**LAB–3**

**Experiment No. 1:**

TO IMPLEMENT EUCLIDEAN AND EXTENDED EUCLIDEAN ALGORITHM

**Objectives:**

* To implement the Euclidean algorithm.
* To implement the extended Euclidean algorithm in C

**Theory:**

The Euclidean algorithm is a method for finding the greatest common divisor (GCD) of two integers. It is named after the ancient Greek mathematician Euclid. The greatest common divisor, represented as gcd(a, b) and is defined as:

gcd(a, b) = d, where d is the largest number that divides both a and b.

If gcd(a, b) = 1 then we say that a and b are relatively prime which means that both a and b do not divide each other.

The algorithm works by repeatedly dividing the larger integer by the smaller integer, and replacing the larger integer with the remainder of the division. This process is repeated until the remainder is zero, at which point the GCD is the last non-zero remainder.

The Extended Euclidean algorithm is an extension of the Euclidean algorithm that not only calculates the greatest common divisor (GCD) of two integers, but also finds a pair of coefficients that satisfy the equation: ax + by = gcd(a,b)

**Demonstration:**

**Program 1**

**EUCLIDEAN ALGORITHM**

**Source code:**

*#include <stdio.h>*

*int euclidean(int a, int b) {*

*while (b != 0) {*

*int r=a%b;*

*a=b;*

*b=r;*

*}*

*return a;*

*}*

*int main() {*

*int x, y, result;*

*printf("Enter two integers: ");*

*scanf("%d %d", &x, &y);*

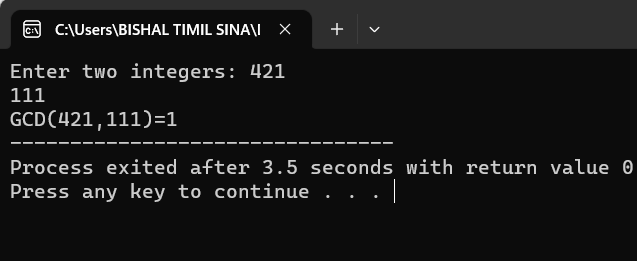
*result = euclidean(x, y);*

*printf("GCD(%d,%d)=%d", x, y, result);*

*return 0;*

*}*

***Output:***

******

**Program no:2**

**To demonstrate the extended Euclidean theorem**

**Source code:**

*#include <stdio.h>*

*void extended\_euclidean(int a, int b, int \*x, int \*y, int \*gcd) {*

*int x0 = 1, y0 = 0, x1 = 0, y1 = 1;*

*int q, r, xn, yn;*

*while (a != 0) {*

*q = b / a;*

*r = b % a;*

*xn = x0 - q \* x1;*

*yn = y0 - q \* y1;*

*x0 = x1;*

*y0 = y1;*

*x1 = xn;*

*y1 = yn;*

*b = a;*

*a = r;*

*}*

*\*x = x0;*

*\*y = y0;*

*\*gcd = b;*

*}*

*int main() {*

*int a, b, x, y, gcd;*

*printf("Enter two numbers: ");*

*scanf("%d %d", &a, &b);*

*extended\_euclidean(a, b, &x, &y, &gcd);*

*printf("Extended Euclidean algorithm:\n");*

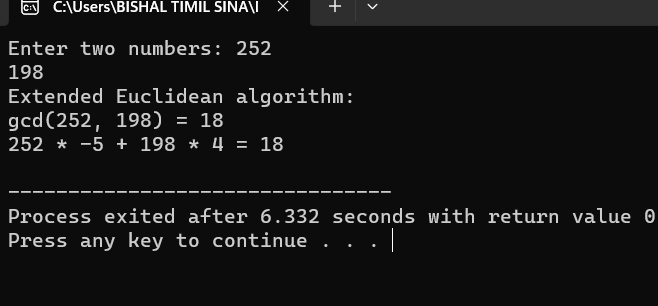
*printf("gcd(%d, %d) = %d\n", a, b, gcd);*

*printf("%d \* %d + %d \* %d = %d\n", a, x, b, y, gcd);*

*return 0;*

*}*

**Output:**

****

**Result and Discussion:**

In the first program, we demonstrated a program to implement the Euclidean algorithm.

