1.

// set union, intersection, difference, cartesian product.

```
#include <stdio.h>
int a[5], b[5];
printf("Enter 5 number of set A: \n");
for (int i=0; i<5; i++)
  {
   scanf("%d", &a[i]);
  }
// union
int count_union = 10;
int count_intersection = 0;
for (int i=0; i<5; i++)
  {
    for (int j=0; j<5; j++)
    {
      if a[i] == b[j]
          count_intersection ++;
    }
  }
}
int intersection [count_intersection];
int k=0;
for (int i=0; i<5; i++) {
    for (int j=0; j<5; j++) {
```

```
        if (a[i] == b[j]) {
            intersection[k] = a[i];
            k++;
        }
    }
}
int count_diff = 5 - count_intersection;
int diffA[count_diff];
int diffB[count_diff];
int x = 0, y = 0;
for (int i = 0; i < 5; i++) {
    int countda = 0;
    for (int j = 0; j < count_intersection; j++)
    {
        if (a[i] != intersection[j])
            countda++;
    }
if (countda == count_intersection)
{
    diffA[x] = a[i];
    x++;
}
for (int i = 0; j < 5; i++) {
    int countdB = 0;
    for (int j = 0; j < count_intersection; j++) {
        if (b[i] != intersection[j])
```

```
for (int i=0; i<count_intersection; i++) {
    printf("%d,", intersection[i]);
}
printf("\n DiffA: ");
for(int i=0; i<count_diff;;i++)
  {
  for (int i=0; i<count_diff; i++)
    printf("%d, ", diffA[i]);
}
printf("DiffB: ");
for (int i=0; i<count_diff; i++)
    printf("%d, ", diffB[i]);
printf("\n union: ");
for (int i=0; i<z; i++)
 {
  printf("%d, ", Union[i]);
 }
printf("Cartesian Product: ");
for(int i=0; i<5; i++) {
    for (int j=0; j<5; j++) {
        printf("(%d, %d); ", a[i], b[j]);
    }
}
```

```
            countdb ++;
        }
        if (countdb == count_intersection)
        {
            diffB[j] = b[i];
            j++;
        }
    }
int z = count_diff + count_diff +
        count_intersection;
int Union[z];
for (int i=0; i<count_diff; i++)
    union[i] = diffA[i];
int l=0;
for (int i=count_diff; i<(count_diff + count_diff);
    i++)
{
 union[i] = diffB[l];
  l++;
}
int ll = 0;
for (int i=(count_diff + count_diff); i<z; i++)
  {
   Union[i] = intersection[ll];
   ll++;
  }
  printf("Intersection: ");
```

```
printf("\n Cartesian product : ");
for (int i=0; i<5; i++) {
    for (int j = 0; j<5; j++)
     {
      printf("(%d, %d); ", b[i], a[j]);
    }
}
return 0;
}
```

2.

// ceiling and floor function

```
#include <stdio.h>
int ceiling (float b){
    if (b >0) {
        int temp = b* 100;
        if ((temp %100) == 0){
            int c=b;
            return c;
        }
        else
        {
        b = b + 1;
        int c=b;
        return c;
    }
}
else if (b<0)
    {
      int c=b;
      return c;
}
```

```
    else
    {
    return 0;
  }
}
int Floor (float k)
  {
  if (k>=0)
   {
    int c=k;
     return c;
   }
   else if (k<0)
      {
      int temp = k*100;
     if ((temp %100) ==0)
       {
       int c= k;
       return c;
      }
      else
      {
       k =k-1;
      int c= k;
      return c;
```

```
    }
    else
    {
     return 0;
    }
}
int main()
  {
  int ce, fl;
  float b, c;
  printf("Enter value: ");
  scanf("%f", &b);
  ce=Ceiling(b);
  fl=Floor(b);
  printf("Ceiling is %d \n Floor is %d", ce, fl);
  return 0;
   }
```

```
// fuzzy set and its implementation
#include <stdio.h>
struct fuzzy
  {
  char label;
  float num;
  };
  int main()
  {
   int n, i, j;
   printf(" Enter elements : ");
   scanf (" %d", &n);
   struct fuzzy setA[n];
   struct fuzzy setB[n];
   struct fuzzy unions[n];
   struct fuzzy intersection[n];
   printf (" (label, value) => (x,0.12) \n");
```

```
printf("\n give data of set A : \n");
for (i=0; i<n; i++)
  {
   fflush (stdin);
   printf(" Enter label: ");
   scanf(" %c", &setA[i].label);
   printf (" Enter value: ");
   scanf (" %f", &setA[i].num);
}
printf("Now, give data of set B: ");
for (i=0; i<n; i++)
{
  fflush (stdin);
  printf (" Enter label:");
scanf( " %c", &setB[i].label);
printf (" Enter value: ");
scanf (" %f", &setB[i].num);
}
```

```
for (i=0; i<n; i++)
{
 for (j=0; j<n; j++)
  {
  if (setA[i].num > setB[i].num)
   {
   unions[i].num = setA[i].num;
   unions[i].label = setA[i].label;
   break;
   }
  else
   {
    unions[i].num = setB[i].num;
    unions[i].label = setB[i].label;
    break;
   }
  }
 }
 for(i=0; i<n; i++)
```

```
{
if (setA[i].num > setB[i].num)
{
intersection[i].num = setB[i].num;
intersection[i].label = setB[i].label;
break;
}
else {
intersection[i].num = setA[i].num;
intersection[i].label = setA[i].label;
break;
}
}
printf("\n union: \n");
for (i=0; i<n; i++)
{
printf("(%c, %.2f)", unions[i].label,
unions[i].num);
```

```
}
printf("\n intersection: \n");
for (i=0; i<n; i++)
{
printf("(%c, %.2f)", intersection[i].label,
   intersection[i].num);
}
printf("\n Complement of fuzzy set A: \n");
for (i=0; i<n; i++)
  {
  printf("(%c, %.2f)", setA[i].label,
  (1-setA[i].num));
}
printf("\n Complement of set B:)
for (i=0; i<n; i++)
  {
  printf("(%c, %.2f)", setB[i].label,
  (1-setB[i].num));
```

```
}
return 0;
}
```

