "Combination: " <<combi;
return 0;
}
```

- Output:-



- ~~The~~ Conclusion :-

The combination of 5 and 3 were found out to be 10.

i.e. $C(5,3) = 10$ //

5. WAP to find the factorial of an input number using recursion.

- **Theory:-**
  - **Factorial:-** It is a function that multiplies a number by every number below it till 1.

    ~~Formula~~ Syntax:-

    $n! = n \times (n-1) \times (n-2) \times \cdots \times 3 \times 2 \times 1$.

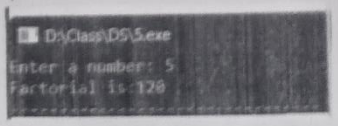
    Eg:- $5! = 5 \times 4 \times 3 \times 2 \times 1$
    $= 120$.

- **Procedure:-**
  1. Take ~~inpu~~ input and store the number in a variable.
  2. Use a recursive function to multiply $n \times (n-1)$ till ~~till~~ $n = 1$. Store the result in another variable.
  3. Display the result.

- **Source code:-**

```
#include <iostream>
using namespace std;
int fact (int n){
    if(n ==0)
        return 1;
    else
        return n* fact(n-1);
}
int main(){
    int n;
    cout <<" Enter a number: ";
    cin>>n;
    cout <<" Factorial is : "<<fact(n);
    return 0;
}
```

- Output :-



- Conclusion :-

The factorial of 5 was found out to be 120.

6. WAP to find the nth term of Fibonacci series using recursion.

- Theory :-
  - Fibonacci Series :- It is a series of number where each number is a sum of the two preceding ones.

  i.e.

  0 1 1 2 3 5 8 13 - - - -

- Procedure :-
  (i) Take n (integer) input from user and store it in a variable.
  (ii) Use recursive function to add n + (n-1) + (n-2) till n = 1. Starting from the number input by user
  (iii) Display the result.

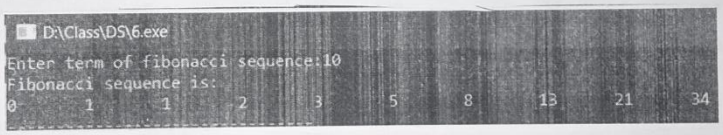
- Source code :-

```
#include <iostream>
using namespace std;
long int fib(long int n){
    if (n==0)
        return 0;
    else if(n==1)
        return 1;
    else
        return fib(n-1)+fib(n-2);
}
```

cont. -

```
int main(){
    long int n,k,m;
    cout<<"Enter term of Fibonacci sequence: ";
    cin>>m;
    cout<<"Fibonacci sequence is: "<<endl;
    for(n=0;n<m;n++){
        k=fib(n);
        cout<<k<<"\t";
    }
    return 0;
}
```

- Output:-



- Conclusion:-

. The first 10 numbers of Fibonacci series were found to be:-

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

7. WAP to raise the base of an input number by a certain power using recursion.

- Theory :-
  - Power of a number :- It is the total number of the multiples of the same number there are.

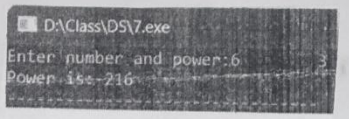
    Eg :- $2^3 = 2 \times 2 \times 2$,,

- Procedure :-
  (i) Take input for a number and its power and store it in a variable.
  (ii) User recursion to multiply the same number multiple times ~~and~~ till condition is satisfied.
  (iii) Display the result.

- Source Code :-

```
#include <iostream>
using namespace std;
long int power (long int n, long int a) {
    if(a ==0)
        return 1;
    else
        return n* power (n,(a-1));
}
int main(){
    long int n,a;
    cout << "Enter number and power: ";
    cin >> n >>a;
    cout<<" power is : " <<power(n,a);
}
```

- Output



- Conclusion:-

The value of 6^3 was found to be 216,,

8. WAP to perform linear search using recursive function.

- Theory :-
  - Linear Search :- It is a sequential search algorithm that starts at one end and goes through each element of a list until the desired element is found; otherwise the search continues till the end of the dataset.
- Procedure :-
  1) Take input of the elements of array, n and key and store the data
  2) Check the first element of array and compare it with key If the key matches with the element, print location of element else check next element.
  3) Check untill the result is found and display the result.
- Source Code :-