Likewise, in the second program, we demonstrated a program to implement the extended

Euclidean algorithm.

**Conclusion:**

Hence, from the above experiment we can create and implement Euclidean and extended euclidean algorithms using C programming language.

**Experiment No. 2:**

TO IMPLEMENT BINARY ADDITION AND MULTIPLICATION

**Objectives:**

* To implement binary addition.
* To implement binary multiplication in C.

**Theory:**

Binary numbers are a numbering system used in computing and digital electronics, where only two symbols, typically 0 and 1, are used to represent all possible values. In contrast to the decimal system used in everyday life, which uses ten digits (0 to 9), the binary system only has two digits, which are commonly referred to as bits.

In the binary system, each digit represents a power of 2, with the rightmost digit representing 2^0 (1), the second-rightmost digit representing 2^1 (2), the third-rightmost digit representing 2^2 (4), and so on. To represent a number in binary, you simply add up the values of the digits that are 1. For example, the binary number 1011 represents the decimal number (1 \* 2^3) + (0 \* 2^2) + (1 \* 2^1) + (1 \* 2^0), which is equal to 11.

Binary numbers are essential for digital computing because they can be easily represented using electronic devices that can distinguish between two states, such as on/off or high/low voltage. This allows for the creation of digital circuits and logical operations that form the basis of modern computer systems. Hence, binary operations like addition, multiplication and so on are also inherently essential.

**Demonstration:**

**Program 1**

*#include <stdio.h>*

*int main() {*

*long n1, n2;*

*int i = 0,c = 0, sum[20];*

*printf("Enter the first binary number: ");*

*scanf("%ld", &n1);*

*printf("Enter the second binary number: ");*

*scanf("%ld", &n2);*

*while (n1 != 0 || n2 != 0) {*

*sum[i++] = (int)((n1 % 10 + n2 % 10 +c) % 2);*

*c = (int)((n1 % 10 + n2 % 10 + c) / 2);*

*n1 = n1 / 10;*

*n2 = n2 / 10;*

*}*

*if (c!= 0) {*

*sum[i++] = c;*

*}*

*--i;*

*printf("Sum of two binary numbers: ");*

*while (i >= 0) {*

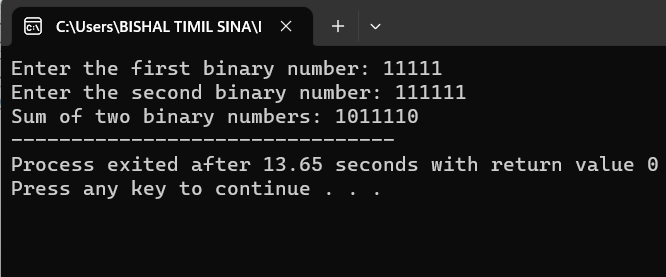
*printf("%d", sum[i--]);*

*}*

*return 0;*

*}*

**Output:**

****

**Program 2**

Source code:

*#include <stdio.h>*

*long binaryAdd(long,long);*

*int main() {*

*long binary1, binary2, multiply = 0;*

*int digit, factor = 1;*

*printf("Enter the first binary number: ");*

*scanf("%ld", &binary1);*

*printf("Enter the second binary number: ");*

*scanf("%ld", &binary2);*

*while (binary2 != 0) {*

*digit = binary2 % 10;*

*if (digit == 1) {*

*binary1 = binary1 \* factor;*

*multiply = binaryAdd(multiply, binary1);*

*}*

*else {*

*binary1 = binary1 \* factor;*

*}*

*binary2 = binary2 / 10;*

*factor = 10;*

*}*

*printf("Product of two binary numbers: %ld", multiply);*

*return 0;*

*}*

*long binaryAdd(long binary1, long binary2) {*

*int i = 0, remainder = 0, sum[20];*

*long binarySum = 0;*

*while (binary1 != 0 || binary2 != 0) {*

*sum[i++] = (int)((binary1 % 10 + binary2 % 10 + remainder) % 2);*

*remainder = (int)((binary1 % 10 + binary2 % 10 + remainder) / 2);*

*binary1 = binary1 / 10;*

*binary2 = binary2 / 10;*

*}*

*if (remainder != 0) {*

*sum[i++] = remainder;*

*}*

*--i;*

*while (i >= 0) {*

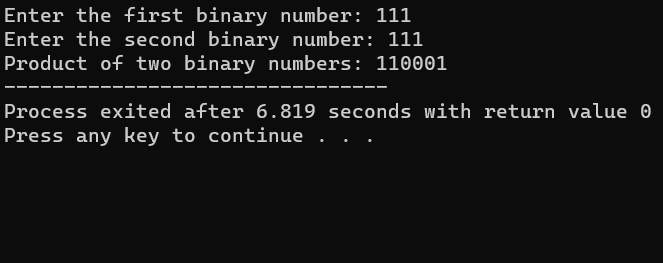
*binarySum = binarySum \* 10 + sum[i--];*

*}*

*return binarySum;*

}

**Output:**

****

**Result and Discussion:**

In the first program, we calculated the sum of two user given binary numbers. Similarly, in the second program we calculated the production of two binary numbers given by the user.

**Conclusion:**

Hence, from the above experiment we are able to implement binary addition and binary multiplication in C programming language.

**LAB–4**

**Experiment No. 1:**

TO PERFORM VARIOUS SET OPERATIONS

**Objectives:**

* To understand various set operations.
* To implement set union, intersection, complement and difference in C.

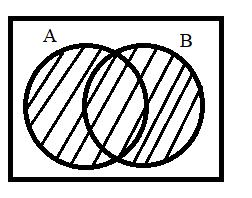
**Theory:**

A set is a collection of distinct elements that have something in common. Sets may be combined and operated in various ways to form new sets. Some of the various set operations are:

* Union
* Intersection
* Complement
* Difference

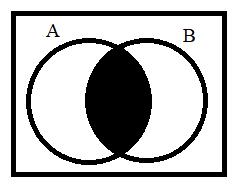
**Union**

The union of two sets A and B denoted by AՍB is the set of all those elements which either belongs to A or B or both . AՍB is read as ‘ A union B’. Symbolically, A Ս B = {x: x ∈A or x∈ B}= {x: x∈A V x∈B}. AՍB can be represented as a Venn diagram in the following way:



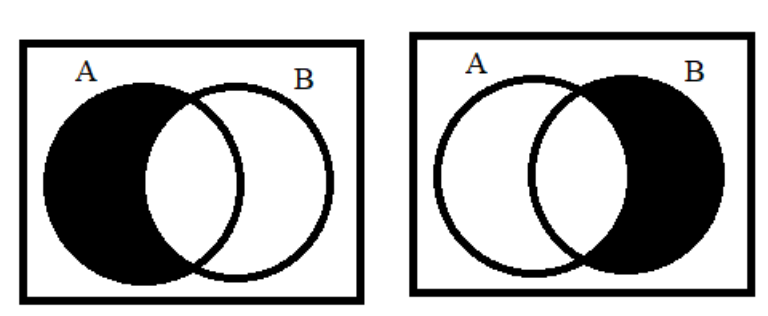
**Intersection**

The intersection of two sets A and B denoted by A∩B is the set of all those elements which belong to both A and B. A∩B is read as ‘ A intersection B’. Symbolically, A∩B = {x: x∈A and x∈B} = {x: x∈A ∧ x∈B}. A∩B can be represented as a Venn diagram in the following way:



**Difference**

The difference of two sets A and B denoted by A-B is the set of elements of A which do not belong to B. A-B is read as ‘ A minus B. Symbolically, A-B = {x: x∈A and x∉B}. Similarly B-A ={x: x∈B and x∉A}. A-B and B-A can be represented as Venn diagrams in the following way:

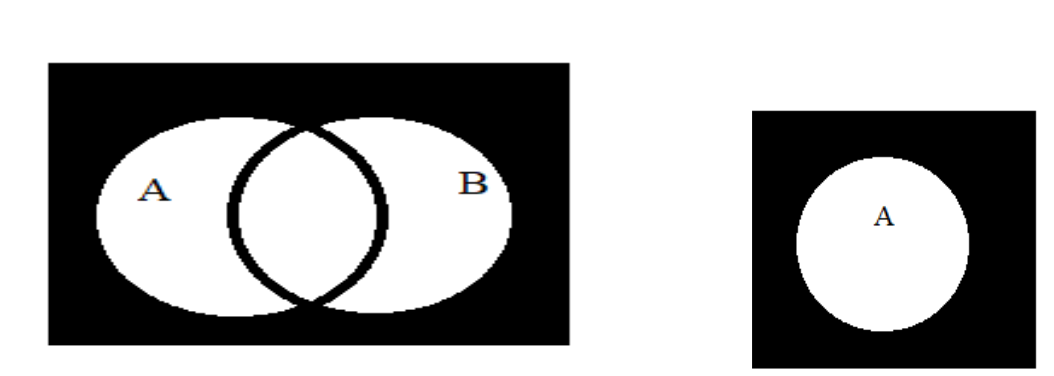


**Complement**

Let A be the subset of a universal set U. Then the complement of A with respect to U is the set of

those elements of U which do not belong to A and denoted by or Ac or A***’***. Symbolically,

Ս = {x: x ∈ Ս }= {x: ∉ Ս }. = {x:∈ } = {x: x ∉ A}. Ս and can be represented in Venn diagram as:



**Demonstration:**

**Program 1**

**TO DO SOME SET OPERATION:**

*/\* IN this single program we can do some set operation using function*

*1.union*

*2.intersectiom*

*3.difference*

*\*/*

*#include<stdio.h>*

*void find\_union(int set1[],int set2[],int n1,int n2,int union\_set[]);*

*void find\_intersection(int set1[],int set2[],int n1,int n2,int intersection[]);*

*void find\_difference(int set1[],int set2[],int n1,int n2,int difference[]);*

*int main(){*

*int set1[10],set2[10],union\_set[25],intersection[15],difference[25];*

*int n1,n2,i;*

*printf("Enter the size of first set\n");*

*scanf("%d",&n1);*

*printf("Enter the size of second set\n");*

*scanf("%d",&n2);*

*printf("Enter the first set\n");*

*for(i=0;i<n1;i++){*

*scanf("%d",&set1[i]);*

*}*

*printf("Enter the second set\n");*

*for(i=0;i<n2;i++){*

*scanf("%d",&set2[i]);*

*}*

*find\_union( set1,set2,n1,n2,union\_set);*

*find\_intersection(set1,set2,n1,n2,intersection);*

*find\_difference(set1,set2,n1,n2,difference);*

*}*

*void find\_union(int set1[], int set2[], int n1, int n2, int unionSet[]) {*

*int i, j, k;*

*k = 0;*

*for (i = 0; i < n1; i++) {*

*unionSet[k] = set1[i];*

*k++;*

*}*

*for (i = 0; i < n2; i++) {*

*for (j = 0; j < n1; j++) {*

*if (set2[i] == set1[j]) {*

*break;*

*}*

*}*

*if (j == n1) {*

*unionSet[k] = set2[i];*

*k++;*

*}*

*}*

*printf("The union of set\n{");*

*for(i=0;i<k;i++){*

*printf("%d",unionSet[i]);*

*if(i<k-1){*

*printf(",");*

*}*

*}*

*printf("}\n");*

*}*

*void find\_intersection(int set1[],int set2[],int n1,int n2,int intersection[]){*

