TRIBHUVAN UNIVERSITY

INSTITUTE OF SIENCE AND TECHNOLOGY

MECHI MULTIPLE CAMPUS

Bhadrapur, Jhapa

Lab report

Discrete Structure (CSC160)

Submitted by:

Bipul Chandra Gautam

Roll no: 07 (Seven)

Submitted to:

_____

Mr. Narayan Dhamala

Date:_____

Contents:

Lab no. 1:

Objective:
To find the ceiling and floor value of a given input float number.

Theory:
The ceiling value is the nearest integer that is greater than or equal to the specified value in the number line.
The floor value is the nearest integer that is smaller or equal to the specified value in the number line.

Algorithm:

1. Start,
2. Take input of a float number,
3. Use ceil() and floor() functions to find floor and ceiling values,
4. Display the values,
5. End,

Source code:

```
#include<iostream>
#include<cmath>

int main(){
    float n;
    std::cout << "Enter a number: ";
    std::cin >> n;

    int _ceiling , _floor;
    _ceiling = ceil(n);
    _floor = floor(n);

    std::cout << "Ceiling value of " << n << " is " << _ceiling << std::endl;
    std::cout << "Floor value of " << n << " is " << _floor << std::endl;

    return 0;
}
```

Output:

```
Enter a number: -3.2
Ceiling value of -3.2 is -3
Floor value of -3.2 is -4
```

Conclusion:
From the above program, the ceiling and floor value of a float number can be calculated.

Lab no. 2:

Objective:
To find the Cartesian product of two given sets.

Theory:
The cartesian product of two sets, A and B, denoted by A×B, is the set of all ordered pairs (x, y), where x is an element of A and y is an element of B.

$$A \times B = \{(x, y) \mid x \in A \land y \in B\}$$

Algorithm:

1. Start,
2. Take size of two sets as input,
3. Take two sets as input in two arrays,
4. Check whether the last element of first set is reached or not,
5. When last element of the second set is not reached, print the element of first and second set,
6. Go to next element of second set,
7. Go to step 5,
8. Go to next element of first set,
9. Go to step 4,
10. End,

Source code:

```cpp
#include<iostream>

int main(){
    int a, b;
    std::cout << "Enter size of first set: ";
    std::cin >> a;
    std::cout << "Enter size of second set: ";
    std::cin >> b;

    int set1[a], set2[b];

    std::cout << "Enter elements of first set: \n";
    for(int i=0; i<a; i++){
        std::cin >> set1[i];
    }
    std::cout << "Enter elements of second set: \n";
    for(int i=0; i<b; i++){
        std::cin >> set2[i];
    }

    std::cout << "Cartesian product of A and B:\n";
    std::cout << "{ ";
    for(int i=0; i<a; i++){
        for(int j=0; j<b; j++){
            std::cout << "(" << set1[i] << " , " << set2[j] << ")";
            if(j<b-1){ std::cout << ","; }
        }
    }
    std::cout << " } ";

    return 0;
}
```

Output:

```
Enter size of first set: 3
Enter size of second set: 2
Enter elements of first set:
1
2
3
Enter elements of second set:
6
7
Cartesian product of A and B:
{ (1 , 6),(1 , 7)(2 , 6),(2 , 7)(3 , 6),(3 , 7) }
```

Conclusion:

From the above program, the cartesian product of two given set can be found.

Lab no. 3:

Objective:
To find the permutation by asking the value of n and r from user.

Theory:
Permutation is the number of ways element or objects can be arranged in a particular order. With the permutation the order of elements matters. Permutation can be calculated using the following formula:

$$P(n,r) = \frac{n!}{(n-r)!}$$

Where,

n = total no of items or objects,
r = items taken from the set,

Algorithm:

1. Start,
2. Take the value of n and r as input from user,
3. Calculate the permutation, p(n, r) = n! / (n-r)!,
4. Display the result,
5. End,

For factorial (fact() function):

1. Take a integer x,
2. If x = 0, return 1,
   otherwise, return x × fact(x-1),

Source code:

```cpp
#include <iostream>

int fact(int x){
    if(x == 0){ return 1;}
    return x * fact(x-1);
}

int main(){
    int n, r;
    std::cout << "Enter value of n: ";
    std::cin >> n;
    std::cout << "Enter value of r: ";
    std::cin >> r;

    std::cout << "P(" << n << "," << r << ") = "<< fact(n)/fact(n-r);

    return 0;
}
```

Output:

```
Enter value of n: 5
Enter value of r: 2
P(5,2) = 20
```

Conclusion:

From the above program, permutation can be calculated from the provided values of n and r.

