**LAB MANUAL**

***Lab Name………DSTL……….………. Lab Code…KCS-353…***

.



**Name……SAMARPIT DUA……………..**

**Adm.No…2019B111068….. Univ. Roll No…1900321290050……**

**Course …B.TECH.…… Branch…CEIT………..**

**Sem……3…….…… Section………A……….…..……..**



**ABES Engineering College, Ghaziabad**

**Department of Computer Science**

**List of Experiments**

|  |  |
| --- | --- |
| **S. No.** | **Experiment Name** |
| 1 | Write a program in C to create two sets and perform the Intersection operation on sets. |
| 2 | Write a program in C to create two sets and perform the Union operation on sets. |
| 3 | Write a program in C to create two sets and perform the Difference operation on sets. |
| 4 | Write a program in C to create two sets and perform the Symmetric Difference operation. |
| 5 | Write a program in C to find the complement of a set. |
| 6 | Write a C Program to find Cartesian product of two sets |
| 7 | . Write a program in C to Display the Boolean Truth Table for AND, OR, NOT |
| 8 | Write a program in C to perform the Power Set operation on a set. |
| 9 | Write a program in C for minimum cost spanning tree. |
| 10 | Write a program in C for finding shortest path in a Graph |

.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Exp. No** | **Name of the Experiment** | **Date of Practical** | **Page No. From To** | **Assessment by Faculty Member** | | | | | |
| **Schedule Actual** | **Pre-Lab Writing work**  **\* MM:05** | **Implementation/ Active Participation MM:10** | **Graph, Results/ Output, Calc.**  **MM: 05** | **In Time submission of Lab reports/ Viva Voce**  **MM:05** | **Total MM:25** | **Signature with Date** |
| 1 | Intersection of sets |  |  |  |  |  |  |  |  |
| 2 | Union of sets |  |  |  |  |  |  |  |  |
| 3 | Difference of sets |  |  |  |  |  |  |  |  |
| 4 | Symmetric difference |  |  |  |  |  |  |  |  |
| 5 | Cartesian Product |  |  |  |  |  |  |  |  |
| 6 | Complement of a set |  |  |  |  |  |  |  |  |
| 7 | Boolean Truth Table |  |  |  |  |  |  |  |  |
| 8 | Power Set |  |  |  |  |  |  |  |  |
| 9 | Minimum Spanning Tree |  |  |  |  |  |  |  |  |
| 10 | Shortest path in a graph |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **Teacher's Remarks(ifany):AverageMarks:** | | | | | | | |  |  |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**ABES Engineering College, Ghaziabad**



Student’s Name: Samarpit Dua

Roll No: 1900321290050

**I N D E X**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
|  | | | |
|  |  |  |  | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name & Sign. Of Faculty members(s) with date** |  | **Name & Sign. Of Lab In charge** |  |  |  | **HOD** |

Note: 1. \*Pre-Lab writing work should include problem statement, objective , Algorithum (if applicable)and Methodology.

2. Faculty members will check pre Lab writing work & Lab work readings/output in each class and sign with date.

3. Please use pen with blue colour Ink only.

**Roll No    ……1900321290050……. Date…12-October-2020…..Page No…1….**

**Practical Name………………**To Calculate the Intersection between two Sets**.………….… Practical No……….**1**.……….**

**PRACTICAL -1**

1. **PRACTICAL STATEMENT OF PRACTICAL:  Write a C program to create two sets and perform intersection operation on them.**
2. **OBJECTIVE OF PRACTICAL:**

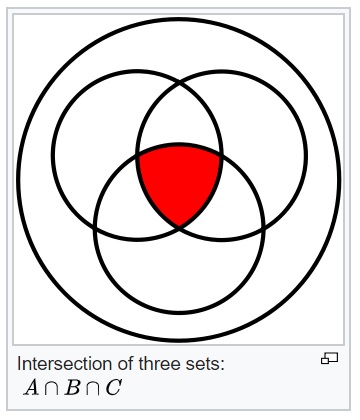
**A) Creating two sets in C language.**

**B) Performing intersection operation on two Sets.**

**3. THEORY : The intersection of two sets A and B, denoted by A ∩ B, is the set of all objects that are members of both the sets A and B. In symbols,**

**That is, x is an element of the intersection A ∩ B if and only if x is both an element of A and an element of B.**

**Intersection is an associative operation; that is, for any sets A, B, and C, one has A ∩ (B ∩ C) = (A ∩ B) ∩ C. Intersection is also commutative; for any A and B, one has A ∩ B = B ∩ A. It thus makes sense to talk about intersections of multiple sets. The intersection of A, B, C, and D, for example, is unambiguously written A ∩ B ∩ C ∩ D.**