4.

```
// euclidian algorithm
#include <stdio.h>
int GCD(int a, int b)
 {
  int c= a % b;
  if (c!=0)
     {
     a = b;
     a = c;
     GCD(a,b);
   }
  else
  {
   return b;
   }
 }
int extended_gcd (int a, int b, int *x, int *y)
  {
```

```
if (a==0)
{
  *x=0;
  *y=1;
  return b;
}
int -x, -y;
int gcd = extended_gcd(b%a, a, &-x, &-y);
*x = -y -(b/a)*-x;
*y = -x;
return gcd;
}
int main()
  {
  int x,y;
  int a,b;
  printf("Enter first :In =>");
  scanf("%d", &a);
  printf("Enter second digit: In =>");
  scanf("%d", &b);
```


TU
Bhaktapur Multipal Campus
CHECKED

```
if (a>b)
{
int temp= GCD(a,b);
printf(" The GCD by Euclidian is %d.", temp);
 }
else
 {
 int temp= GCD (b,a);
 printf (" GCD is %d.", temp);
 }
 printf (" gcd = %d \n",
 extended_gcd (a, b, &x, &y));
 printf (" x = %d, y= %d", x, y);
 return 0;
}
```

6.

```
#include<stdio.h>
int main()
{
 int a, b;
printf("Enter numbe of rows: ");
scanf ("%d", &a);
printf ("Enter number of columns: ");
scanf (" %d", &b);
int A[a][b], B[a][b];
printf("Enter first boolean matrix: ");
 for (int i=0; i<a; i++)
    {
    for (int j=0; j<b; j++)
    {
       scanf( "%d", &B[i][j]);
    }
}
```

```
    int join[a][b];
  for (int i=0; i<a; i++)
    {
    for (int j=0; j<b; j++)
    {
     if ((A[i][j]==1) && (B[i][j]==1)) {
           join[i][j]=1;
     }
     else
     {
     join[i][j]=(A[i][j]+B[i][j]);
     }
    }
  }
}
int meet[a][b];
  for (int i=0; i<a; i++)
    {
    for (int j=0; j<b; j++)
      {
```

```
    meet[i][j]=(A[i][j] * B[i][j]);
    }
}
int product[a][b];
for (int i=0; i<a; i++)
{
for (int j=0; j<b; j++)
  {
   product[i][j]=0;
   for (int k=0; k<a; k++)
   {
int pp= product[i][j]+A[i][k]* B[k][j];
if ((pp!=1) && (pp!=0))
  {
   pp=1;
  }
product[i][j]=pp;
```

```
    }
  }
 }
printf("\njion:");
for (int i=0; i<a; i++)
{
  printf("[");
for (int j=0; j<b; j++)
  {
  printf("%d", join[i][j]);
  }
printf("]\n");
}
printf("\nMeet:");
for (int i=0; i<a; i++)
{
  printf("[");
for (int j=0; j<b; j++)
```

```
    {
    printf("%d", join[i][j]);
   }
   printf("]\n");
  }
printf("\nMeet:\n");
   for (int i=0; i<a; i++)
   {
    printf("[");
for (int j=0; j<b; j++)
  {
  printf("%d", meet[i][j]);

  }
  printf("]\n");
}
printf("Boolean product:\n");
for (int i=0; i<a; i++)
 {
```

```
        printf("[");
    for (int j = 0; j < b; j++)
    {
     printf("%d", product[i][j]);
    }
    printf("]\n");

}
return 0;

}
```

8.