*int i,j,k=0;*

*for (i = 0; i < n1; i++) {*

*for (j = 0; j < n2; j++) {*

*if (set1[i] == set2[j]) {*

*intersection[k] = set1[i];*

*k++;*

*}*

*}*

*}*

*printf("\nIntersection set:* *\n {");*

*for (i = 0; i < k; i++) {*

*printf("%d", intersection[i]);*

*if (i < k - 1) {*

*printf(", ");*

*}*

*}*

*printf("}\n");*

*}*

*void find\_difference(int set1[],int set2[],int n1,int n2,int difference[]){*

*int i,j,k=0,l=0;*

*for (i = 0; i < n1; i++) {*

*difference[k] = set1[i];*

*k++;}*

*for (i = 0; i < n1; i++) {*

*for (j = 0; j < n2; j++) {*

*if (set2[i] == set1[j])*

*break;*

*}*

*if (j!=n1){*

*difference[k]=set2[j];*

*l++;*

*}*

*}*

*printf("\nDifference set:* *\n {");*

*for (i = 0; i < l; i++) {*

*printf("%d", difference[i]);*

*if (i < l- 1) {*

*printf(", ");*

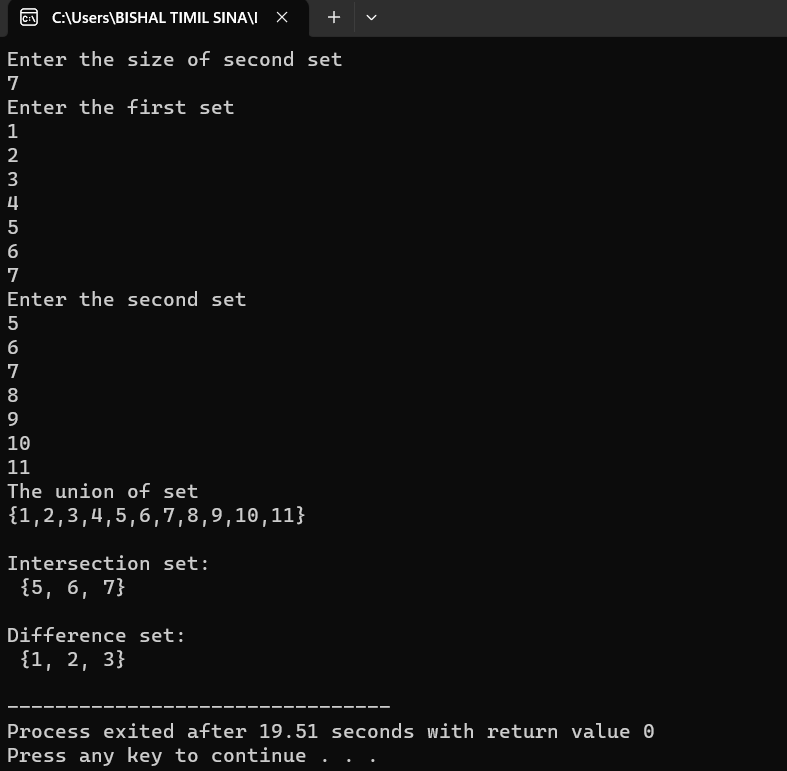
*}*

*}*

*printf("}\n");*

*}*

***Output:***

****

**Result and Discussion:**

In the first program, we created a program to find the union, intersection and of two user given sets. For union we created the find\_union function ,for intersection we created find\_intersection function and find\_different is created to find the difference of the set and complement set is also an difference set we assume that first set is universal set and second set is an set A. so we can demonstrate all operation of set.

