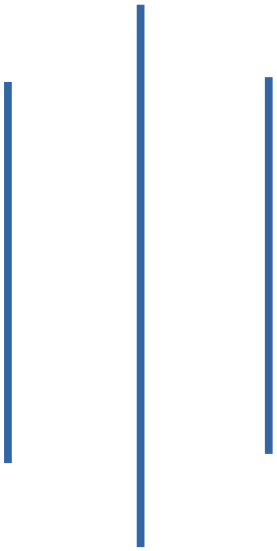
TRIBHUVAN UNIVERSITY

PATAN MULTIPLE CAMPUS

PATANDHOKA, LALITPUR



SUBJECT: DATA STRUCTURES AND ALGORITHMS (BIT 201)

SUBMITTED BY

NAME: SURESH DAHAL

ROLL NO.: 23

CLASS: BIT – II/I

DATE: 2081-08-28

SUBMITTED TO

……...

CHECKED BY

1. Write a program to form a sparse matrix and print all the non-zero elements with their location address.

Algorithm

1. Start.
2. Take input for the dimensions of the matrix (rows and columns).
3. Initialize the matrix with user input for each element.
4. Traverse the matrix to identify non-zero elements. For each non-zero element, store its value along with its row and column indices.
5. Print all non-zero elements along with their row and column indices.
6. Stop.

Example

Consider the following 3x3 matrix:

Matrix:
1 0 0
0 0 5
0 7 0

The non-zero elements are:

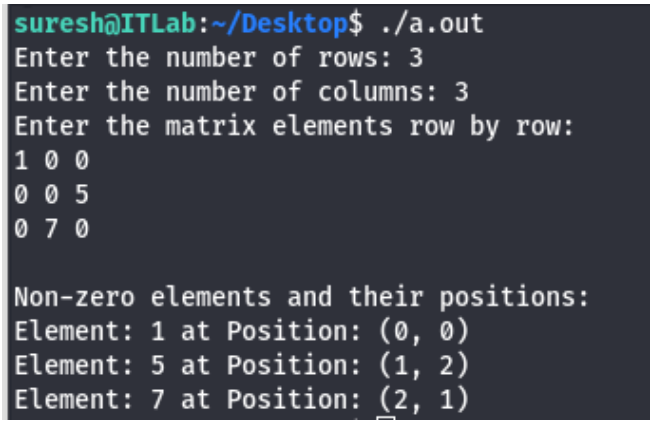
- 1 at (0, 0)
- 5 at (1, 2)
- 7 at (2, 1)

Program

```
#include <stdio.h>
int main() {
   int rows, cols;
   printf("Enter the number of rows: ");
   scanf("%d", &rows);
```

```
    printf("Enter the number of columns: ");
    scanf("%d", &cols);
    int matrix[rows][cols];
    printf("Enter the matrix elements row by row:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix[i][j]);
        }
    }
    printf("\nNon-zero elements and their positions:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            if (matrix[i][j] != 0) {
                printf("Element: %d at Position: (%d, %d)\n", matrix[i][j], i, j);
            }
        }
    }
    return 0;
}
```

Output



Conclusion

Hence we have written a program to form a sparse matrix and return all non-zero elements with their location address.

2. Define recursion with suitable example. Write algorithm and program with output of the following:

a. Calculate factorial for n integer number

b. Prime number checking

c. Fibonacci series

d. Tower of Hanoi

Theory: Recursion is a programming technique in which a function calls itself to solve a smaller instance of the same problem. This process continues until a base condition is met, which stops further recursive calls. Recursion is commonly used for problems that can be broken into similar sub-problems, such as factorial calculation, Fibonacci sequence, and solving the Tower of Hanoi.

For example, we can find factorial of a number using recursion as:

```
int factorial(int n) {
   if (n == 0 || n == 1) {
      return 1; // Base case
   } else {
      return n * factorial(n - 1); // Recursive call
   }
}
```

a. Calculate factorial for n integer number

Algorithm

1. Start
2. Define a recursive function factorial(n).

   If n == 0 or n == 1, return 1.