```
#linear search
#include <iostream>
  using namespace std;
  void linear search (int a[], int low, int high, int key){
      if (key == a[low])
          cout << "Search successful at location :" << low+1;
      else
          Linear search (a, low+1, high, key);
  }
  int main(){
      int a[35], n, i, key;
      cout << "Enter no. of elements: ";
```

```
    cin>>n;
    cout<<"Enter elements of array: "endl;
    for (i=0; i<n; i++){
        cin >> a[i];
    }
    cout<<"Enter element to search: ";
    cin >> key;
    linearsearch (a,0,n-1,key);
}
```

- Output :-

```
D:\Class\DS\8.exe
Enter no. of elements: 7
Enter elements of array:
1       2       3       5       6       7       9
Enter element to search:5
Searching successfull at location:4
```

- Conclusion :-

The location of '5' in the given array :-
1, 2, 3, 5, 6, 7, 9
was found to be '4'.

9. WAP to perform binary search using recursion

- Theory :-

  - Binary Search :- It is a search algorithm used in a sorted array by repeatedly dividing the search interval in half.

- Procedure :-

  i) Take input of elements of array and key and store the data. Elements of array should be in ascending order.

  ii) ~~check if~~ key Divide the array in half and check if key is ~~greater or~~ present in the lower half or upper half of array.

  iii) Repeat step (ii) until the key matches and show the result.

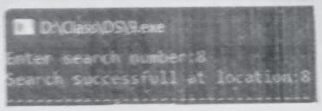
- Source code :-

```
#include <iostream>
using namespace std;
void binarysearch (int a[], int low, int high, int key)
    int mid;
    mid = (low + high)/2;
    if (key == a[mid])
        cout<<"Search successful at location : "<<mid
    else if (key < a[mid])
        binary search (a, low, mid-1, key);
    else if (key > a[mid])
        binary search (a, mid+1, high, key);
}
```

cont...

```
int main(){
    int a[] = {1,2,3,4,5,6,7,8,9,10};
    int low=0, high=10, key;
    cout << "Enter search number: ";
    cin >> key;
    binarysearch(a, low, high, key);
}
```

- Output;



- Conclusion:-

The location of '8' in given array
1, 2, 3, 4, 5, 6, 7, 8, 9, 10
was found to be 8.

(10) WAP to implement Euclidean Algorithm for computing GCD

- Theory:-

  - Euclidean Algorithm:- It is the method of finding the GCD of two numbers by dividing the larger by the smaller, the smaller by the remainder, the first remainder by the second remainder and so on until the exact division is obtained where the GCD is the exact divisor.

- Procedure:-

  i) Take input of two numbers and store the data.

  ii) Divide the larger number by smaller number and obtain the remainder. Set value of first number = second number, second number = remainder.

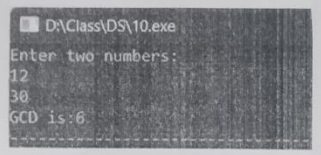
  iii) Repeat step (ii) until the second number becomes zero when the second number is zero, display the value of first number as GCD of original numbers.

- Source code:-

```
#include<iostream>
using namespace std;
int main(){
    int a,b,r;
    cout<<"Enter two numbers:"<<endl;
    cin>>a>>b;
    while(b>0){
    r=a%b
    a=b
    b=r;
    }
```

```
    cout<<"GCD is: "<<a;
}
```

- Output:-



- Conclusion:-

The GCD of 12 and 30 was found to be 6.

11. WAP to implement extended euclidean algorithm for computing GCD.

- Theory:-

Extended Euclidean Algorithm : It is an extension of euclidean Algorithm to find GCD of integers. In addition to finding GCD it also finds integer x and y such that

$$GCD(a,b) = ax + by$$

- Procedure:-

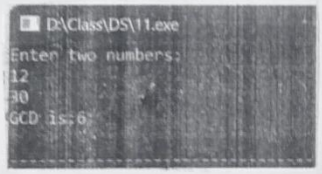
- Input :- Two integer a and b , $a \geq b$
- Output :- $d = gcd(a,b)$ and integers x and y satisfying $ax+by=d$.
- If $b=0$ then set $d=a, x=1, y=0$ and return (d, x, y)
- Set $x_2=1, x_1=0, y_2=0, y_1=1$
- While $b>0$, do:
  - $q = floor(a/b), r = a - qb, x = x_2 - qx_1, y = y_2 - qy_1$
  - $a = b, b = r, x_2 = x_1, x_1 = x, y_2 = y_1, y_1 = y$
- Set $d=a, x=x_2, y=y_2$ and return (d, x, y)

- Source code:-