Lab no. 4:

Objective:
To find the combination by asking the value of n and r from user.

Theory:
A combination is a mathematical technique that determines the number of possible arrangements in a collection of items where the order or the selection does not matter. Combination can be calculated by using the following formula:

$$C(n,r) = \frac{n!}{(n-r)!\,r!}$$

Where,

n = total no of items,
r = items taken from the set,

Algorithm:

1. Start,
2. Take the value of n and r from user as input,
3. Calculate and display the value of combination, C(n, r) = n!/{(n-r)! . r!},
4. End,

For factorial (fact() function):

1. Take a integer x,
2. If x = 0, return 1,
   otherwise, return x × fact(x-1),

Source code:

```
#include <iostream>

int fact(int x){
    if(x == 0){ return 1;}
    return x * fact(x-1);
}

int main(){
    int n, r;
    std::cout << "Enter value of n: ";
    std::cin >> n;
    std::cout << "Enter value of r: ";
    std::cin >> r;

    std::cout << "C(" << n << "," << r << ") = " << fact(n)/(fact(r)*fact(n-r));

    return 0;
}
```

Output:

```
Enter value of n: 5
Enter value of r: 2
C(5,2) = 10
```

Conclusion:

Using the above program, the value of combination can be calculated with the given value of n and r.

Lab no. 5:

Objective:
To find the factorial of an input number using recursive function.

Theory:
Factorial of a non-negative number n, denoted by "n!" is the product of all positive integer from 1 up to n.

$$n! = n \times (n-1) \times (n-2) \times \ldots\ldots\ldots \times 2 \times 1$$

or

$$n! = n \times (n-1)!$$

Algorithm:

1. Start,
2. Take a number n as input from user,
3. Calculate factorial using fact() function and display the result,
4. End,

For fact() Function:

1. take a number x,
2. if x = 0, return 1 otherwise return x × fact(x-1),

Source code:

```
#include "iostream"

int fact(int x){
    if(x == 0){ return 1;}
    return x * fact(x-1);
}

int main(){
    int n;
    std::cout << "Enter a number to find factorial: ";
    std::cin >> n;
    std::cout << n << "! = " << fact(n);

    return 0;
}
```

Output:

```
Enter a number to find factorial: 5
5! = 120
```

Conclusion:
Using the above program, we can find the factorial of given input.

Lab no. 6:

Objective:
To find the nth term of a Fibonacci sequence using recursive function.

Theory:
The Fibonacci sequence is a sequence in which each term is the sum of the two preceding terms. Fibonacci sequence is:

0, 1, 1, 2, 3, 5, 8, 13,…….

Where,
First term = 0
Second term = 1
n^{th} term = $(n-1)^{th}$ term + $(n-2)^{th}$ term

Algorithm:

1. Start,
2. Take the value of n as input,
3. Calculate the n^{th} term of Fibonacci sequence using the fib() function and display the value,
4. End,

For fib() function:

1. Take a number x,
2. If x = 0, return 0
   If x = 1, return 1
   otherwise, return fib(x-1) + fib(x-2),

Source code:

```
#include "iostream"

int fib(int x){
    if(x == 0){
        return 0;
    }
    else if(x == 1){
        return 1;
    }
    return fib(x-1) + fib(x-2);
}

int main(){
    int n;
    std::cout << "Enter the value of n to find nth term of Fibonacci sequence: ";
    std::cin >> n;

    std::cout <<  n << "th term of fibonacci sequence is: " << fib(n);


    return 0;
}
```

Output:

```
Enter the value of n to find nth term of Fibonacci sequence: 6
6th term of fibonacci sequence is: 8
```

Conclusion:

Using the program mentioned above, with the provided input of n, the n^{th} term of Fibonacci sequence can be found.

Lab no. 7:

Objective:
To raise the base of an input number by a certain power using recursive function.