   Otherwise, return n * factorial(n – 1).
3. Take input n from the user.
4. Call the recursive function to compute the factorial of n.

5. Display the result.
6. Stop.

Example

Consider we have to find the factorial of 3

3! = 3*2!

2! = 2*1!

1! = 1

So, 3! = 3*2*1 = 6

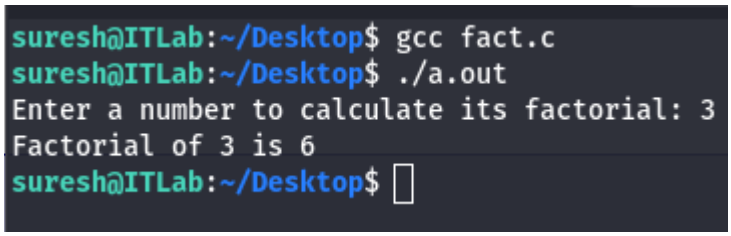
Program

```
#include <stdio.h>
int factorial(int n) {
   if (n == 0 || n == 1) {
      return 1; // Base case
   } else {
      return n * factorial(n - 1); // Recursive call
   }
}
int main() {
   int n;
   printf("Enter a number to calculate its factorial: ");
   scanf("%d", &n);

   if (n < 0) {
      printf("Factorial is not defined for negative numbers.\n");
   } else {
      printf("Factorial of %d is %d\n", n, factorial(n));
   }

   return 0;
}
```

Output



Conclusion

Hence we have used the recursion to find the factorial of the given integer n.

b. Prime number checking

Algorithm

1. Start.
2. Define a recursive function isPrime(n, i).

   If n <= 2, return n == 2.

   If n % i == 0, return false

   If i * i > n, return true.

   Otherwise, call isPrime(n, i + 1).
3. Take input n from the user.
4. Call the recursive function to check if n is prime.
5. Display the result.
6. Stop.

Example

Consider that we have to check whether 5 is prime or not.

5 % 2 = 1, not prime

2*2 > 5 = false, not prime

5 % 3 = 2, not prime

3*3 > 5 = true, prime

Because, if square of a number is greater than the given number and it doesn't divide the given number, the given number is prime.

Program

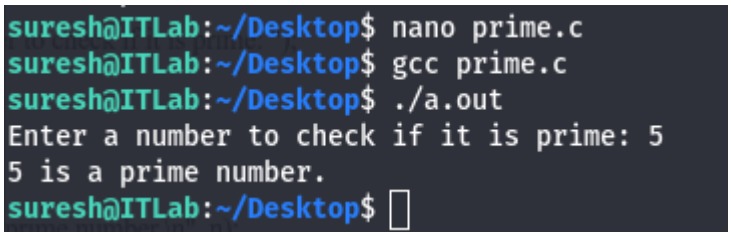
```
#include <stdio.h>
#include <stdbool.h>
bool isPrime(int n, int i) {
    if (n <= 2) {
        return (n == 2);
    }
    if (n % i == 0) {
        return false;
    }
    if (i * i > n) {
        return true;
    }
    return isPrime(n, i + 1);
}

int main() {
    int n;
    printf("Enter a number to check if it is prime: ");
    scanf("%d", &n);

    if (n <= 1) {
        printf("%d is not a prime number.\n", n);
    } else if (isPrime(n, 2)) {
        printf("%d is a prime number.\n", n);
    } else {
        printf("%d is not a prime number.\n", n);
    }

    return 0;
}
```

Output



Conclusion

Hence we have used recursion to check whether the given number is prime or not.

c. Fibonacci series

Algorithm

1. Start.
2. Define a recursive function fibonacci(n).

   If n == 0, return 0.

   If n == 1, return 1.

   Otherwise, return fibonacci(n - 1) + fibonacci(n - 2).
3. Take input n from the user.
4. Use a loop to compute and display the Fibonacci series up to n terms.
5. Stop.