****

1. **ALGORITHM / FLOW CHART:**

**Algorithm for this is as follows :**

**1. Take three arrays.**

**2. Enter the size of two arrays.**

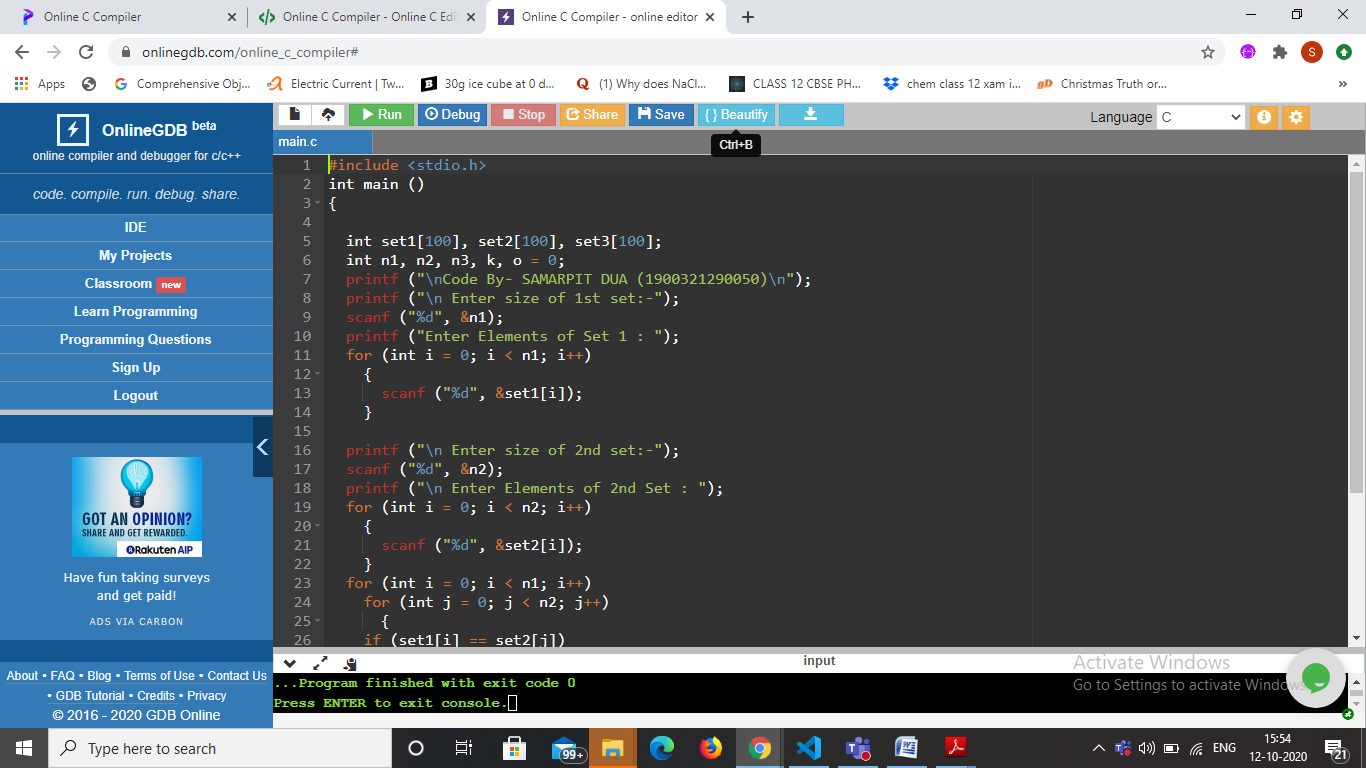
**3. Put values in two arrays.**

**4. Check for common value in both the arrays using for loop.**

**5. Store the same element in third array.**

**6. Print the elements of third array.**

1. **IMPLEMENTATION :**

****

1. **Result /Output :**
2. ****



**PRACTICAL -2**

1. **PRACTICAL STATEMENT OF PRACTICAL: Write a program in C to create two sets and perform the Union operation on two sets.**
2. **OBJECTIVE OF PRACTICAL : Create two arrays with same or variable size.Perform union operation on them and store the result in third array.**
3. **Theory : The union of two sets *A* and *B* is the set of elements which are in *A*, in *B*, or in both *A* and *B*. In symbols. For example, if *A* = {1, 3, 5, 7} and *B* = {1, 2, 4, 6} then *A* ∪ *B* = {1, 2, 3, 4, 5, 6, 7}.**
4. **ALGORITHM / FLOW CHART :**

1) Initialize union U as empty.

2) Copy all elements of first array to U .

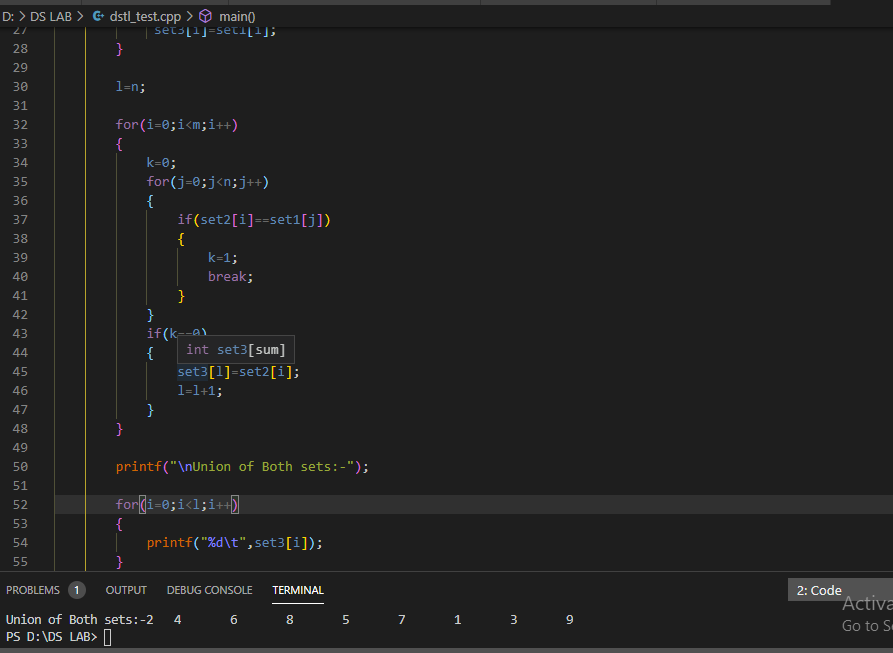
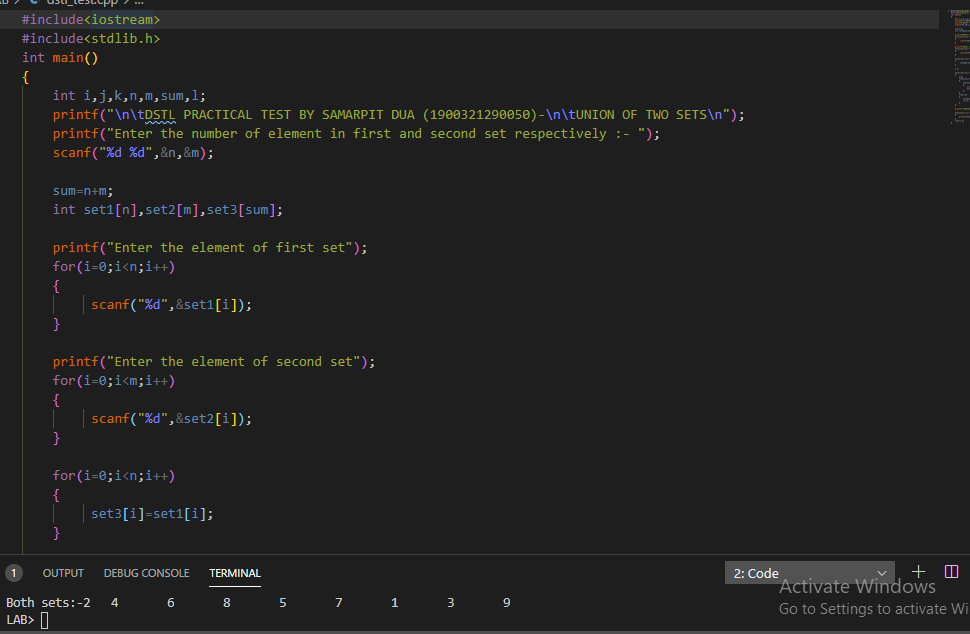
3) Do following for every element x of second array:

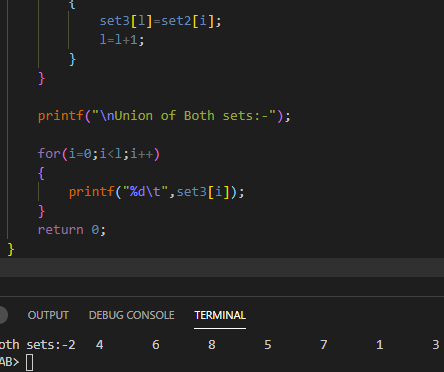
a) If x is not present in first array, then copy x to U.

b) else continue the loop and leave that element.

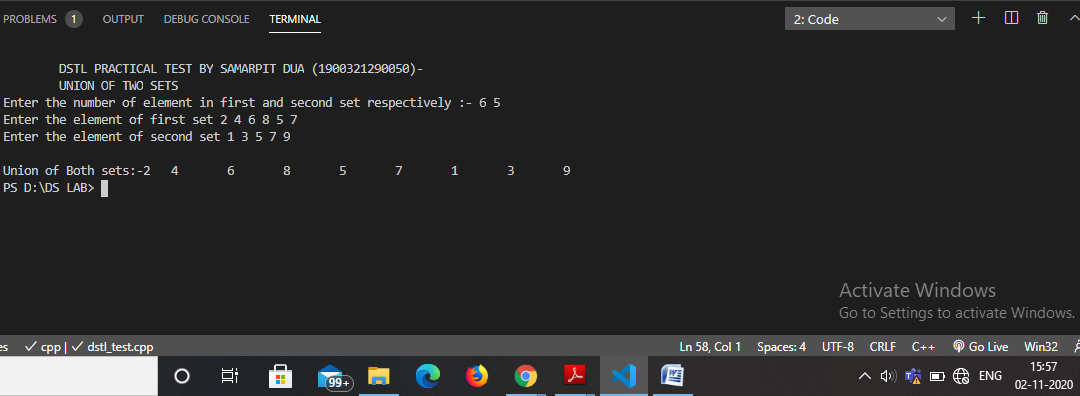
4) Return U.

1. **IMPLEMENTATION:**

****

****

**Result /Output:**



**PRACTICAL-3**

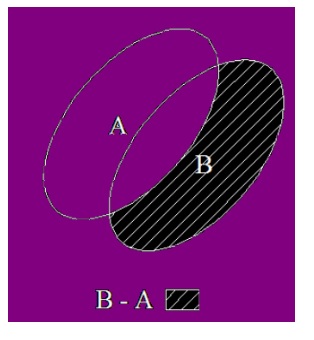
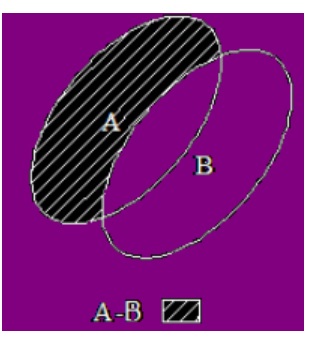
1. **PRACTICAL STATEMENT OF PRACTICAL: Write a program in C to create two sets and perform the Difference operation on sets.**

**2 . OBJECTIVE OF PRACTICAL : Write C program to obtain difference of two sets.**

**3 . THEORY: Difference of sets is defined as a method of rearranging sets by removing the elements which belong to another set. Difference of sets is denoted by either by the symbols - or \. P minus Q can be written either P - Q or P \ Q.**

**The differences of two sets P and Q, is written as P - Q, It contains elements of P which are not present in elements of Q. Here, result P - Q is obtained. Take set P as usual and compare with set Q. Now, remove that element in set P which matches with set Q. If P = {a, b, c, d} and Q = {d, e}, then P - Q = {a, b, c}. The difference between two sets A and B are represented in the order as the set of all those elements of A which are not in B. It is denoted by A - B.In symbol, we write it as**

**A - B = {x: x ∈ A and x ∉ B}  
Similarly, B - A = {x: x ∈ B and ∉ A}.**

** **

1. **ALGORITHM / FLOW CHART :**

BEGIN:

Input: Set A, Set B;

Create Empty Set C to contain (A-B).

for each element a in Set A:

if a does not exist in Set B:

Add a to Set C;

Return Set C;

END;

**IMPLEMENTATION :**

#include<stdio.h>

int main()