Theory:
The power of a number is a mathematical concept that represents the result of multiplying a number by itself a certain number of times. In mathematical notation, the power of a number is written as a^n or a^n, Where "a" is the base and "n" is the exponent or power. Which represents the result of "a" multiplied by itself "n" times. In other words:

$$a^n = a \times a^{n-1}$$

Algorithm:

1. Start,
2. Take base and power as input,
3. Calculate power using pow function and display the result,
4. End,

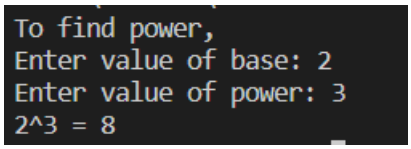
For pow() function:

1. Take two number base and power,
2. If power = 0, return 1
   Otherwise, return (base × pow(base, power-1))

Source code:

```
#include "iostream"

int pow(int base, int power){
    if(power == 0){
        return 1;
    }
    return base * pow(base, power-1);
}

int main(){
    int base, power;
    std::cout << "To find power,\nEnter value of base: ";
    std::cin >> base;
    std::cout << "Enter value of power: ";
    std::cin >> power;

    std::cout << base << "^" << power << " = " << pow(base, power);

    return 0;
}
```

Output:



Conclusion:

Using the above program, we can calculate the power of integer with given value of base and power.

Lab no. 8:

Objective:
To perform linear search by using recursive function.

Theory:
Linear search is an simple algorithm, which is used to search an specified element in a list. It works by sequentially checking the element of the list until a match is found or until all elements have been checked.

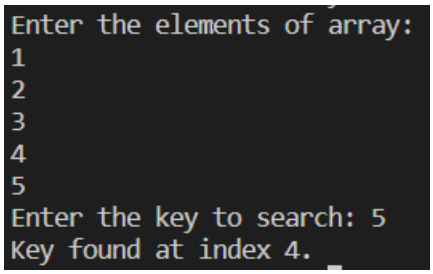
Algorithm:

1. Start,
2. Take input of an array,
3. Take a key element as input for searching,
4. Start searching from the beginning of the array,
5. Compare the key value and the element of the list,
6. If the value matches, return the index of element,
   if the value doesn't match, move to the next element, and repeat from step 5,
7. Continue step 5-6 until a match is found or all elements have been checked,
8. If no match found, return special value (such as -1).
9. End,

Source code:

```
#include "iostream"

int search(int arr[], int key, int start, int size){
    if(start == size){ return -1; }
    if(key == arr[start]){ return start; }
    return search(arr, key, start+1, size);
}

int main(){
    int n,key;
    std::cout << "Enter the size of array: ";
    std::cin >> n;
    int arr[n];
    std::cout << "Enter the elements of array:\n";
    for(int i=0; i<n; i++){ std::cin >> arr[i]; }
    std::cout << "Enter the key to search: ";
    std::cin >> key;

    int index = search(arr, key, 0, n);
    if(index != -1){
        std::cout << "Key found at index " << index << ".\n";
    }else{ std:: cout << "Not found!\n"; }



    return 0;
}
```

Output:



Conclusion:

From the above program, we can search a element in a list using linear search algorithm.

Lab no. 9:

Objective:
To perform binary search by using recursive function.

Theory:
Binary search is a searching algorithm that works by repeatedly dividing the search interval in half. The element is always searched in the middle portion of an array. This algorithm can be implemented only on a sorted list. If the elements are not sorted already, we need to sort them first.

Algorithm:

1. Start,
2. Take a sorted array and a key element as input,
3. Search the key in the array using search() function,
4. Display the result,
5. End,

For search() Function:

1. Take a array, key value to search, starting and ending index of array,
2. If starting index > ending index, return not found,
   otherwise calculate middle index, mid = (start + end)/2,
3. If the element at middle index match with key, return middle index,
4. If key element > element at middle index, set the starting index to middle index + 1 and recur to the search() function,
5. If key element < element at middle index, set the ending index to middle index – 1 and recur to the search() function,

Source code:

```cpp
#include <iostream>

int search(int arr[], int key, int start, int end){
    if(start > end){ return -1; }
    int mid = (end+start)/2;
    if(arr[mid] == key){ return mid; }
    else if(arr[mid] > key){ return search(arr, key, start, mid-1); }
    else if(arr[mid] < key){ return search(arr, key, mid+1, end); }
}

int main(){
    int n,key;
    std::cout << "Enter the size of array: ";
    std::cin >> n;
    int arr[n];
    std::cout << "Enter the elements of array in sorted order:\n";
    for(int i=0; i<n; i++){ std::cin >> arr[i]; }
    std::cout << "Enter the key to search: ";
    std::cin >> key;

    int index = search(arr, key, 0, n-1);
    if(index != -1){
        std::cout << "Key found at index " << index << ".\n";
    }else{ std:: cout << "Not found!\n"; }
    return 0;
}
```

Output:

```
Enter the size of array: 5
Enter the elements of array in sorted order:
4
5
6
7
9
Enter the key to search: 8
Not found!
```

Conclusion:

Using the above program, an element in a list can be searched using with the help of binary search algorithm.