Example

Consider we have to find first 5 Fibonacci numbers.

The first two cases will be f0 = 0, f1 = 1

Then, the third term f2 = f(2-1) + f(2-2)

i.e. f2 = f1 + f0 = 0+1 = 1

Similarly, f3 = f2+f1 = 1+1 = 2

and f4 = f3+f2 = 3

So, the first 5 fibonacci series would be: 0 1 1 2 3

Program

```
#include <stdio.h>

int fibonacci(int n) {
    if (n == 0) {
        return 0;
    } else if (n == 1) {
        return 1;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}

int main() {
    int n;
    printf("Enter the number of terms in the Fibonacci series: ");
    scanf("%d", &n);

    printf("Fibonacci series up to %d terms:\n", n);
    for (int i = 0; i < n; i++) {
        printf("%d ", fibonacci(i));
    }
    printf("\n");

    return 0;
}
```

Output

```
suresh@ITLab:~/Desktop$ nano fibonacci.c
suresh@ITLab:~/Desktop$ gcc fibonacci.c
suresh@ITLab:~/Desktop$ ./a.out
Enter the number of terms in the Fibonacci series: 5
Fibonacci series up to 5 terms:
0 1 1 2 3
```

Conclusion

Hence we have implemented a program to find fibonacci numbers using recursion.

d. Tower of Hanoi

Algorithm

1. Start.
2. Define a recursive function towerOfHanoi(n, source, target, auxiliary).
   if n == 1, move the disk from source to target.
   Otherwise:
      Call towerOfHanoi(n - 1, source, auxiliary, target).
      Move the disk from source to target.
      Call towerOfHanoi(n - 1, auxiliary, target, source).
3. Take input n (number of disks).
4. Call the recursive function to solve the problem.
5. Stop.

Example

Consider we have 3 disks placed in ascending order in peg A, there are 3 pegs A, B, C and we have to transfer all 3 disks from peg A to peg C following rules of ToH.

The following steps would be involved:

Disk 1 moved from A to C

Disk 2 moved from A to B

Disk 1 moved from C to B

Disk 3 moved from A to C

Disk 1 moved from B to A

Disk 2 moved from B to C

Disk 1 moved from A to C

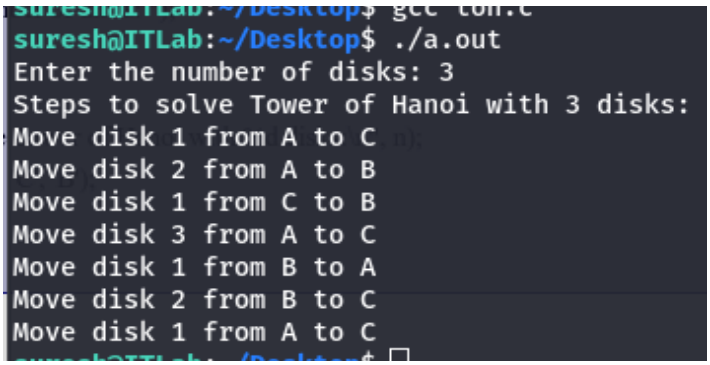
Program

```
#include <stdio.h>

void towerOfHanoi(int n, char source, char target, char auxiliary) {
    if (n == 1) {
        printf("Move disk 1 from %c to %c\n", source, target);
        return;
    }
    towerOfHanoi(n - 1, source, auxiliary, target);
    printf("Move disk %d from %c to %c\n", n, source, target);
    towerOfHanoi(n - 1, auxiliary, target, source);
}

int main() {
    int n;
    printf("Enter the number of disks: ");
    scanf("%d", &n);
    printf("Steps to solve Tower of Hanoi with %d disks:\n", n);
    towerOfHanoi(n, 'A', 'C', 'B');
    return 0;
}
```

Output



Conclusion:

Hence we have used recursion to implement Tower of Hanoi algorithm.