{

    int i,j,k,n,m,o,l=0;

    printf("\n\tCODE BY SAMARPIT DUA 1900321290050:-\n");

    printf("\tDIFFERENCE OF TWO SETS\n");

    printf("Enter the number of element in first and second set respectively :- ");

    scanf("%d%d",&n,&m);

    o=n+m;

    int set1[n],set2[m],set3[o];

    printf("Enter the element of first set");

    for(i=0;i<n;i++)

    {

        scanf("%d",&set1[i]);

    }

    printf("Enter the element of second set");

    for(i=0;i<m;i++)

    {

        scanf("%d",&set2[i]);

    }

    for(i=0;i<m;i++)

    {

        k=0;

        for(j=0;j<n;j++)

        {

            if(set1[i]==set2[j])

            {

                k=1;

                break;

            }

        }

        if(k==0)

        {

            set3[l]=set1[i];

            l=l+1;

        }

    }

    printf("\nDifference (SET1 and SET2):-");

    for(i=0;i<l;i++)

    {

        printf("%d\t",set3[i]);

    }

    k=0,l=0;

    printf("\nDifference between SET2 and SET1 is : ");

    for(i=0;i<n;i++)

    {

        k=0;

        for(j=0;j<m;j++)

        {

            if(set1[j]==set2[i])

            {

                k=1;

                break;

            }

        }

        if(k==0)

        {

            set3[l]=set2[i];

            l=l+1;

        }

    }

    for(i=0;i<l;i++)

    {

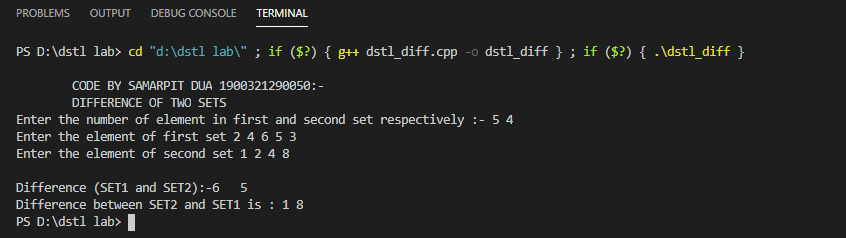
        printf("%d\t",set3[i]);

    }

    return 0;

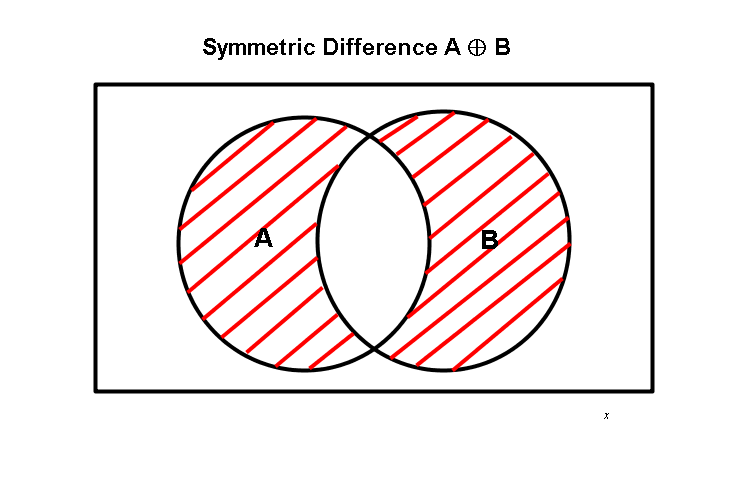
}

1. Result /Output:



**PRACTICAL -4**

1. **PRACTICAL STATEMENT OF PRACTICAL:** Write a program in C to create two sets and perform the Symmetric Difference operation.
2. **THEORY:**The **symmetric difference of two sets** A and B is the **set** (A – B) ∪ (B – A) and is denoted by A △ B. The shaded part of the given Venn diagram represents A △ B. A △ B is the **set** of all those elements which belongs either to A or to B but not to **both**.



1. **IMPLEMENTATION :**

#include <stdio.h>

void symm\_diff(int \*, int \*, int, int);

int main()

{

    int n1, n2, i, j;

    printf("\n\t CODE BY SAMARPIT DUA (1900321290050)\n\tSYMMETRIC DIFFERENCE BETWEEN TWO SETS\n");

    printf("\nEnter the total number of elements in first and second set : ");

    scanf("%d %d", &n1, &n2);

    int set1[n1], set2[n2];

   //Taking Input of Elements of set

    printf("\nEnter the elements in first array");

    for (i = 0; i < n1; i++)

    {

        scanf("%d", &set1[i]);

    }

    printf("\nEnter the elements in second array");

    for (j = 0; j < n2; j++)

    {

        scanf(" %d", &set2[j]);

    }

    symm\_diff(set1,set2, n1, n2);

    return 0;

}

void symm\_diff(int \*a, int \*b, int n1, int n2)

{

    int i, j, k = 0, o, l = 0, m = 0, flag;

    int c[n1 + n2], d[n1 + n2], s[n1 + n2];

    //Calaculating set1-set2

    for (i = 0; i < n1; i++)

    {

        c[k++] = a[i];

    }

    for (j = 0; j < n2; j++)

    {

        o = 0;

        for (i = 0; i < n1; i++)

        {

            if (a[i] == b[j])

            {

                o = 1;

                break;

            }

        }

        if (o == 0)

        {

            c[k++] = b[j];

        }

    }

    //Calculating set2-set1

    for (i = 0; i < n1; i++)

    {

        for (j = 0; j < n2; j++)

        {

            if (a[i] == b[j])

            {

                d[l++] = a[i];

            }

        }

    }

    for (i = 0; i < k; i++)

    {

        for (j = 0; j < l; j++)

        {

            if (c[i] == d[j])

            {

                flag = 1;

                break;

            }

            else

            {

                flag = 0;

            }

        }

        if (flag == 1)

        {

            continue;

        }

        else

        {

            s[m++] = c[i];

        }

    }

    printf("\nSymmetric Difference of sets is\n");

    for (i = 0; i < m; i++)

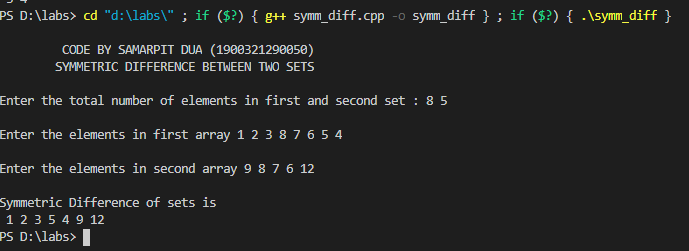
    {

        printf(" %d", s[i]);

    }

}

**Result /Output:**

****

**PRACTICAL -5**

**PRACTICAL STATEMENT OF PRACTICAL: Write a C Program to find Cartesian product of two sets.**

**THEORY : *Ordered Pair:***  An *ordered pair* is a pair of objects where one element is designated first and the other element is designated second, denoted (*a*, *b*).

***Cartesian Product:*** The *Cartesian product* of two sets *A* and *B*, denoted *A* × *B*, is the set of all possible ordered pairs where the elements of *A* are first and the elements of *B*are second.

In set-builder notation,  *A* × *B* = {(*a, b*) : *a* ∈ *A* and *b* ∈ *B*}.

*Example:*  Let *A* = {H, T} and *B* = {1, 2, 3, 4, 5, 6}.

*A* × *B* = {(H, 1), (H, 2), (H, 3), (H, 4), (H, 5), (H, 6), (T, 1), (T, 2), (T, 3), (T, 4), (T, 5), (T, 6)}

*B* × *A*= {(1, H), (2, H), (3, H), (4, H), (5, H), (6, H), (1, T), (2, T), (3, T), (4, T), (5, T), (6, T)}

Note that in this case *A* × *B*≠ *B* × *A*, i.e., the Cartesian product is not commutative.  
Also, note that *n*(*A*) ∙ *n*(*B*) = 2(6) = 12 = *n*(*A* × *B*).

**IMPLEMENTATION :**

#include<stdio.h>

int main()

{

    printf("\nCODE BY SAMARPIT DUA (1900321290050)");

    int m,n;

    int a[1000],b[1000];

    printf("\nEnter no. of elements in set A:-");

    scanf("%d",&m);

    printf("\n Enter elements of set A:- ");

    for(int i=0;i<m;i++)

    scanf("%d",&a[i]);

    printf("\nEnter no. of elements in set B:-");

    scanf("%d",&n);

    printf("\n Enter elements of set B:- ");

    for(int i=0;i<n;i++)

    scanf("%d",&b[i]);

    printf("\nCartesion product:\n");

    for(int i=0;i<m;i++)

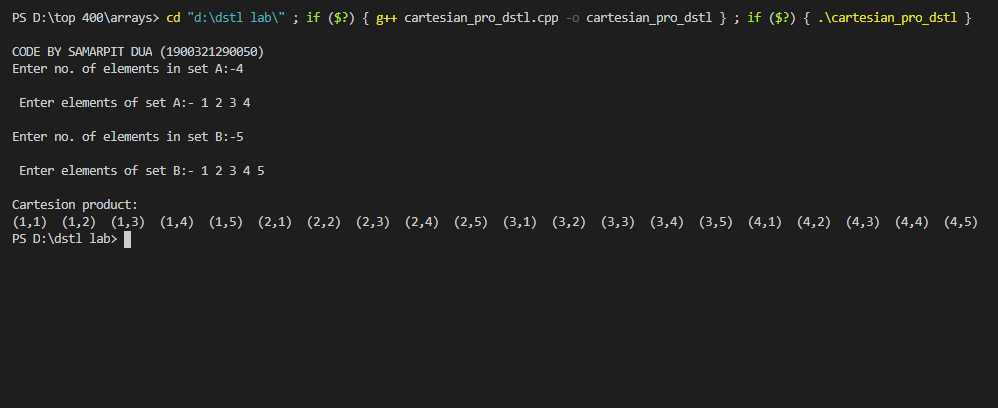
    for(int j=0;j<n;j++)

    printf("(%d,%d)  ",a[i],b[j]);

    return 0;

}

1. **Result /Output:**

****

**PRACTICAL-6**

1. **PRACTICAL STATEMENT OF PRACTICAL:**

Write a program in C to find the complement of a set.

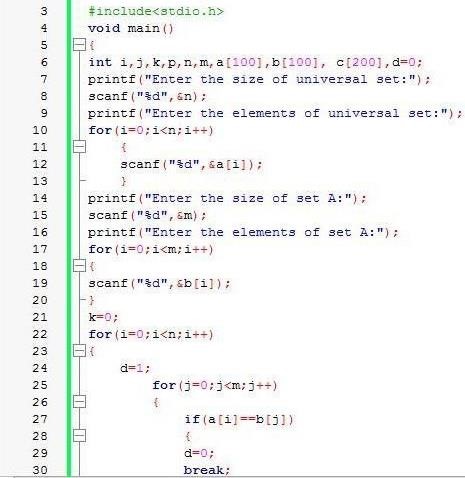
1. **OBJECTIVE OF PRACTICAL**

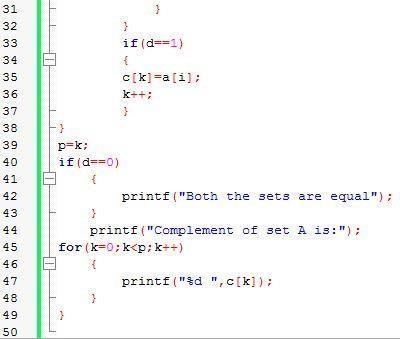
To find the complement of a set.

## 3.ALGORITHM / FLOW CHART

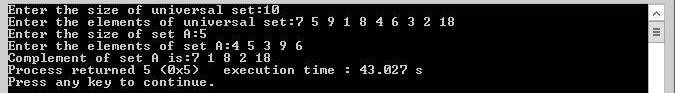
|  |
| --- |
| Input : arr1[] = {3, 6, 10, 12, 15} arr2[] = {1, 3, 5, 10, 16}  Output : 6 12 15  The elements 6, 12 and 15 are present in arr[], but not present in arr2[]  Input : arr1[] = {10, 20, 36, 59} arr2[] = {5, 10, 15, 59}  Output : 20 36 |
|  |

## 4.IMPLEMENTATION





## 5.RESULT /OUTPUT



**PRACTICAL -7**

1. **PRACTICAL STATEMENT OF PRACTICAL: Write a program in C to Display the Boolean**

**Truth Table for AND, OR, NOT gates.**

1. **OBJECTIVE OF PRACTICAL :**  **Write a program in C to Display the Boolean**

**Truth Table for AND, OR, NOT gates.**

1. **THEORY :** Digital systems are said to be constructed by using logic gates. These gates are the AND, OR,NOT, NAND, NOR, EXOR and EXNOR gates. The basic operations are described below with

the aid of truth tables.

**AND gate**- The AND gate is an electronic circuit that gives a high output (1) only if all its inputs

are high. A dot (.) is used to show the AND operation i.e. A.B. Bear in mind that this dot is

sometimes omitted i.e. AB.

**OR gate**- The OR gate is an electronic circuit that gives a high output (1) if one or more of its

inputs are high. A plus (+) is used to show the OR operation.

**NOT gate**- The NOT gate is an electronic circuit that produces an inverted version of the input

at its output. It is also known as an inverter. If the input variable is A, the inverted output is

known as NOT A. This is also shown as A', or A with a bar over the top, as shown at the

outputs.

1. **IMPLEMENTATION:**

#include <stdio.h>

void main()

{

    int a[2][2], b[2][2], c[2];

    int i, j;

    for (i = 0; i <= 1; i++)

    {

        for (j = 0; j <= 1; j++)

        {

            a[i][j] = (i && j);

            b[i][j] = (i || j);

        }

    }

    for (i = 0; i <= 1; i++)

    {

        c[i] = (!i);

    }

    printf("\n\tSAMARPIT DUA 1900321290050\t ");

    printf("\nThe Truth Table for AND Gate( && ) is:\n");

    printf(" A B : (C=A&&B)\n");

    for (i = 0; i <= 1; i++)

    {

        for (j = 0; j <= 1; j++)

        {

            printf(" %d %d : %d\n", i, j, a[i][j]);

        }

    }

    printf("\nThe Truth Table for OR Gate( || ) is:\n");

    printf(" A B : (C=A||B)\n");

    for (i = 0; i <= 1; i++)

    {

        for (j = 0; j <= 1; j++)

        {

            printf(" %d %d : %d\n", i, j, b[i][j]);

        }

    }

    printf("\nThe Truth Table for NOT Gate (!) is:\n");

    printf(" A : (B = !A)\n");

    for (i = 0; i <= 1; i++)

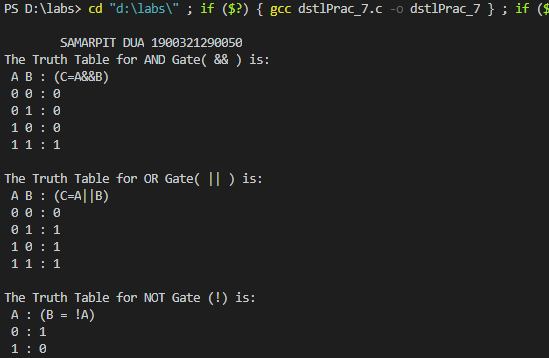
    {

        printf(" %d : %d\n", i, c[i]);

    }

}

1. **Result /Output:**

****

**PRACTICAL -8**

**1. PRACTICAL STATEMENT OF PRACTICAL:**

Write a program in C to perform the Power Set operation on a sets.

**2.OBJECTIVE OF PRACTICAL :**

The power set of any set S is the set of all subsets of S, including the empty set and S itself, variously denoted as P(S), identifying the power set of S with the set of all functions from S to a given set of two elements, 2S.

**3. ALGORITHM / FLOW CHART**:

Input: Set[], set\_size

1. Get the size of power set

powet\_set\_size = pow(2, set\_size)

2 Loop for counter from 0 to pow\_set\_size

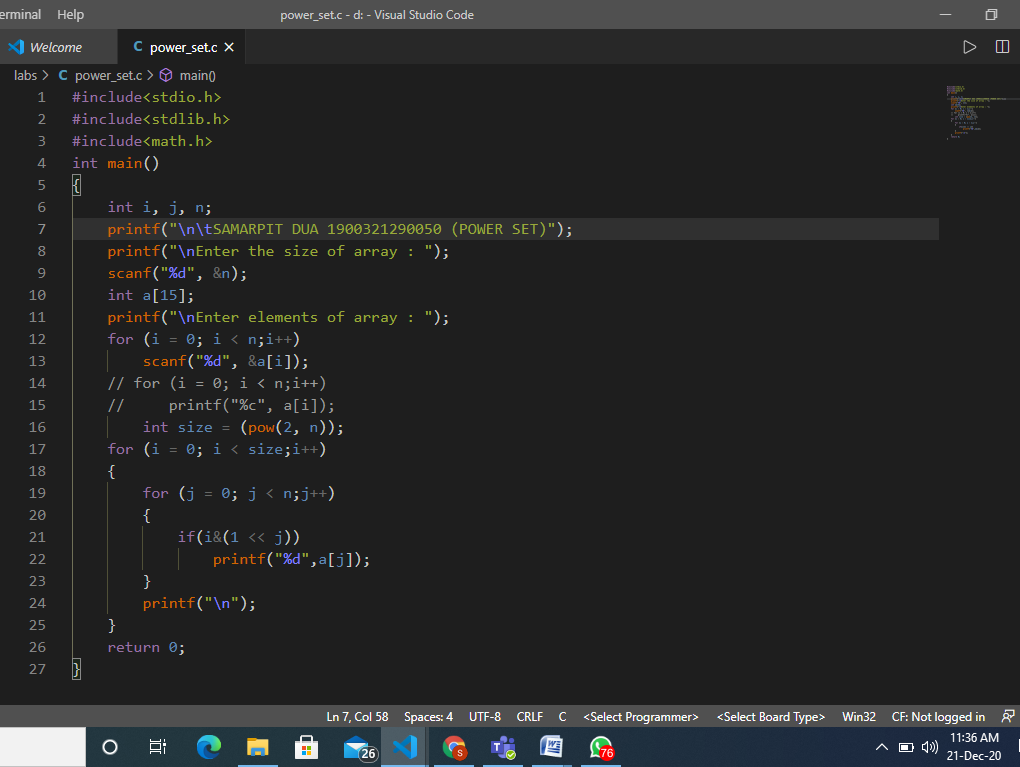
(a) Loop for i = 0 to set\_size

(i) If ith bit in counter is set

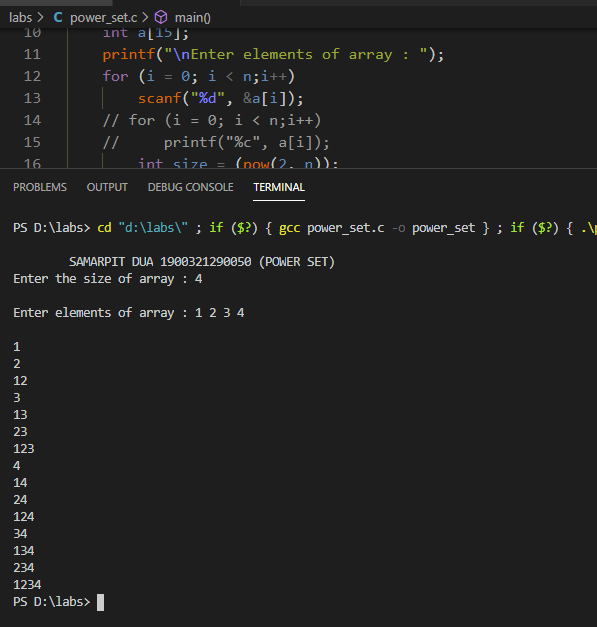
Print ith element from set for this subset

(b) Print seperator for subsets i.e., newline

**4. IMPLEMENTATION:**

****

**5.RESULT/OUTPUT:**

****

**Practical-9**

1. **PRACTICAL STATEMENT OF PRACTICAL:** **Write a program in C for minimum cost spanning tree.**
2. **THEORY : Prim's Algorithm is used to find the minimum spanning tree from a graph. Prim's algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.Prim's algorithm starts with the single node and explore all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.**
3. **ALGORITHM / FLOW CHART :**

**1) Create a set *mstSet* that keeps track of vertices already included in MST.**

**2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.**

**3) While mstSet doesn’t include all vertices   
….a) Pick a vertex *u* which is not there in *mstSet*and has minimum key value.**

**….b) Include *u*to mstSet.**

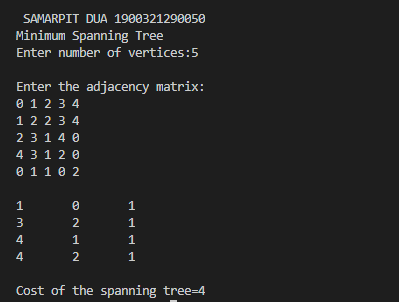
**….c) Update key value of all adjacent vertices of *u*. To update the key values, iterate through**

**All adjacent vertices. For every adjacent vertex *v*, if weight of edge *u-v* is less than the previous key value of *v*, update the key value as weight of *u-v***

**The idea of using key values is to pick the minimum weight edge from**[**cut**](http://en.wikipedia.org/wiki/Cut_(graph_theory))**. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.**

1. **SOURCE CODE:**
2. #include<stdio.h>
3. #define MAX 30
4. typedef struct edge
5. {
6. int u,v,w;
7. }edge;
8. typedef struct edgelist
9. {
10. edge data[MAX];
11. int n;
12. }edgelist;
13. edgelist elist;
14. int G[MAX][MAX],n;
15. edgelist spanlist;
16. void kruskal();
17. int find(int belongs[],int vertexno);
18. void union1(int belongs[],int c1,int c2);
19. void sort();
20. void print();
21. void main()
22. {
23. int i,j,total\_cost;
24. printf("\n SAMARPIT DUA 1900321290050 \nMinimum Spanning Tree ");
25. printf("\nEnter number of vertices:");
26. scanf("%d",&n);
27. printf("\nEnter the adjacency matrix:\n");
28. for(i=0;i<n;i++)
29. for(j=0;j<n;j++)
30. scanf("%d",&G[i][j]);
31. kruskal();
32. print();
33. }
34. void kruskal()
35. {
36. int belongs[MAX],i,j,cno1,cno2;
37. elist.n=0;
38. for(i=1;i<n;i++)
39. for(j=0;j<i;j++)
40. {
41. if(G[i][j]!=0)
42. {
43. elist.data[elist.n].u=i;
44. elist.data[elist.n].v=j;
45. elist.data[elist.n].w=G[i][j];
46. elist.n++;
47. }
48. }
49. sort();
50. for(i=0;i<n;i++)
51. belongs[i]=i;
52. spanlist.n=0;
53. for(i=0;i<elist.n;i++)
54. {
55. cno1=find(belongs,elist.data[i].u);
56. cno2=find(belongs,elist.data[i].v);
57. if(cno1!=cno2)
58. {
59. spanlist.data[spanlist.n]=elist.data[i];
60. spanlist.n=spanlist.n+1;
61. union1(belongs,cno1,cno2);
62. }
63. }
64. }
65. int find(int belongs[],int vertexno)
66. {
67. return(belongs[vertexno]);
68. }
69. void union1(int belongs[],int c1,int c2)
70. {
71. int i;
72. for(i=0;i<n;i++)
73. if(belongs[i]==c2)
74. belongs[i]=c1;
75. }
76. void sort()
77. {
78. int i,j;
79. edge temp;
80. for(i=1;i<elist.n;i++)
81. for(j=0;j<elist.n-1;j++)
82. if(elist.data[j].w>elist.data[j+1].w)
83. {
84. temp=elist.data[j];
85. elist.data[j]=elist.data[j+1];
86. elist.data[j+1]=temp;
87. }
88. }
89. void print()
90. {
91. int i,cost=0;
92. for(i=0;i<spanlist.n;i++)
93. {
94. printf("\n%d\t%d\t%d",spanlist.data[i].u,spanlist.data[i].v,spanlist.data[i].w);
95. cost=cost+spanlist.data[i].w;
96. }
97. printf("\n\nCost of the spanning tree=%d",cost);
98. }

**OUTPUT:**

****

**PRACTICAL - 10**

OBJECTIVE: Write a program in C for finding shortest path in a graph.

THEORY:One algorithm for finding the shortest path from a starting node to a target node in a weighted graph is Dijkstra’s algorithm. The algorithm creates a tree of shortest paths from the starting vertex, the source, to all other points in the graph.

ALGORITHM:

**Dijkstra’s Algorithm:**

1. Create cost matrix C[ ][ ] from adjacency matrix adj[ ][ ]. C[i][j] is the cost of going from vertex i to vertex j. If there is no edge between vertices i and j then C[i][j] is infinity.

2. Array visited[ ] is initialized to zero.

for(i=0;i<n;i++)

visited[i]=0;

3. If the vertex 0 is the source vertex then visited[0] is marked as 1.

4. Create the distance matrix, by storing the cost of vertices from vertex no. 0 to n-1 from the source vertex 0.

for(i=1;i<n;i++)

distance[i]=cost[0][i];

Initially, distance of source vertex is taken as 0. i.e. distance[0]=0;

5. for(i=1;i<n;i++)

Choose a vertex w, such that distance[w] is minimum and visited[w] is 0. Mark visited[w] as 1.

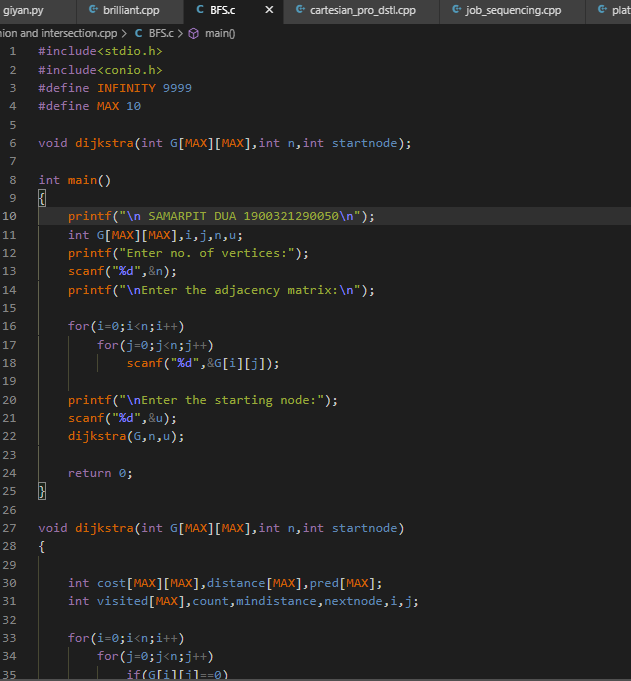
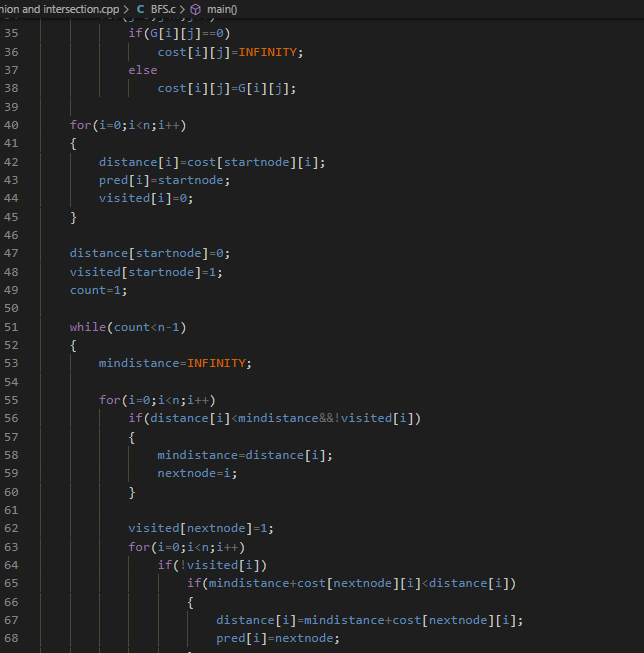
Recalculate the shortest distance of remaining vertices from the source.

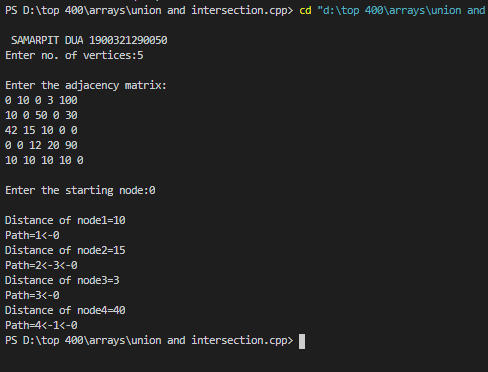
Only, the vertices not marked as 1 in array visited[ ] should be considered for recalculation of distance. i.e. for each vertex v

if(visited[v]==0)

distance[v]=min(distance[v],

distance[w]+cost[w][v])

1. **IMPLEMENTATION:**
2. ****
3. ****
4. ****
5. **Result /Output:**

****