Lab no. 10:

Objective:
To implement Euclidean algorithm for computing GCD.

Theory:
The Euclidean algorithm is a method for finding the GCD of two integers. It is based on the principle that the GCD of two numbers is the same as the GCD of the smaller number and the remainder of the larger number divided by the smaller number. Let “a” and “b” are two integers where, a > b, than:

$$GCD(a, b) = GCD(b, a \bmod b)$$

Algorithm:

1. Start,
2. Take input of two integers, a and b,
3. Check b > 0 or not,
4. If b > 0, remainder = a mod b
   set a = b
   set b = remainder
   if b = 0, GCD(a, b) = a,
5. Repeat step 3-4 until b becomes 0,
6. End,

Source code:

```
#include "iostream"

int GCD(int a, int b){
    if(b==0) return a;

    return GCD(b, a%b);
}

int main(){
    int a, b;
    std::cout << "To find GCD,\nEnter first number: ";
    std::cin >> a;
    std::cout << "Enter second number: ";
    std::cin >> b;

    std::cout << "GCD of " << a << " and " << b << " is " << GCD(a, b) <<
std::endl;
    return 0;
}
```

Output:

```
To find GCD,
Enter first number: 20
Enter second number: 56
GCD of 20 and 56 is 4
```

Conclusion:

Using the above program, the GCD of two integers can be calculated.

Lab no. 11:

Objective:

To implement Extended Euclidean algorithm for computing GCD.

Theory:

Extended Euclidean algorithm finds GCD of two numbers “a” and “b” and also Bezout’s coefficients “x” and “y” such that:

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

Algorithm:

1. Start,
2. Take two positive integer a and b as input,
3. If b = 0, set GCD = a, x=1, y=0 and return GCD, x, and y,
4. Set $x_2 = y_1 = 1$, $x_1 = y_2 = 0$,
5. Set q = floor(a/b), r= a-qb, $x = x_2-qx_1$, $y = y_2-qy_1$,
6. Set, a = b, b = r, $x_2=x_1$, $x_1=x$, $y_2=y_1$, $y_1=y$,
7. Repeat step 5-6 until b becomes 0,
8. Set GCD = a, $x = x_2$, $y = y_2$,
9. Display GCD, x, and y,
10. End,

Source code:

```
#include "iostream"
#include "cmath"

int main(){
    int a, b, x, y, gcd;
    std::cout << "Enter two numbers to find GCD:\n";
    std::cin >> a >> b;
    int temp1 = a, temp2 = b;
    if(b == 0){
        gcd = a;
        x = 1; y = 0;
    }
    int x1=0, x2=1, y1=1, y2=0, q, r;
    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;
    }
    gcd = a; x=x2; y=y2;
    std::cout << "From extended Euclidean algorithm,\n";
    std::cout << "GCD(" << temp1 << ", " << temp2 << ") = " << gcd;
    std::cout << "\nx = " << x << "\ny = " << y;

    return 0;
}
```

Output:

```
Enter two numbers to find GCD:
36
48
From extended Euclidean algorithm,
GCD(36, 48) = 12
x = -1
y = 1
```

Conclusion:

From the above program, we can calculate GCD of two given integer and Bezout's coefficients using extended Euclidean algorithm.

Lab no. 12:

Objective:

To find the Boolean join of two input matrices.