```
#include<iostream>
#include<cmath>
  using namespace std;
  int main(){
      int a, b, d, x1, y1, q, r;
      cout<<" Enter two numbers : "<<endl;
      cin>>a>>b;
```

```
int x2=1, x1=0, y2=0, y1=1;
while (b>0) {
q=floor(a/b)
r=a-(q*b);
x=x2-(q*x1);
y=y2-(q*y1);
a=b; b=r; x2=x1; y2=y1;
x1=x; y1=y;
}
d=a; x=x2; y=y2;
cout<<" GCD is:"<<a<<endl;
}
```

- Output:-



- Conclusion:-

From program, the GCD of 12 and 30 was found to be 6.

12. WAP to find the Boolean ~~meet~~ join of two matrices

- Theory :-

  - Boolean join :- ~~It is when either of the values of two chosen~~ Boolean join is a method that converts boolean values into an unsigned integer signal such that when either of the signal is true/high the resulting signal will be true/high.

- Procedure :-

  - Input :- two matrices 'A' and 'B' with equal number of rows and columns
  - Output :- one matrix 'C' containing the result.
  - Take the first value of each matrices and perform OR operation on them.
  - ~~Assign the resultant in a new matrix~~
    ~~c[i][j] = a[i][j] || b[i][j];~~

- Source Code :-

```
#include<iostream>
    Using namespace std;
    int main(){
        int a[100][100], b[100][100], c[100][100];
        int m,n,p,q,i,j;
        cout<<" Enter row and column of A: ";
        cin >> m >>n;
        Cout<<" Enter elements of A: "<<endl;
```

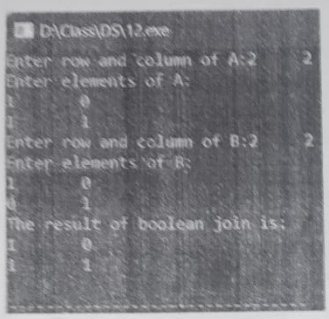
```
for (i=1; i<=m; i++){
  for (j=1; j<=n; j++){
      cin >>a[i][j];
  }
}
cout<<" Enter row and column of B : ";
cin >>p>>q;
cout<<" Enter elements of B : "<<endl;
for(i=1; i<=p; i++){
    for(j=1; j<=q; j++){
       cin >> b[i][j];
    }
}
for(i=1; i<=m; i++){
   for (j=1; j<=q; j++){
       c[i][j] = a[i][j] || b[i][j];
   }
}
cout <<" The result of boolean join is : "<<endl;
for (i=1; i<m; i++){
   for(j=1; j<=q; j++){
       cout << c[i][j] <<"\t";
   }
   cout <<endl;
}
```

→ output

}

→ conclusion

- Output:



- Conclusion:-
  - The result of boolean join between $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ was found to be $\begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$.

13. WAP to find the meet of two boolean matrices.

- Theory :- Boolean meet is a method of converting boolean value, into an unsigned integer signal such that when both signals are true / high the resulting signal will be high / true else the signals will be low / false.

- Procedure :-

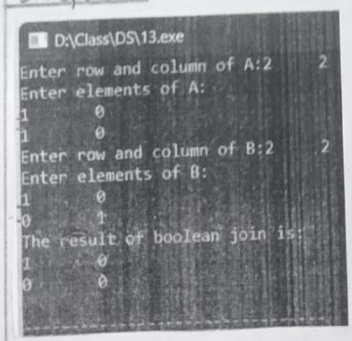
  - Input :- two matrices 'A' and 'B' with equal number of rows and columns
  - Output :- one matrix 'C' to ~~store~~ display result.
  - Process :- Take first values from each matrices :-
    - If both values are 1, set result as 1
    - If one of the values is 0, set result as 0

- Source code :-