**Conclusion:**

Hence, from the above experiment we are able to implement various set operations including union, intersection, difference and complement of sets in C programming language.

**LAB–5**

**Experiment No. 1:**

TO GENERATE PERMUTATIONS AND COMBINATIONS

**Objectives:**

* To generate permutations in C programming language.
* To generate combinations in Cprogramming language.

**Theory:**

Permutations and combinations are two fundamental concepts in combinatorics, a branch of mathematics that deals with counting and arranging objects. Both concepts are used to calculate the number of possible outcomes when selecting or arranging a subset of objects from a larger set. Both permutations and combinations have many applications in probability theory, statistics, and other fields that involve counting and arranging objects.

**Permutation**

A permutation is an arrangement of objects in a specific order. Permutations are used to calculate the number of possible arrangements of objects. The number of permutations of a set of n objects taken r at a time is denoted by P(n,r) which is calculated using the formula: P(n,r) = n! / (n-r)!

**Combination**

A combination is a selection of objects from a set, without regard to the order in which they are

chosen. Combinations are used to calculate the number of possible selections of objects without

regard to order. The number of combinations of n objects taken r at a time is denoted by C(n,r)

which is calculated by the formula:

C(n,r) = n! / (n-r)!r!

**Demonstration:**

**Program 1:**

**Permutation:**

Source code:  
*#include<stdio.h>*

*int main(){*

*int r,n,i;*

*long f1=1,f2=1,p;*

*printf("Enter the value for N and R\n");*

*scanf("%d%d",&n,&r);*

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

*f1\*=i;*

*}*

*for(i=1;i<=(n-r);i++){*

*f2\*=i;*

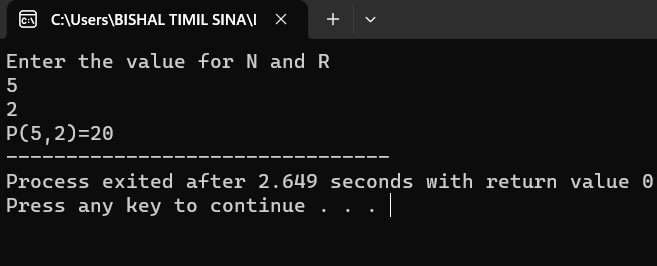
*}*

*p=f1/f2;*

*printf("P(%d,%d)=%d",n,r,p);*

*}*

**Output:**

****

**Program no :2**

**Combination:**source code:  
*#include<stdio.h>*

*int main(){*

*int r,n,i;*

*long a=1,b=1, c=1,C;*

*printf("Enter the value for N and R\n");*

*scanf("%d%d",&n,&r);*

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

*a\*=i;*

*}*

*for(i=1;i<=(n-r);i++){*

*b\*=i;*

*}*

*for(i=1;i<=r;i++){*

*c\*=i;*

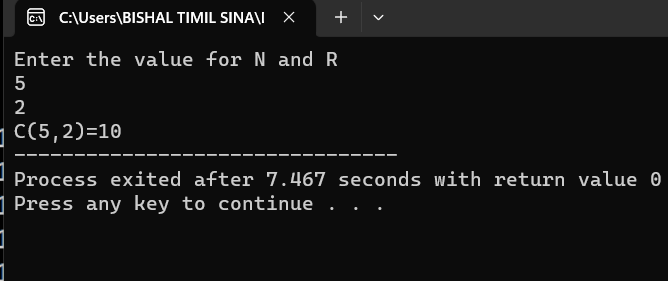
*}*

*C=a/(b\*c);*

*printf("C(%d,%d)=%d",n,r,C);*

*}*

***Output:***

******

**Result and Discussion:**

In the above program, we demonstrated a program to find the number of permutations and combinations according to the input provided by the user.

**Conclusion:**

Hence, from the above experiment we are able to generate permutations and combinations using C programming language.