// generate truth table of compound preposition

```
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <conio.h>
int or2(int p, int z)
  {
   if ((p==0) && (z==0))
      {
       return 0;
      }
   else
   {
    return 1;
   }
}
```

```
int and2(int p, int q)
 {
  if ((p==1) && (q==1))
   {
    return 1;
   }
  else
  {
   return 0;
  }
}
int nor (int p)
 {
  if (p==0)
   {
    return 1;
   }
  else
  {
   return 0;
  }
```

```
int conditional2(int p, int z)
 {
  if ((p==1) && (z==0))
  {
   return 0;
  }
  else
  {
  return 1;
  }
}
int bi-conditional2(int p, int z)
 {
  if ((p==z)) {
     return 1;
  }
  else
  {
   return 0;
}
```

```
}
int main()
{
int n, k;
char opr[3];
printf("Enter number of variable: ");
scanf("%d", &n);
printf("Total operation: ");
scanf("%d", &k);
if(n==2)
{
int table[4][2]=
  {
    {0,0},
    {0,1},
    {1,0},
    {1,1}
};
```

```
    int result [4] [K];
  }
else
{
  int table [8] [3] =
  {
    { 0,0,0},
    { 0,0, 1},
    { 0,1,0},
    { 1,1,0},
  { 1, 0,1}
   { 1,1,1}
};
 int result [8] [K];
}
printf (" OR: \n AND \n NOR: - In conditional:
        > \n B - Conditional < \n");
```

```
printf("First operation and operand (o,v,r):");
//~p
scanf("%s", opr);
int z[3], flg=1;
int var1, var2, count=0;
for (int i=0; i<3; i++)
 {
  z[i] = opr[i];
 if ((z[i] == '+'))
 {
  flg=1;
 }
if ((z[i]=='-'))
 {
 flag1=2;

 }
if ((z[i]=='*'))
 {
 flg 1=13;
```

```
}
if ((z[i] == 60))
{
 flag 1 = 15;
}
if (opr [i] == 'r')
 {
   if (count == 1)
   {
    var 2 = 3;
}
else
{
var 1 = 3;
}
count++;
}
}
```

```
int ((flg ] == ] ) && ){
    for (int i=0; i<8; i++)
    {
    result [i] [ ]
    }
}
return 0;
}
```

10.

```
// Compute a^n, b^n, mod m, linear
search by recursion.

#include <stdio.h>
int i=0;
int recur(int a, int n)
  {
  if(n==1)
  {
  return a;
 }
 else
  {
  a= a* (recur(a,(n-1)));
  }
}
int exponentMod(int A, int B, int C)
 {
```

```
if (A==0)
    return 0;
if (B==0)
    return 1;

int y;
if (B%2==0)
{
y= exponentMod(A,B/2,C);
y= (y*y)%C;
}
else
{
y= A%C;
y= (y* exponentMod(A,B-1,C)%C)%C;
}
return (int)((y+C)%C);
}
int linear(int a[], int size, int target)
```

```
{
if(a[i]==target)
{
return 100;
}
else
{
i++;
linear(a, size-1, target);
}
if (size==0)
 {
  return 10;
 }
}
int main()
 {
 int k= recur(4,4);
 printf("%d", k);
 int a[5]= {1,2,3,4,5};
```

```
int p = linear(2,5,9);
if (p == 100)
{
printf("found");
else
{
printf("Not found");
}
int mod = exponentmod(2,90,13);
printf("mod : %d", mod);
return 0;
}
```

// generate permutation and combination

```
#include <stdio.h>
int factorial (int n)
 {
  if(n == 1)
    {
    return n;
  }
  else
  {
  n = n* factorial(n-1);
  }
}
int main ()
  {
  int a, b;
printf("Enter number");
```

```
scanf ("%d", &a);
printf ("Enter number of object:");
scanf ("%d", &b);
int fa = factorial (a);
int fb = factorial (b);
printf ("Permutation: %d",
((fa) / (factorial (a-b))));
printf ("Combination: %d", ((fa) / (fb *
factorial (a-b))));

return 0;
}
```