```
#include <iostream>
using namespace std;
int main(){
    int a[100][100], b[100][100], c[100][100];
    int m, n, p, q, i, j;
    cout << " Enter rows and column of A : ";
    cin >> m >> n;
    cout << " Enter elements of A : " << endl;
    for (i = 1; i <= m; i++){
        for(j = 1; j <= n; j++){
            cin >> a[i][j];
        }
    }
```

```
cout << "Enter row and column of B :";
cin >> p >> q;
cout << "Enter elements of B:" << endl;
for (i=1; i<=p; i++){
    for (j=1; j<=q; j++){
        cin >> b[i][j];
    }
}
Cout << "The result of boolean join is :" << endl;
for (i=1; i<=m; i++){
    for(j=1; j<=q; j++){
        c[i][j]=a[i][j] && b[i][j];
        cout << c[i][j] << "\t";
    }
    cout << endl;
}
}
```

- Output:-



- Conclusion:-

The result of boolean meet between $\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ was found to be $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$.

14. WAP to find the product of two boolean matrices.

- Theory:-
  - Boolean product of two matrices:- It is very similar to matrix multiplication, except instead of multiplication, we use boolean products when calculating each entry in the matrix. Instead of addition we use boolean sum.
- Procedure:-
  - Input:- two matrices 'A' and 'B' with equal number of rows and column
  - Output:- One matrix 'C' that displays result.
  - Process:- When calculating each entry in the matrix:-
    - If 1∧1 then set result as 1 else set result as 0.
    - If 0∨0 then set result as 0 else set result as 1.
- Source code:-

```
#include <iostream>
using namespace std;
int main (){
int a[100][100], b[100][100], c[100][100];
int m,n,p,q, i, j, k, s=0;
cout <<" Enter raw and column of A:";
cin >> m>>n;
cout<<" Enter elements of A:"<<endl;
```

```
for(i=1; i<=m; i++){
    for(j=1; j<=n; j++){
        cin>>a[i][j];
    }
}
cout<<"Enter row and column of B:";
cin>>p>>q;
cout<<"Enter elements of B:"<<endl;
for(i=1; i<=p; i++){
    for(j=1; j<=q; j++){
        cin>>b[i][j];
    }
}
for(i=1; i<=m; i++)
    for(j=1; j<=q; j++)
        for(k=1; k<=n; k++){
            s=s||(a[i][k] && b[k][j]);
        }
        c[i][j]=s;
        s=0;
    }
}
cout<<"The result of boolean product is"<<endl;
for(i=1; i<=m; i++){
    for(j=1; j<=q; j++){
        cout<<c[i][j]<<"\t";
    }
    cout<<endl;
}
}
```

- Output :-

```
D:\Class\DS\14.exe
Enter row and column of A:2     2
Enter elements of A:
1       0
1       0
Enter row and column of B:2     2
Enter elements of B:
0       1
1       1
The result of boolean product is:
1       1
0       1
```

- Conclusion :-

  - The result of boolean product between $\begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$ and $\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ was found to be $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$.

10. WAP to find the union of two sets:-

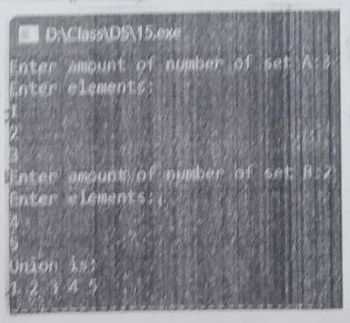
- Theory :-
  - Union of two sets :- It is a set containing all elements that are present in set A and B.
- Procedure :-
  - Input :- two set 'A' and 'B'
  - Output :- one set "U" containing result
  - Process :-
    - Display all elements of set A.
    - compare elements of set B with all elements of set A.
    - If elements of set A ≠ set B, display element of set B, else ignore it. repeat this step for all elements of set B.

- Source Code :-

```
#include <iostream>
using namespace std;
int main(){
    int a[100],b[100],i,n,m,j,count;
    cout<<" Enter amount of numbers of set A: ";
    cin>> m;
    cout<<" Enter elements: "<<endl;
    for(i=0;i<m;i++){
        cin>>a[i];
    }
```