Theory:

The Boolean join of two matrices A and B having same size can be represented as A ∨ B and is defined as,

$$(A\ V\ B)ij = \begin{cases} 1, if\ Aij\ V\ Bij = 1 \\ 0, \quad Otherwise \end{cases}$$

Algorithm:

1. Start,
2. Take two Boolean matrices A and B of same size as input,
3. Create new Boolean matrix “result” of same size,
4. For each element of result, set it to the Boolean OR of the corresponding elements in A and B,
5. Display the result,
6. End,

Source code:

```
#include<iostream>

int main(){
    int m, n;
    std::cout << "For Boolean matrices of size m*n,\nEnter m: ";
    std::cin >> m;
    std::cout << "Enter n: ";
    std::cin >> n;

    bool matrix1[m][n], matrix2[m][n], result[m][n];

    std::cout << "Enter the elements of first metrix:\n";
    for(int i=0; i<m; i++){
        for(int j=0; j<n; j++){
            std::cin >> matrix1[i][j];
        }
    }
    std::cout << "Enter the elements of second metrix:\n";
    for(int i=0; i<m; i++){
        for(int j=0; j<n; j++){
            std::cin >> matrix2[i][j];
        }
    }
    for(int i=0; i<m; i++){
        for(int j=0; j<n; j++){
            result[i][j] = (matrix1[i][j] | matrix2[i][j]);
        }
    }
    std::cout << "The Boolean join of two matrices is:\n";
    for(int i=0; i<m; i++){
        for(int j=0; j<n; j++){
```

```
            std::cout << result[i][j] << " ";
        }
        std::cout << std::endl;
    }

    return 0;
}
```

Output:

```
For Boolean matrices of size m*n,
Enter m: 2
Enter n: 3
Enter the elements of first metrix:
0 1 0
1 0 1
Enter the elements of second metrix:
1 1 0
0 1 1
The Boolean join of two matrices is:
1 1 0
1 1 1
```

Conclusion:

From the above program, we can find the Boolean join of two Boolean matrices given as input.

Lab no. 13:

Objective:
To find the meet of two Boolean matrices.

Theory:
The Boolean meet of two matrices A and B having same size can be represented as A ∧ B and is defined as,

$$(A \wedge B)ij = \begin{cases} 1, if\ Aij \wedge Bij = 1 \\ 0, \quad Otherwise \end{cases}$$

Algorithm:

1. Start,
2. Take two Boolean matrices A and B of same size as input,
3. Create new Boolean matrix “result” of same size,
4. For each element of result, set it to the Boolean AND of the corresponding elements in A and B,
5. Display the result,
6. End,

Source code:

```
#include<iostream>

int main(){
    int m, n;
    std::cout << "For Boolean matrices of size m*n,\nEnter m: ";
    std::cin >> m;
    std::cout << "Enter n: ";
    std::cin >> n;

    bool matrix1[m][n], matrix2[m][n], result[m][n];

    std::cout << "Enter the elements of first metrix:\n";
    for(int i=0; i<m; i++){
        for(int j=0; j<n; j++){
            std::cin >> matrix1[i][j];
        }
    }
    std::cout << "Enter the elements of second metrix:\n";
    for(int i=0; i<m; i++){
        for(int j=0; j<n; j++){
            std::cin >> matrix2[i][j];
        }
    }
    for(int i=0; i<m; i++){
        for(int j=0; j<n; j++){
            result[i][j] = (matrix1[i][j] & matrix2[i][j]);
        }
    }
    std::cout << "The Boolean meet of two matrices is:\n";
```

```
    for(int i=0; i<m; i++){
        for(int j=0; j<n; j++){
            std::cout << result[i][j] << " ";
        }
        std::cout << std::endl;
    }

    return 0;
}
```

Output:

```
For Boolean matrices of size m*n,
Enter m: 2
Enter n: 3
Enter the elements of first metrix:
1 0 1
0 1 0
Enter the elements of second metrix:
1 1 0
0 1 1
The Boolean meet of two matrices is:
1 0 0
0 1 0
```

Conclusion:

From the above program, we can find the Boolean meet of two Boolean matrices given as input.

Lab no. 14:

Objective:
To find the product of two Boolean matrices.

Theory:
Boolean product of two Boolean matrices is similar to matrix multiplication where, + is replace by join(V) and × is replaced by meet(∧) operators. Let A and B are two Boolean matrices than the product of A and B is C (a Boolean matrix) where

$$Cij = (Ai1 \wedge B1j) V (Ai2 \wedge B2j) V \dots\dots\dots V (Ain \wedge Bnj)$$

Algorithm:

1. Start,
2. Take two Boolean matrices A and B as input,
3. Check If no of column of A is equal to no of row of B. If not, print an error massage and terminate the program,
4. Create a new matrix “result” with same number of rows as A and the same number of columns as B,
5. For each element (result)ij, iterate through the elements in the i^{th} row of A and j^{th} column of B,
6. Take the Boolean AND of corresponding element,
7. Perform Boolean OR on result from step 6 and store it in (result)ij,
8. Repeat step 5-7 until all elements of result have been computed,
9. Display the result,
10. End,