```
cout << "Enter no of elements of set B: ";
cin >> n;
cout << "Enter elements: " << endl;
for (j=0; j<n; j++){
    cin >> b[j];
}
cout << "Union is: " << endl;
for (i=0; j<m; i++){
    cout << a[i] << " ";
}for(j=0; j<n; j++){
    count=0;
    for (i=0; j<m; i++){
        if(a[i] == b[j])
            count=1;
    }
    if(count!=1)
        cout << b[j] << " ";
}
}
```

- Output:



- Conclusion

- The union between {1,2,3} and {4,5} was found to be {1,2,3,4,5}

16. WAP to find the intersection of two sets:-

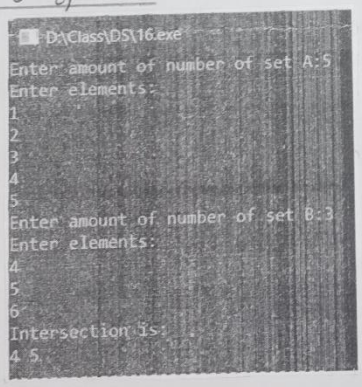
- Theory:-
  - Intersection of two sets :- It is a set containing only the common elements of two different sets.
- Procedure:-
  - Input:- two sets 'A' and 'B'
  - Output:- One set 'I' containing result.
  - Process:-
    - Compare each elements of set A and B
    - If element of set A = element of set B
      Display element of set A else ignore it.

Source code:

```
#include <iostream>
using namespace std;
int main(){
    int a[100], b[100], i, n, m, j;
    cout << "Enter amount of number of set A: ";
    cin>>m
    cout<<" Enter elements : "<<endl;
    for (i=0; i<m; i++){
        cin>> a[i];
    }
    cout<<" Enter amount of numbers of set B : ";
    cin>>n;
```

```
cout << "Enter elements : " << endl;
for (j = 0; j < n; j++){
    cin >> b[j];
}
cout << "Intersection is :" << endl;
for (i = 0; i < m; i++){
    for(j = 0; j < n; j++){
        if(a[i] == b[j]){
            cout << b[j] << " ";
        }
    }
}
}
```

- Output :-



- Conclusion :-

The intersection between {1,2,3,4,5} and {4,5,6} was found to be {4,5}.

(17) WAP to find set difference of two sets

- Theory :-
  - Set difference :- let A and B be two sets. then set difference of A and B (A-B) is a set that contains elements of A that are not in set B.
- Procedure :-
  - Input :- two set 'A' and 'B'
  - Output :- one set containing result.
  - Process :-
    - Compare elements of set A and B
    - If element of set A ≠ element of set B. display element of set A else ignore it.
- Source code :-

```
#include <iostream>
using namespace std;
int main(){
    int a[100], b[100], i, j, m, n;
    cout<<"Enter amount of numbers of set A: ";
    cin>>m;
    cout<<"Enter elements: "endl;
    for(i=0; i<m; i++){
        cin >> a[i];
}
```

```
cout << "Enter amount of numbers of set B: ";
cin >> n;
cout << "Enter elements: " << endl;
for (j = 0; j < n; j++) {
    cin >> b[j];
}
cout << "A-B is: " << endl;
for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
        if (b[j] != a[i]) {
            cout << a[i] << " ";
        }
        break
    }
}
}
```

- Output:-

```
D:\Class\DS\17.exe
Enter amount of number of set A:4
Enter elements:
1       2       3       4
Enter amount of number of set B:2
Enter elements:
3       6
A-B is:
1 2 4
```

- Conclusion:-

The Set difference between {1, 2, 3, 4} and {3, 6} was found to be {1, 2, 4}.

18. WAP to construct the truth table for negation.

- Theory :-

  - Negation :- It is the opposite of given mathematical statement.
    Notation for negation is $\overline{A}$
    i.e.
    - Negation of true/high is false/low.
    - Negation of false/low is true/high

- Truth table :-

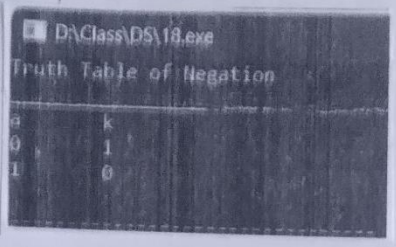
A	B
0	1
1	0

- Source Code :-

```
# include <iostream>
  Using namespace std;
  int negation(int a){
      if(a==1)
          return 0;
      else
          return 1;

  }
  int main(){
  int a,k;
  cout<<"Truth Table for Negation"<<endl;
  cout<<"_____"<<endl;
  Cout<<"a\tk"<<endl;
  for(a=0; a<=1; a++){
      k= negation(a);
      cout<<a<<"\t"<<k<<endl;
  }
}
```

- Output:-



- Conclusion:-

Truth table was verified and constructed using a C++ program.

19. WAP to create a truth table for conjuction.

- Theory:-

  - Conjuction:- It is a 'and' statement. For a conjuction to be true, both statements must be true. Notation used for conjuction is '$\wedge$'.

    i.e

    '$p \wedge q$' is true when both p and q statements are true.

- Truth table :-

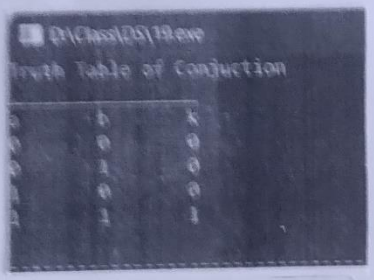
A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

- Source code:

```
#include<iostream>
using namespace std;
int conjuction(int a, int b){
    if(a==1 && b==1)
        return 1;
    else
        return 0;
}
```

```
int main ()
{
    int a, b, k;
    cout << " Truth Table of Conjuction " << endl;
    cout << "_____" << endl;
    cout << "a\t b\t k" << endl;
    for (a = 0; a <= 1 ; a++)
        for (b = 0; b <= 1 ; b++)
            k = conjuction (a,b) ;
            cout << a << "\t" << b << "\t" << k;
            cout << endl;
    }
}
}
```

- Output :-



- Conclusion :-

Truth table of conjuction was verified and constructed using a C++ program.

20. WAP to construct the truth table of Disjuction.

- Theory :-

  - Disjuction :- It is a 'or' statement. For a disjuction to be true, one of the statements must be true. Notation for disjuction is 'V'.

  i.e
  'pVq' is true when either p or q or both statements are true.

- Truth table :-

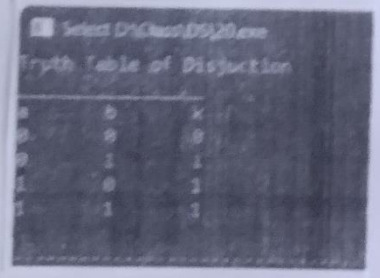
A	B	A V B
0	0	0
0	1	1
1	0	1
1	1	1

- Source code :-

```
#include <iostream>
using namespace std;
int disjuction ( int a , int b) {
    if (a == 1 || b == 1)
        return 1;
    else
        return 0;
}
```

```
int main (){
    int a,b,k;
    cout <<" Truth table of disjuction "<<endl;
    cout <<"_____"<<endl;
    cout <<" a\t b\t k" <<endl;
    for (a=0; a<=1; a++){
        for (b=0; b<=1; b++){
            k= disjuction (a, b);
            cout <<a<<"\t"<<b<<"\t"<<k;
            cout <<endl;
        }
    }
}
```

- Output:-



- Conclusion :-

Truth table for Disjuction was verified and constructed using a C++ program.

WAP to construct truth table for Implication.

- **Theory:-**

  - **Implication:-** It is the compound statement of the form "if p, then q". It is false only when p is true and q is false, else it will always be true. It is denoted by '$p \Rightarrow q$'.

- **Truth table:-**

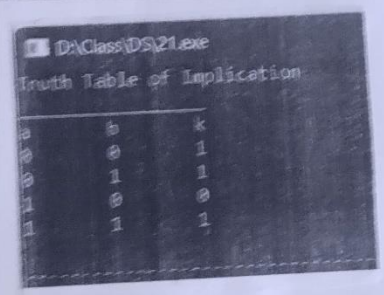
A	B	A ⇒ B
0	0	1
0	1	1
1	0	0
1	1	1

- **Source Code:-**

```
#include<iostream>
using namespace std;
int implication(int a, int b){
    if(a==1 && b==0)
        return 0;
    else
        return 1;
}
int main(){
    int a,b,k;
    cout<<" Truth Table of Implication "<<endl;
    cout<<" ____________________ "<<endl;
```

```
cout<< "a\t b\t k"<<endl;
for (a=0;a<=1;a++){
    for (b=0; b<=1; b++){
        k=implication (a,b);
        cout<<a<<"\t"<<b<<"\t"<<k<<endl;
    }
}
}
```

- Output :-



- Conclusion :-

Truth table for implication was verified and constructed using a C++ program.

22. WAP to construct a truth table of Bi-implication.

- Theory:

  - Bi-implication :- It is the compound statement of the form "p if and only if q". It is false when either both p and q are false or both p and q are true. else It is always true.
    It is denoted by 'p ⟺ q'.

- Truth table:-

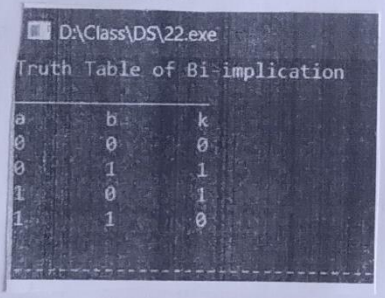
A	B	A ⟺ B
0	0	0
0	1	1
1	0	1
1	1	0

- Source Code:-

```
#include <iostream>
using namespace std;
int bi_implication (int a, int b){
    if ((a==0 && b==0) || (a==1 && b==1))
        return 0;
    else
        return 1;
}
```

```
int main(){
    int a,b,k;
    cout<<" Truth Table of Bi-implication "<<endl;
    cout<<"_____"<<endl;
    cout<<"a\t b\t k"<<endl;
    for (a=0;a<=1;a++){
        for(b=0; b<=1 ; b++) {
            k = bi_implication (a,b);
            cout<<a<<"\t"<<b<<"\t"<<k ;
            cout <<endl;
        }
    }
}
```

- Output :-



- Conclusion :-

Truth table for Bi-implication was constructed and verified using a C++ program.

23. WAP to check the validity of arguments using truth tables.

- Theory :-

  - Valid Arguments :- An argument is valid if the truth of the premise logically ~~guaran~~ guarantees the truth of the conclusion. If both the premise and conclusion is true then the argument is said to be valid.

- Truth table :-

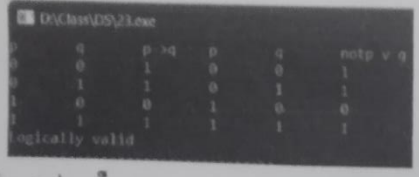
A	B	A ⟹ B	¬A ⟹ B
0	0	1	1
0	1	1	1
1	0	0	0
1	1	1	1

- Source Code :-

```
#include<iostream>
using namespace std;
int implication(int p, int q){
    if (p==1 && q==0)
        return 0;
    else
        return 1;
int other (int p, int q){
    int k;
    if (p==0)
        k=1;
    else
        k=0;
```

```
    if (k==1 || q==1)
        return 1;
    else
        return 0;
}
int main (){
    int p, q, r[4], s[4], i=0, count=0;
    cout << "p" << "\t" << "q" << "\t" << "p->q" << "\t";
    cout << "p" << "\t" << "q" << "\t" << "notp Vq" << endl;
    for (p=0; p<=1; p++) {
        for (q=0; q<=1; q++){
            r[i] = implication (p,q);
            cout << p << "\t" << q << "\t" << r[i] << "\t";
            s[i] = other (p,q);
            cout << p << "\t" << q << "\t" << r[i] << endl;
            i++;
        }
    }
    for (i=0; i<=3; i++){
        if (r[i] == s[i])
            count++;
    }
    if(count==4)
        cout << " Logically Valid";
    else
        cout << " Logically invalid";
}
```

- Output :-



- Conclusion :-

~~Kati~~ The statement was found out to be logically valid.