Source code:

```
#include <iostream>
using std::cout;
using std::cin;

int main(){
    int m1, n1, m2, n2;
    cout << "For first Boolean matrix,\nEnter the no of rows: ";
    cin >> m1;
    cout << "Enter the no of columns: ";
    cin >> n1;
    cout << "For second Boolean matrix,\nEnter the no of rows: ";
    cin >> m2;
    cout << "Enter the no of columns: ";
    cin >> n2;
    if(n1 != m2){
        cout << "Wrong size!";
        return 0;
    }
    bool matrix1[m1][n1], matrix2[m2][n2], result[m1][n2];

    cout << "Enter the elements of first matrix:\n";
    for(int i=0; i<m1; i++){
        for(int j=0; j<n1; j++){ cin >> matrix1[i][j]; }
```

```
    }
    cout << "Enter the elements of Second matrix matrix:\n";
    for(int i=0; i<m2; i++){
        for(int j=0; j<n2; j++){ cin >> matrix2[i][j]; }
    }

    for(int i=0; i<m1; i++){
        for(int j=0; j<n2; j++){
            result[i][j] = 0;
            for(int k=0; k<n1; k++){
                 result[i][j] += matrix1[i][k] * matrix2[k][j];
            }
        }
    }

    cout << "Boolean product of two matrices is:\n";
    for(int i=0; i<m1; i++){
        for(int j=0; j<n2; j++){
            cout << result[i][j] << " ";
        }
        cout << std::endl;
    }

    return 0;
}
```

Output:

```
For first Boolean matrix,
Enter the no of rows: 3
Enter the no of columns: 3
For second Boolean matrix,
Enter the no of rows: 3
Enter the no of columns: 3
Enter the elements of first matrix:
1 0 1
1 1 0
0 0 1
Enter the elements of Second matrix matrix:
0 1 0
1 1 1
0 0 1
Boolean product of two matrices is:
0 1 1
1 1 1
0 0 1
```

Conclusion:

From the above program, the Boolean product of two matrices given by user can be calculated.

Lab no. 15:

Objective:
To find the union operation of two sets.

Theory:
The union of two sets A and B denoted by “A ∪ B” is equal to the set of elements that are present in set A, in set B, or in both the sets A and B. This operation can be represented as:

$$A \cup B = \{a \mid a \in A \text{ or } a \in B\}$$

Algorithm:

1. Start,
2. Take input of two sets,
3. Display the elements of first set,
4. Display the elements of second set that are not present in first set,
5. End,

Source code:

```
#include<iostream>

int main(){
    int n, p;

    std::cout << "Enter size set A: ";
    std::cin >> n;
    int set1[n];
    std::cout << "Enter elements of set A:\n";
    for(int i=0; i<n; i++){ std::cin >> set1[i]; }

    std::cout << "Enter size set B: ";
    std::cin >> p;
    int set2[p];
    std::cout << "Enter elements of set B:\n";
    for(int i=0; i<p; i++){ std::cin >> set2[i]; }

    std::cout << "The union of set is:\n A U B = {";
    for(int i=0; i<n; i++){
        std::cout << set1[i];
        if(i < (n-1)){ std::cout << ", "; }
    }
    for(int i=0; i<p; i++){
        bool condition = true;
        for(int j=0; j<n; j++){
            if(i==0 && j==0){ std::cout << ", "; }
            if(set2[i] == set1[j]){
                condition = false;
                break;
            }
        }
        if(condition){
            std::cout << set2[i];
            if(i < (p-1)){ std::cout << ", "; }
        }
    }
```

```
    std::cout << "}";

    return 0;
}
```

Output:

```
Enter size set A: 5
Enter elements of set A:
1
2
3
4
5
Enter size set B: 3
Enter elements of set B:
2
3
4
The union of set is:
 A U B = {1, 2, 3, 4, 5, }
```

Conclusion:

The union of two given set was calculated using the program mentioned above.

Lab no. 16:

Objective:
To find the intersection of two sets.

Theory:
The intersection of two sets A and B denoted by "A ∩ B" is equal to the set of elements that are present in both the sets A and B. This operation can be represented as:

$$A \cap B = \{a \mid a \in A \text{ and } a \in B\}$$

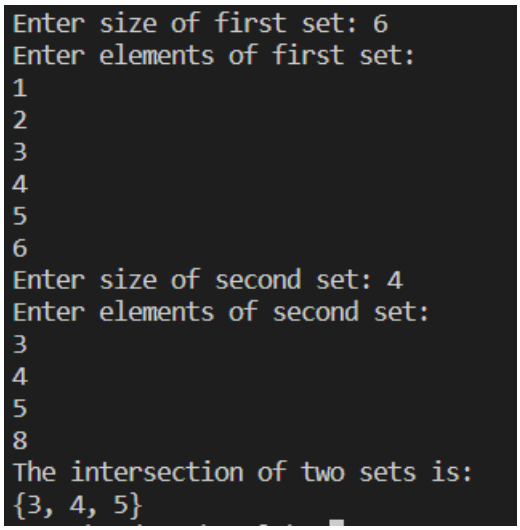
Algorithm:

1. Start,
2. Take input of two sets,
3. Store the elements present in both set in new set,
4. Display the result,
5. End,

Source code:

```
#include<iostream>

int main(){
    int n, p;
    std::cout << "Enter size of first set: ";
    std::cin >> n;
    int set1[n];
    std::cout << "Enter elements of first set:\n";
    for(int i=0; i<n; i++){ std::cin >> set1[i]; }

    std::cout << "Enter size of second set: ";
    std::cin >> p;
    int set2[p];
    std::cout << "Enter elements of second set:\n";
    for(int i=0; i<p; i++){ std::cin >> set2[i]; }

    int intersection[n], count = 0;
    for(int i=0; i<n; i++){
        for(int j=0; j<p; j++){
            if(set1[i] == set2[j]){
                intersection[count++] = set1[i];
            }
        }
    }
    std::cout << "The intersection of two sets is: \n{";
    for(int i=0; i<count; i++){
        std::cout << intersection[i];
        if(i < (count-1)){ std::cout << ", "; }
    }
    std::cout << "}";

    return 0;
}
```

Output:



Conclusion:

Using the above program, the intersection of provided two sets can be calculated.

Lab no. 17:

Objective:
To find the set difference of two sets.

Theory:
The set difference between two sets A and B denoted by “A-B” is equal to the set of elements that are present in A but not present in B. This operation can be represented as:

$$A - B = \{a \mid a \in A \text{ and } a \notin B\}$$

Algorithm:

1. Start,
2. Take two sets A and B as input,
3. Store the elements present A but not present in B in another set,
4. Display the result,
5. End,

Source code:

```
#include<iostream>

int main(){
    int n, p;

    std::cout << "Enter size set A: ";
    std::cin >> n;
    int set1[n];
    std::cout << "Enter elements of set A:\n";
    for(int i=0; i<n; i++){ std::cin >> set1[i]; }

    std::cout << "Enter size set B: ";
    std::cin >> p;
    int set2[p];
    std::cout << "Enter elements of set B:\n";
    for(int i=0; i<p; i++){ std::cin >> set2[i]; }

    int difference[n], count = 0;
    for(int i=0; i<n; i++){
        bool condition = true;
        for(int j=0; j<p; j++){
            if(set1[i] == set2[j]){
                condition = false;
                break;
            }
        }
        if(condition){
            difference[count++] =  set1[i];
        }
    }

    std::cout << "Set difference is:\nA - B = {";
    for(int i=0; i<count; i++){
        std::cout << difference[i];
        if(i < (count-1)){ std::cout << ", "; }
```

```
    }
    std::cout << "}";

    return 0;
}
```

Output:

```
Enter size set A: 5
Enter elements of set A:
1
2
3
4
5
Enter size set B: 3
Enter elements of set B:
3
5
7
Set difference is:
A - B = {1, 2, 4}
```

Conclusion:

From the above program, the set difference of given two sets can be calculated.

Lab no. 18:

Objective:
To construct the truth table of Conjunction.

Theory:
Let p and q be two propositions, the conjunction of p and q denoted by "p∧q" is a proposition, that is true when both p and q are true. "p∧q" is equivalent to "p.q".

Algorithm:

1. Start,
2. Initialize two propositions p and q, with value true and false,
3. Perform AND operation between p and q,
4. Display the result,
5. End,

Source code:

```
#include<iostream>

int main(){
    bool p[] = {false, true};
    bool q[] = {false, true};

    std::cout << "TRUTH TABLE OF CONJUNCTION: \n";
    std::cout << "p\tq\ty\n_____________________\n";

    for(int i=0; i<sizeof(p); i++){
        for(int j=0; j<sizeof(q); j++){
          std::cout << p[i] << "\t" << q[j] << "\t" << (p[i] & q[j]) << std::endl;
        }
    }

    return 0;
}
```

Output:

```
TRUTH TABLE OF CONJUNCTION:
p       q       y
___________________
0       0       0
0       1       0
1       0       0
1       1       1
```

Conclusion:
Using the above program, we can print the truth table of conjunction.

Lab no. 19:

Objective:
To construct the truth table of Disjunction.

Theory:
Let p and q be two propositions, the disjunction of p and q denoted by “pVq” is a proposition, that is false when both p and q are false. “pVq” is equivalent to “p+q”.

Algorithm:

1. Start,
2. Initialize two propositions p and q, with value true and false,
3. Perform OR operation between p and q,
4. Display the result,
5. End,

Source code:

```
#include<iostream>

int main(){
    bool p[] = {false, true};
    bool q[] = {false, true};

    std::cout << "TRUTH TABLE OF DISJUNCTION: \n";
    std::cout << "p\tq\ty\n_____________________\n";

    for(int i=0; i<sizeof(p); i++){
        for(int j=0; j<sizeof(q); j++){
          std::cout << p[i] << "\t" << q[j] << "\t" << (p[i] | q[j]) << std::endl;
        }
    }

    return 0;
}
```

Output:

```
TRUTH TABLE OF DISJUNCTION:
p       q       y
___________________
0       0       0
0       1       1
1       0       1
1       1       1
```

Conclusion:
Using the above program, we can print the truth table of disjunction.

Lab no. 20:

Objective:
To construct the truth table of Implication.

Theory:
Let p and q be two propositions, the implication of p and q denoted by “p -> q” is a proposition, that is false when p is true, and q is false. “p->q” is equivalent to “¬p V q”.

Algorithm:

1. Start,
2. Initialize two propositions p and q, with value true and false,
3. Perform OR operation between negation of p and q,
4. Display the result,
5. End,

Source code:

```
#include<iostream>

int main(){
    bool p[] = {false, true};
    bool q[] = {false, true};

    std::cout << "TRUTH TABLE OF IMPLICATION: \n";
    std::cout << "p\tq\ty\n_____________________\n";

    for(int i=0; i<sizeof(p); i++){
        for(int j=0; j<sizeof(q); j++){
         std::cout << p[i] << "\t" << q[j] << "\t" << (!p[i] | q[j]) << std::endl;
        }
    }

    return 0;
}
```

Output:

```
TRUTH TABLE OF IMPLICATION:
p       q       y
_____________________
0       0       1
0       1       1
1       0       0
1       1       1
```

Conclusion:
From the above program, the truth table of implication can be calculated.

Lab no. 21:

Objective:
To construct the truth table of Bi-implication.

Theory:
Let p and q be two propositions, the bi-implication of p and q denoted by “p <-> q” is a proposition, that is true when p->q is true and q->p is also true. “p <-> q” is equivalent to “(p->q) ∧ (q->p)”.

Algorithm:

1. Start,
2. Initialize two propositions p and q, with value true and false,
3. Perform AND operation between negation of p->q and q->p,
   (p->q => !p OR q, q->p => !q OR p)
4. Display the result,
5. End,

Source code:

```
#include<iostream>

int main(){
    bool p[] = {false, true};
    bool q[] = {false, true};

    std::cout << "TRUTH TABLE OF BI-IMPLICATION: \n";
    std::cout << "p\tq\ty\n_____________________\n";

    for(int i=0; i<sizeof(p); i++){
       for(int j=0; j<sizeof(q); j++){
        std::cout<<p[i]<<"\t"<<q[j]<<"\t"<<((!p[i]|q[j])&(!q[j]|p[i]))<<std::endl;
       }
    }

    return 0;
}
```

Output:

```
TRUTH TABLE OF BI-IMPLICATION:
p       q       y
___________________
0       0       1
0       1       0
1       0       0
1       1       1
```

Conclusion:
From the above program, the truth table of bi-implication can be displayed.

Lab no. 22:

Objective:
To check the validity of arguments using truth tables.

Theory:

Algorithm:

Source code:

Output:

Conclusion: