Lab no: 01 **Date:2081/09/**

Title: Write a program to demonstrate Dynamic memory allocation in c

Dynamic memory allocation in C refers to the process of allocating memory at runtimes, as opposite to that of static memory allocation (which happens at compile time). It allows for more flexible memory management, especially when the exact memory requirements are not known in advance.

In C, dynamic memory is manages using functions from the standard library:

1. **Malloc ():**

   The malloc () function allocates a single block of memory of a specified size and returns a pointer to the beginning of the block. The memory is not initialized, meaning it contains garbage values.

2. **Calloc ():**

   The Calloc () function allocates memory for an array of elements, initializes them to zero, and returns a pointer to the memory.

3. **Realloc ():**

   The realloc () function changes the size of the memory block pointed to by ptr to newSize. It preserves the content up to the minimum of the old and new sizes.

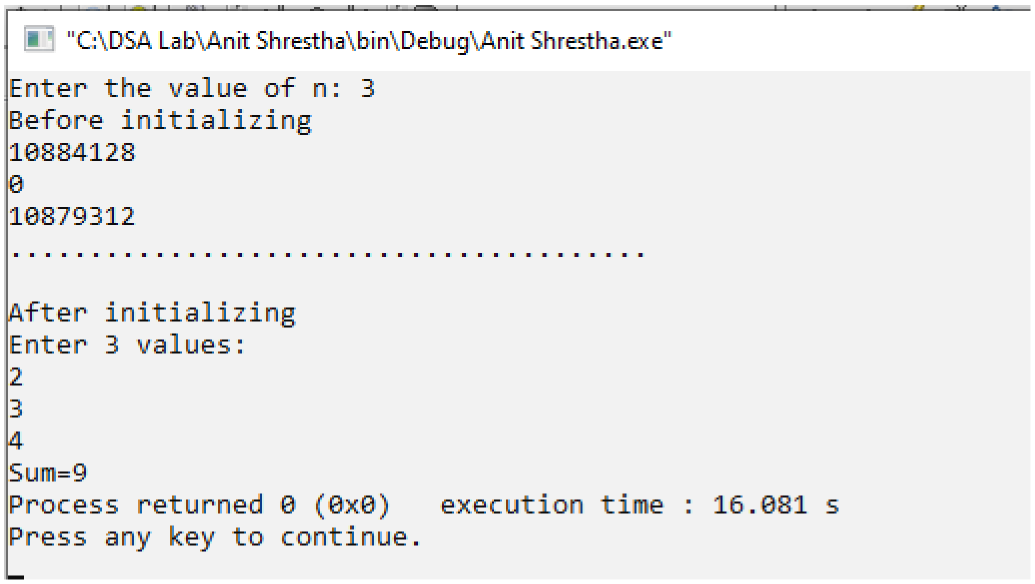
Programming Language: C

IDE: Code Blocks

//Use of malloc ()

```
#include <stdio.h>
 #include <stdlib.h>
 int main()
{
 system("color f0");
  int n,i,sum=0;

  printf("Enter the value of n: ");
  scanf("%d",&n);

  int *ptr;
  ptr=(int*)malloc(n*sizeof(int));

  printf("Before initializing \n");
  for(i=0;i<n;i++) {
     printf("%d",*(ptr+i));
  printf("\n");
  }
  printf("......................................\n\n");
 printf("After initializing \n");
 printf("Enter %d values: \n", n);
 for(i=0;i<n;i++) {
     scanf("%d",ptr+i);
     sum=sum+*(ptr+i);
 }
 printf("Sum=%d",sum);
 free(ptr);
     return 0;
 }
```

Output:

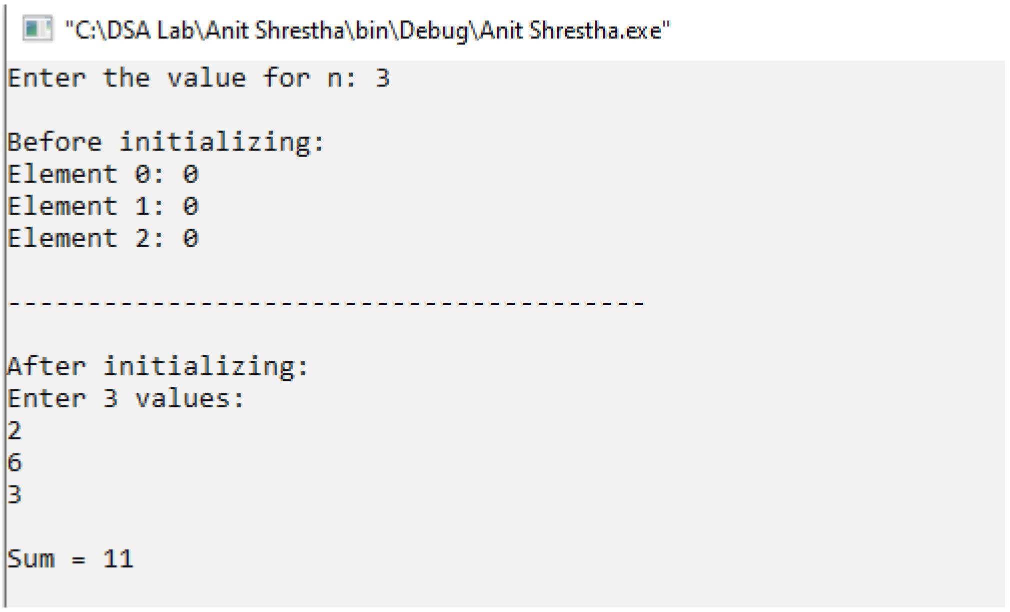


```
"C:\DSA Lab\Anit Shrestha\bin\Debug\Anit Shrestha.exe"
Enter the value of n: 3
Before initializing
10884128
0
10879312
........................................

After initializing
Enter 3 values:
2
3
4
Sum=9
Process returned 0 (0x0)   execution time : 16.081 s
Press any key to continue.
```

//Use of calloc ()

```
#include <stdio.h>
 #include <stdlib.h>

 int main() {
     system("color f0");

     int n, i, sum = 0;

     printf("Enter the value for n: ");
     scanf("%d", &n);

     // Allocate memory for n integers using calloc
     int *ptr = (int*)calloc(n, sizeof(int));
     if (ptr == NULL) {
         printf("Memory allocation failed!\n");
         return 1; // Exit the program if memory allocation fails
     }

     // Display the memory content before initialization
     printf("\nBefore initializing:\n");
     for (i = 0; i < n; i++) {
         printf("Element %d: %d\n", i, *(ptr + i));
     }

     printf("\n----------------------------------------\n\n");

     // Input values and calculate the sum
     printf("After initializing:\n");
     printf("Enter %d values: \n", n);
     for (i = 0; i < n; i++) {
         scanf("%d", ptr + i);
         sum += *(ptr + i);
     }

     // Display the sum
     printf("\nSum = %d\n", sum);

     // Free the allocated memory
     free(ptr);

     return 0;
 }
```

Output:



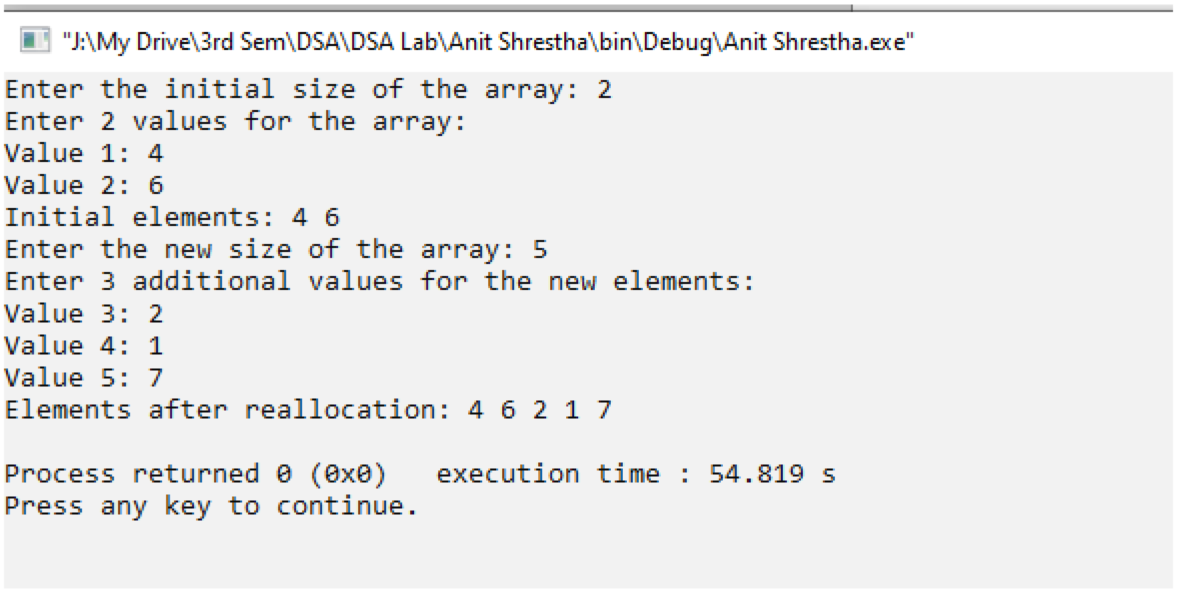
"C:\DSA Lab\Anit Shrestha\bin\Debug\Anit Shrestha.exe"

```
Enter the value for n: 3

Before initializing:
Element 0: 0
Element 1: 0
Element 2: 0

----------------------------------------

After initializing:
Enter 3 values:
2
6
3

Sum = 11
```

//Use of realloc

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *marks; // Pointer to hold the base address of the allocated memory
    int n; // Initial number of elements
    int newSize; // New size after reallocation
    int i;

    // Ask the user for the initial size of the array
    printf("Enter the initial size of the array: ");
    scanf("%d", &n);

    // Dynamically allocate memory using malloc
    marks = (int*)malloc(n * sizeof(int));

    // Check if the memory has been successfully allocated
    if (marks == NULL) {
        printf("Memory not allocated.\n");
        exit(0);
    }

    // Ask the user to input values for the array
    printf("Enter %d values for the array:\n", n);
    for (i = 0; i < n; i++) {
        printf("Value %d: ", i + 1);
        scanf("%d", &marks[i]); // Assign user-defined values to the array
    }

    // Print the initial elements
    printf("Initial elements: ");
    for (i = 0; i < n; i++) {
        printf("%d ", marks[i]);
    }
    printf("\n");

    // Ask the user for the new size of the array
    printf("Enter the new size of the array: ");
    scanf("%d", &newSize);

    // Resize the array using realloc
    marks = (int*)realloc(marks, newSize * sizeof(int));

    // Check if the memory has been successfully reallocated
    if (marks == NULL) {
        printf("Reallocation failed.\n");
        exit(0);
    }

    // Initialize the new elements if the new size is greater than the old size
    if (newSize > n) {
        printf("Enter %d additional values for the new elements:\n", newSize - n);
        for (i = n; i < newSize; i++) {
            printf("Value %d: ", i + 1);
            scanf("%d", &marks[i]); // Assigning user-defined values to the new elements
        }
    }

    // Printing the elements after reallocation
    printf("Elements after reallocation: ");
    for (i = 0; i < newSize; i++) {
        printf("%d ", marks[i]);
    }
    printf("\n");

    // Free the allocated memory
    free(marks);

    return 0;
}
```

Output



```
"J:\My Drive\3rd Sem\DSA\DSA Lab\Anit Shrestha\bin\Debug\Anit Shrestha.exe"
Enter the initial size of the array: 2
Enter 2 values for the array:
Value 1: 4
Value 2: 6
Initial elements: 4 6
Enter the new size of the array: 5
Enter 3 additional values for the new elements:
Value 3: 2
Value 4: 1
Value 5: 7
Elements after reallocation: 4 6 2 1 7

Process returned 0 (0x0)   execution time : 54.819 s
Press any key to continue.
```

Lab no: 02 **Date:2081/09/**

Title: Write a Menu based program to show the basic operations of Stack.

Stack are a fundamental data structure with simple operations that are efficient and widely used in various applications in computer science. Their LIFO nature makes them suitable for scenarios where the most recently added item needs to be accessed first.

Basic Operations of a Stack

1. Push
    - Adds an element to the top of the stack
    - If the stack is not full, the new element is placed on top of the current top element.
2. Pop
    - Removes the top element from the stack
    - If the stack is not empty, the top element is removed, and the next element becomes the new top.
3. isEmpty
    - Checks whether the stack is empty.
    - Returns true if the stack has no elements, otherwise it returns false.
4. isFull
    - Checks whether the stack is full.
    - Returns true if the stack cannot accept more elements, otherwise it returns false.

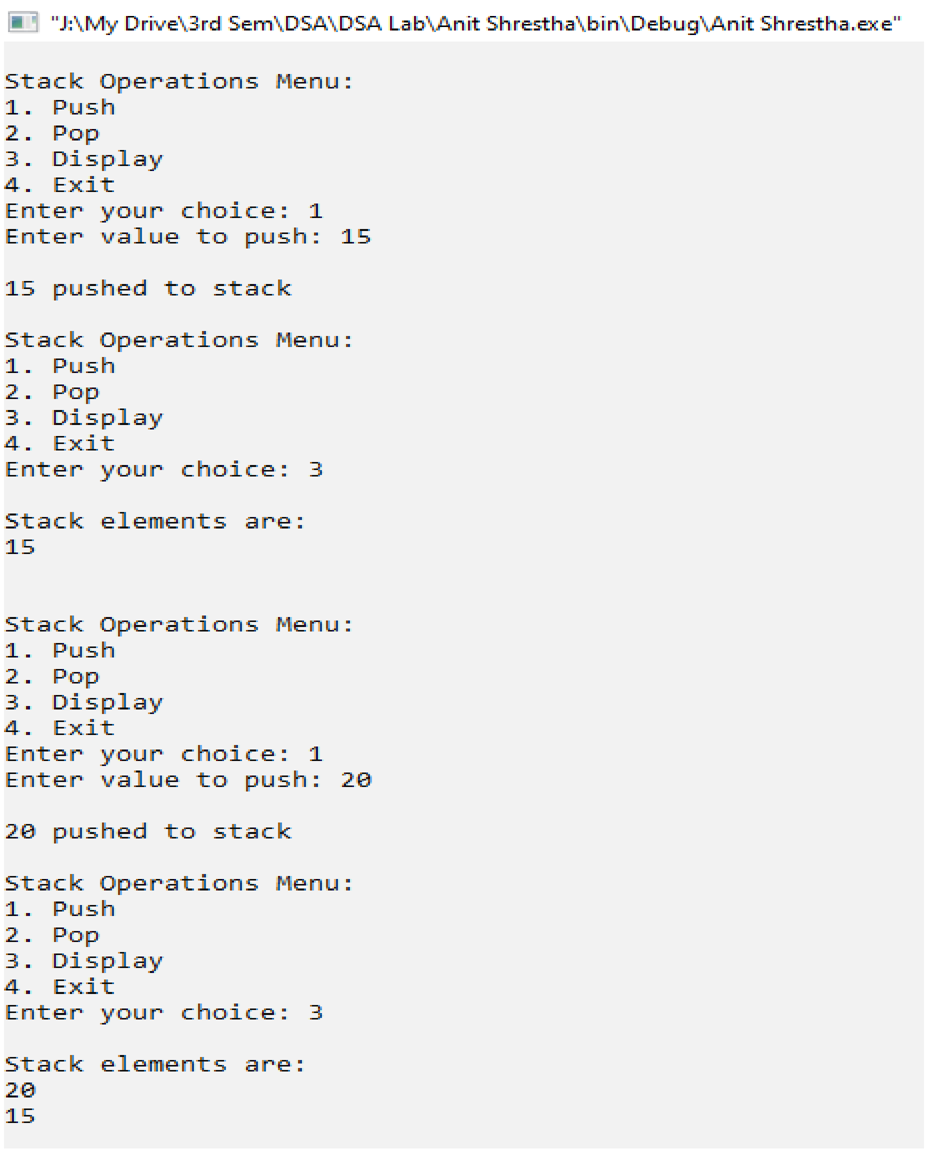
Programming Language: C

IDE: Code Blocks

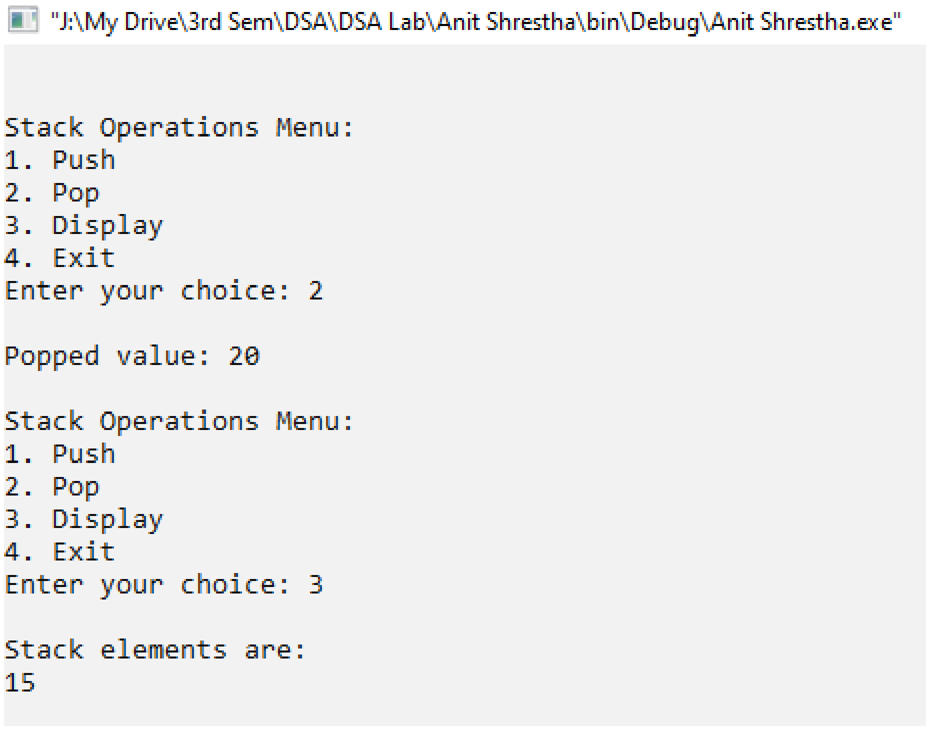
Source Code

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100

// Structure for Stack
struct Stack {
    int items[MAX_SIZE];
    int top;
};

// Function to initialize stack
void initializeStack(struct Stack *s) {
    s->top = -1;
}

// Function to check if stack is full
int isFull(struct Stack *s) {
    return s->top == MAX_SIZE - 1;
}

// Function to check if stack is empty
int isEmpty(struct Stack *s) {
    return s->top == -1;
}

// Function to push an element
void push(struct Stack *s, int value) {
    if (isFull(s)) {
        printf("\nStack Overflow! Cannot push %d\n", value);
    } else {
        s->items[++(s->top)] = value;
        printf("\n%d pushed to stack\n", value);
    }
}

// Function to pop an element
int pop(struct Stack *s) {
    if (isEmpty(s)) {
        printf("\nStack Underflow! Cannot pop from empty stack\n");
        return -1;
    } else {
        return s->items[(s->top)--];
    }
}

// Function to display the stack
void display(struct Stack *s) {
    if (isEmpty(s)) {
        printf("\nStack is empty!\n");
        return;
    }
    printf("\nStack elements are:\n");
    for (int i = s->top; i >= 0; i--) {
        printf("%d\n", s->items[i]);
    }
    printf("\n");
}

int main() {
    struct Stack stack;
    initializeStack(&stack);

    int choice, value;

    do {
        printf("\nStack Operations Menu:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(&stack, value);
                break;

            case 2:
                value = pop(&stack);
                if (value != -1) {
                    printf("\nPopped value: %d\n", value);
                }
                break;

            case 3:
                display(&stack);
                break;

            case 4:
                printf("\nExiting program...\n");
                break;

            default:
                printf("\nInvalid choice! Please try again.\n");
        }
    } while (choice != 4);

    return 0;
}
```

Output:



```
"J:\My Drive\3rd Sem\DSA\DSA Lab\Anit Shrestha\bin\Debug\Anit Shrestha.exe"

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 15

15 pushed to stack

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3

Stack elements are:
15


Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 1
Enter value to push: 20

20 pushed to stack

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3

Stack elements are:
20
15
```



```
"J:\My Drive\3rd Sem\DSA\DSA Lab\Anit Shrestha\bin\Debug\Anit Shrestha.exe"


Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 2

Popped value: 20

Stack Operations Menu:
1. Push
2. Pop
3. Display
4. Exit
Enter your choice: 3

Stack elements are:
15
```

Lab no: 03 **Date:2081/09/**

Title: Write a program to evaluate the user input postfix or prefix expression.

Postfix:

In postfix notation, operators follow their operands. For example, the expression 'A+B' is written as 'AB+'.

To evaluate a postfix expression:

1. Use a stack to hold operands.
2. Read the expression from left to right.
3. For each token
    - If it is an operand, push it into the stack
    - If it is an operator, pop the required number of operands from the stack, apply the operator, and push the result back onto the stack.
4. At the end of the expression, the stack will contain one element, which is the result.

Prefix:

In postfix notation, operators precede their operands. For example, the expression 'A+B' is written as '+AB'.

To evaluate a postfix expression:

1. Use a stack to hold operands.
2. Read the expression from right to left.
3. For each token
    - If it is an operand, push it into the stack
    - If it is an operator, pop the required number of operands from the stack, apply the operator, and push the result back onto the stack.
4. At the end of the expression, the stack will contain one element, which is the result.

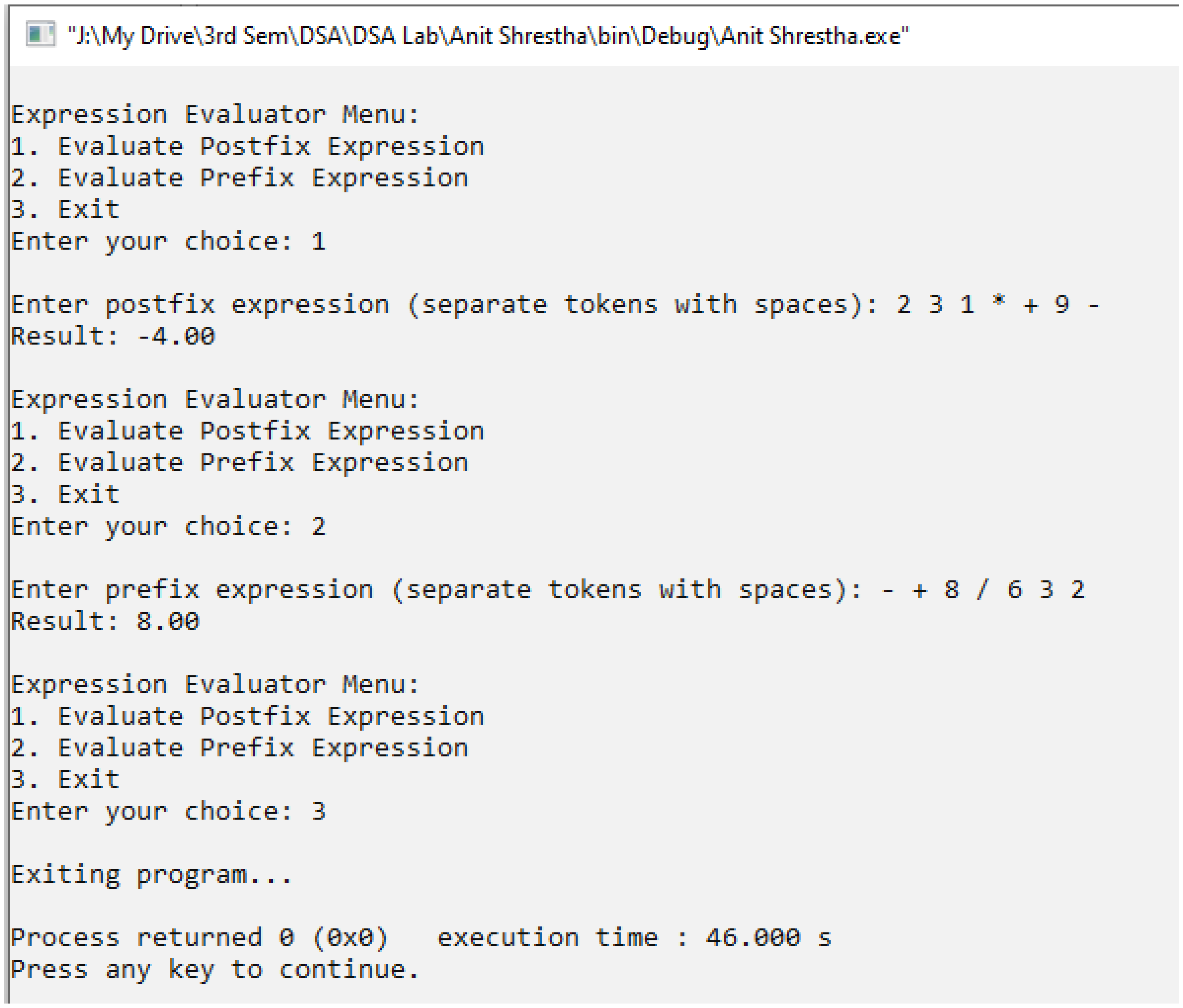
Programming Language: C

IDE: Code Blocks

Source Code

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#define MAX_SIZE 100

// Stack structure
struct Stack {
    float items[MAX_SIZE];
    int top;
};

// Stack operations
void initStack(struct Stack *s) {
    s->top = -1;
}

void push(struct Stack *s, float value) {
    s->items[++s->top] = value;
}

float pop(struct Stack *s) {
    return s->items[s->top--];
}

// Function to check if character is an operator
int isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}

// Function to perform operation
float performOperation(float op1, float op2, char operator) {
    switch(operator) {
        case '+': return op1 + op2;
        case '-': return op1 - op2;
        case '*': return op1 * op2;
        case '/': return op1 / op2;
        default: return 0;
    }
}

// Function to evaluate postfix expression
float evaluatePostfix(char* expression) {
    struct Stack stack;
    initStack(&stack);

    int i;
    float op1, op2, result;
    char *token = strtok(expression, " ");

    while(token != NULL) {
        // If token is a number
        if(isdigit(token[0]) || (token[0] == '-' && isdigit(token[1]))) {
            push(&stack, atof(token));
        }
        // If token is an operator
        else if(strlen(token) == 1 && isOperator(token[0])) {
            op2 = pop(&stack);
            op1 = pop(&stack);
            result = performOperation(op1, op2, token[0]);
            push(&stack, result);
        }
        token = strtok(NULL, " ");
    }

    return pop(&stack);
}

// Function to evaluate prefix expression
float evaluatePrefix(char* expression) {
    struct Stack stack;
    initStack(&stack);

    // Find length of expression
    int length = strlen(expression);

    // Read from right to left
    int i;
    float op1, op2, result;
    char *token;

    // Create a copy of expression to preserve original
    char expr_copy[MAX_SIZE];
    strcpy(expr_copy, expression);

    // Get last token
    token = strtok(expr_copy, " ");
    char *tokens[MAX_SIZE];
    int token_count = 0;

    // Store all tokens
    while(token != NULL) {
        tokens[token_count++] = token;
        token = strtok(NULL, " ");
    }

    // Process tokens in reverse
    for(i = token_count - 1; i >= 0; i--) {
        // If token is a number
        if(isdigit(tokens[i][0]) || (tokens[i][0] == '-' && isdigit(tokens[i][1]))) {
            push(&stack, atof(tokens[i]));
        }
        // If token is an operator
        else if(strlen(tokens[i]) == 1 && isOperator(tokens[i][0])) {
            op1 = pop(&stack);
            op2 = pop(&stack);
            result = performOperation(op1, op2, tokens[i][0]);
            push(&stack, result);
        }
    }

    return pop(&stack);
}

int main() {
    char expression[MAX_SIZE];
    int choice;
    float result;

    do {
        printf("\nExpression Evaluator Menu:\n");
        printf("1. Evaluate Postfix Expression\n");
        printf("2. Evaluate Prefix Expression\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar(); // Clear input buffer

        switch(choice) {
            case 1:
                printf("\nEnter postfix expression (separate tokens with spaces): ");
                fgets(expression, MAX_SIZE, stdin);
                expression[strcspn(expression, "\n")] = 0; // Remove newline
                result = evaluatePostfix(expression);
                printf("Result: %.2f\n", result);
                break;

            case 2:
                printf("\nEnter prefix expression (separate tokens with spaces): ");
                fgets(expression, MAX_SIZE, stdin);
                expression[strcspn(expression, "\n")] = 0; // Remove newline
                result = evaluatePrefix(expression);
                printf("Result: %.2f\n", result);
                break;

            case 3:
                printf("\nExiting program...\n");
                break;

            default:
                printf("\nInvalid choice! Please try again.\n");
        }
    } while(choice != 3);

    return 0;
}
```

Output:



"J:\My Drive\3rd Sem\DSA\DSA Lab\Anit Shrestha\bin\Debug\Anit Shrestha.exe"

```
Expression Evaluator Menu:
1. Evaluate Postfix Expression
2. Evaluate Prefix Expression
3. Exit
Enter your choice: 1

Enter postfix expression (separate tokens with spaces): 2 3 1 * + 9 -
Result: -4.00

Expression Evaluator Menu:
1. Evaluate Postfix Expression
2. Evaluate Prefix Expression
3. Exit
Enter your choice: 2

Enter prefix expression (separate tokens with spaces): - + 8 / 6 3 2
Result: 8.00

Expression Evaluator Menu:
1. Evaluate Postfix Expression
2. Evaluate Prefix Expression
3. Exit
Enter your choice: 3

Exiting program...

Process returned 0 (0x0)   execution time : 46.000 s
Press any key to continue.
```

...for a complete human being
Texas
International college

Lab no: 4 **Date:2081/09/**

Title: Menu based program to show the basic operation of linear Queue

---

A **linear queue** is a data structure that organizes elements sequentially, following the **FIFO (First In, First Out)** principle, where insertion occurs at the rear and deletion occurs at the front.

Basic operations of a linear queue

1. Enqueue
    - Adds an element to the rear of the queue
    - If the queue is not full, the new element added at the rear position, and the rear pointer is incremented.
2. Dequeue
    - Removes an element from the front of the queue.
    - If the queue is not empty, the element at the front is removed, and the front pointer is incremented.
3. isEmpty
    - Checks whether the queue is empty.
    - Returns true if the rear pointer is equal to the rear pointer.
4. isFull
    - Checks whether the queue is full.
    - Returns true if the rear pointer has reached the maximum size of the queue

5. Front
    - Returns the front element of the queue without removing it.
    - If the queue is not empty, returns the element at the front pointer.

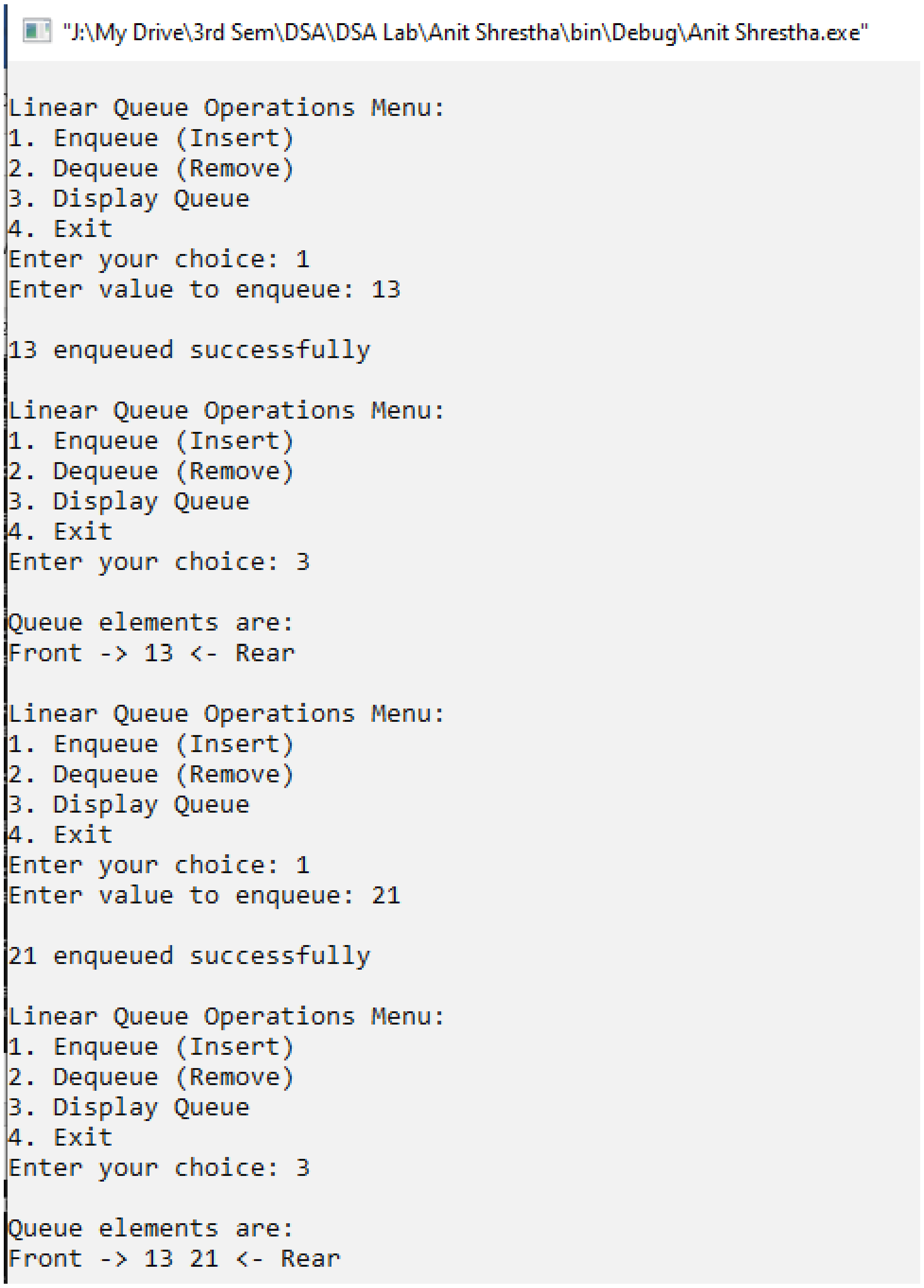
Programming Language: C

IDE: Code Blocks

Source code

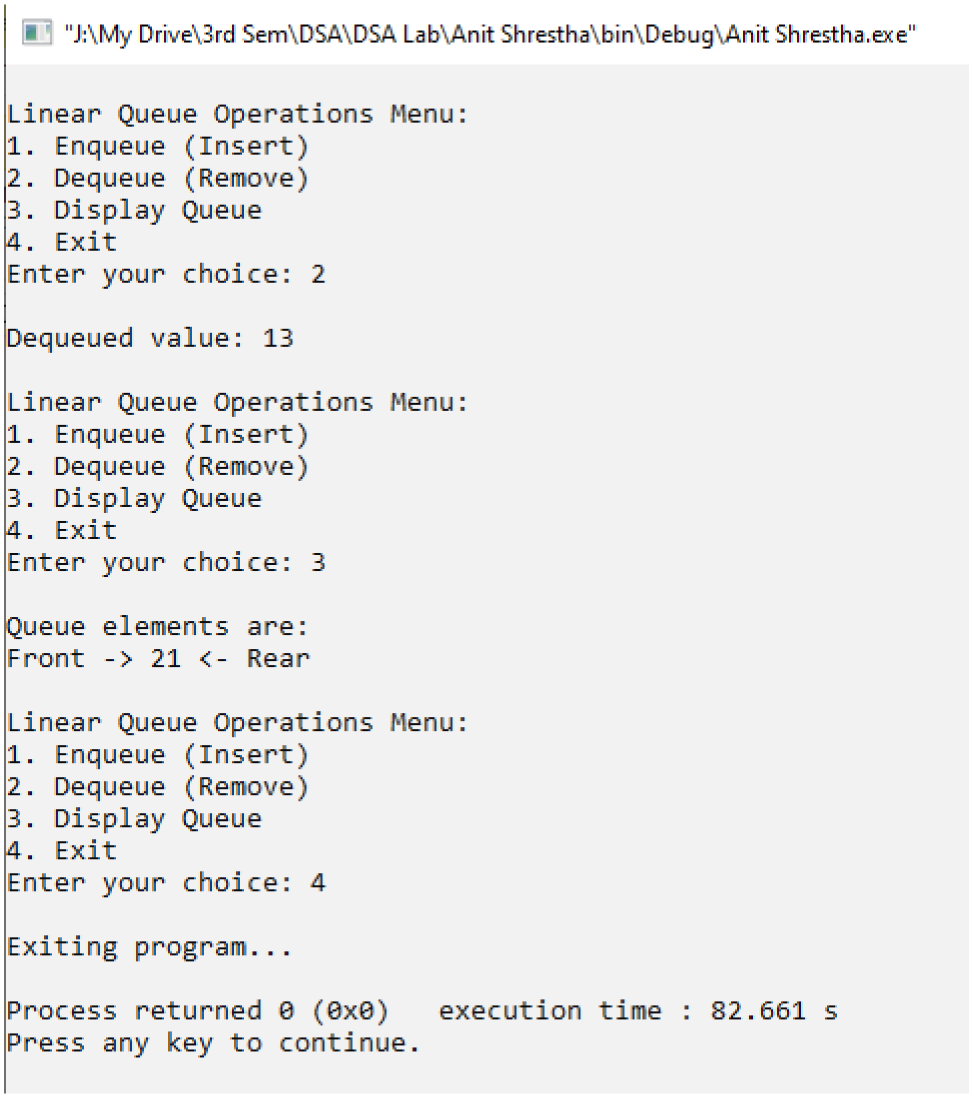
```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 5

// Structure for Queue
struct Queue {
    int items[MAX_SIZE];
    int front;
    int rear;
};

// Function to initialize queue
void initializeQueue(struct Queue *q) {
    q->front = -1;
    q->rear = -1;
}

// Function to check if queue is full
int isFull(struct Queue *q) {
    return q->rear == MAX_SIZE - 1;
}

// Function to check if queue is empty
int isEmpty(struct Queue *q) {
    return q->front == -1 || q->front > q->rear;
}

// Function to enqueue (insert) element
void enqueue(struct Queue *q, int value) {
    if (isFull(q)) {
        printf("\nQueue Overflow! Cannot enqueue %d\n", value);
        return;
    }

    if (q->front == -1) {
        q->front = 0;
    }

    q->rear++;
    q->items[q->rear] = value;
    printf("\n%d enqueued successfully\n", value);
}

// Function to dequeue (remove) element
int dequeue(struct Queue *q) {
    if (isEmpty(q)) {
        printf("\nQueue Underflow! Queue is empty\n");
        return -1;
    }

    int value = q->items[q->front];
    q->front++;
    // Reset queue if it becomes empty
    if (q->front > q->rear) {
        initializeQueue(q);
    }

    return value;
}

// Function to display the queue
void display(struct Queue *q) {
    if (isEmpty(q)) {
        printf("\nQueue is empty!\n");
        return;
    }

    printf("\nQueue elements are:\n");
    printf("Front -> ");
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d ", q->items[i]);
    }
    printf("<- Rear\n");
}

int main() {
    struct Queue queue;
    initializeQueue(&queue);

    int choice, value;

    do {
        printf("\nLinear Queue Operations Menu:\n");
        printf("1. Enqueue (Insert)\n");
        printf("2. Dequeue (Remove)\n");
        printf("3. Display Queue\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch(choice) {
            case 1:
                printf("Enter value to enqueue: ");
                scanf("%d", &value);
                enqueue(&queue, value);
                break;

            case 2:
                value = dequeue(&queue);
                if (value != -1) {
                    printf("\nDequeued value: %d\n", value);
                }
                break;

            case 3:
                display(&queue);
                break;

            case 4:
                printf("\nExiting program...\n");
                break;

            default:
                printf("\nInvalid choice! Please try again.\n");
        }
    } while(choice != 4);

    return 0;
}
```

Output:



"J:\My Drive\3rd Sem\DSA\DSA Lab\Anit Shrestha\bin\Debug\Anit Shrestha.exe"

```
Linear Queue Operations Menu:
1. Enqueue (Insert)
2. Dequeue (Remove)
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to enqueue: 13

13 enqueued successfully

Linear Queue Operations Menu:
1. Enqueue (Insert)
2. Dequeue (Remove)
3. Display Queue
4. Exit
Enter your choice: 3

Queue elements are:
Front -> 13 <- Rear

Linear Queue Operations Menu:
1. Enqueue (Insert)
2. Dequeue (Remove)
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to enqueue: 21

21 enqueued successfully

Linear Queue Operations Menu:
1. Enqueue (Insert)
2. Dequeue (Remove)
3. Display Queue
4. Exit
Enter your choice: 3

Queue elements are:
Front -> 13 21 <- Rear
```



```
"J:\My Drive\3rd Sem\DSA\DSA Lab\Anit Shrestha\bin\Debug\Anit Shrestha.exe"

Linear Queue Operations Menu:
1. Enqueue (Insert)
2. Dequeue (Remove)
3. Display Queue
4. Exit
Enter your choice: 2

Dequeued value: 13

Linear Queue Operations Menu:
1. Enqueue (Insert)
2. Dequeue (Remove)
3. Display Queue
4. Exit
Enter your choice: 3

Queue elements are:
Front -> 21 <- Rear

Linear Queue Operations Menu:
1. Enqueue (Insert)
2. Dequeue (Remove)
3. Display Queue
4. Exit
Enter your choice: 4

Exiting program...

Process returned 0 (0x0)   execution time : 82.661 s
Press any key to continue.
```

Lab no: 05 **Date:2081/09/**

Title: Write a recursive program to calculate the factorial and Fibonacci sequence for user input value.

Factorial:

In mathematics, the factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n.

For example,

5! = 5 * 4 * 3 * 2 * 1 = 120.

Fibonacci sequence:

The Fibonacci numbers or Fibonacci sequence are the numbers in which every element is the sum of previous two elements.

Example:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55...

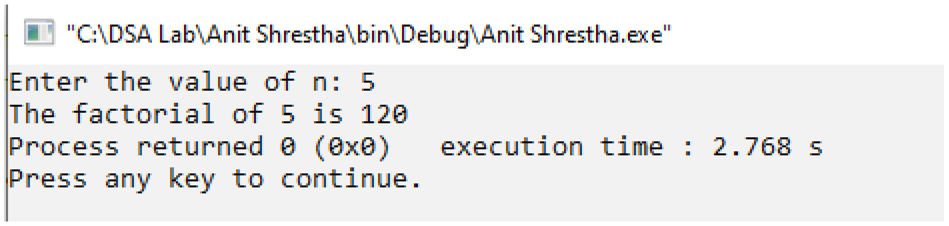
Programming Language: C

IDE: Code Blocks

Source Code

```
#include <stdio.h>
 #include <stdlib.h>

 int fact(int n)  // Define the type of 'n' as int
{
    if (n == 0 || n == 1)
        return 1;
    else
        return n * fact(n - 1);
 }

 int main()
{
    system("color f0");
    int n, fact1;
    printf("Enter the value of n: ");
    scanf("%d", &n);
    fact1 = fact(n);

    printf("The factorial of %d is %d", n, fact1);
    return 0;
 }
```

Output:



"C:\DSA Lab\Anit Shrestha\bin\Debug\Anit Shrestha.exe"

```
Enter the value of n: 5
The factorial of 5 is 120
Process returned 0 (0x0)   execution time : 2.768 s
Press any key to continue.
```

Lab no: 06 **Date:2081/09/**

Title: Write a recursive program to find the GCD of user input integers.

GCD:

The Greatest Common Divisor (GCD) of two integers is the largest positive integer that divides both number without leaving a remainder. For example, the GCD of 8 and 12 is 4, as 4 is the largest number that divides both 8 and 12 evenly.

A common method to find the GCD of two number is the Euclidean Algorithm which is based on the principle that the GCD of two numbers also divides their difference.

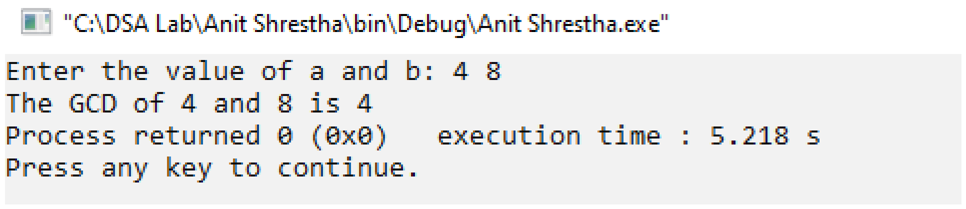
Programming Language: C

IDE: Code Blocks

Source Code

```
#include <stdio.h>
 #include <stdlib.h>

 int gcd(int a, int b)  // Define the types of 'a' and 'b' as int
{
    if (b == 0)
        return a;
    else
        return gcd(b, a % b);
 }

 int main()
{
    system("color f0");
    int a, b, n;
    printf("Enter the value of a and b: ");
    scanf("%d %d", &a, &b);
    n = gcd(a, b);

    printf("The GCD of %d and %d is %d", a, b, n);
    return 0;
 }
```

Output:



"C:\DSA Lab\Anit Shrestha\bin\Debug\Anit Shrestha.exe"

```
Enter the value of a and b: 4 8
The GCD of 4 and 8 is 4
Process returned 0 (0x0)   execution time : 5.218 s
Press any key to continue.
```

...for a complete human being

Texas

International college

Lab no: 07 **Date:2081/09/**

Title: Write a program to print all the moves required for Tower of Hanoi problem for user input no of Disk.

The tower of Hanoi is a classic problem in computer science and mathematics that involves moving a set of disks from one peg to another, following specific rules. The rules are:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty peg.
3. No larger disk may be placed on top of a smaller disk.

The recursive solutions to the Tower of Hanoi problems involve moving disks between three pegs (A, B, C) as follows:

1. Move (n-1) disks from peg A to peg B using peg C as an auxiliary peg.
2. Move the nth disk from peg A to peg C.
3. Move the (n-1) disks from peg B to peg C using peg A as an auxiliary peg.

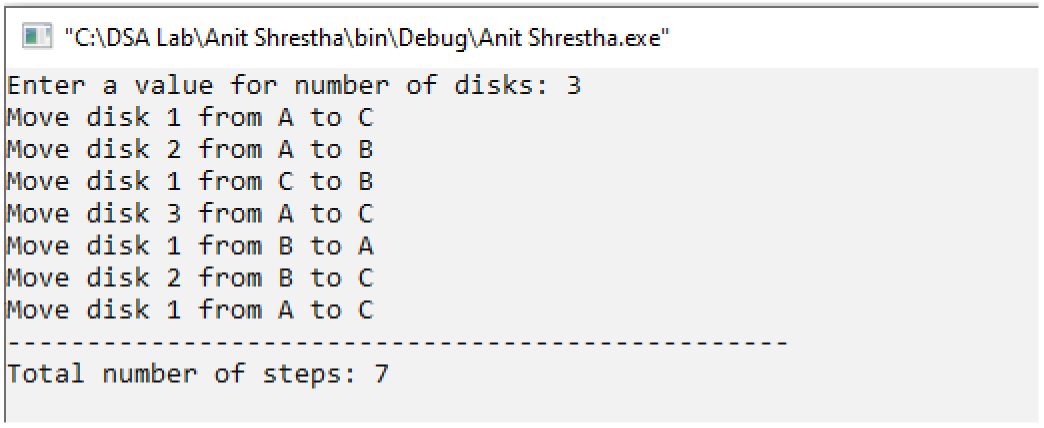
Programming Language: C

IDE: Code Blocks

Source Code

```
#include <stdio.h>

 int stepCount = 0;  // Global variable to count the number of steps

 void TOH(int n, char a, char b, char c)  // Define 'n' as int
{
    if (n > 0) {
        TOH(n - 1, a, c, b);  // Move n-1 disks from source (a) to auxiliary (b)
        printf("Move disk %d from %c to %c\n", n, a, c);  // Move disk n from source (a) to
destination (c)
        stepCount++;  // Increment step count after each move
        TOH(n - 1, b, a, c);  // Move n-1 disks from auxiliary (b) to destination (c)
    }
 }

 int main()
{
    system("color f0");
    int n;
    char a = 'A', b = 'B', c = 'C';

    printf("Enter a value for number of disks: ");
    scanf("%d", &n);

    TOH(n, a, b, c);
    printf("-------------------------------------------------\n");
    printf("Total number of steps: %d\n", stepCount);  // Display total steps

    return 0;
 }
```

Output:



"C:\DSA Lab\Anit Shrestha\bin\Debug\Anit Shrestha.exe"

```
Enter a value for number of disks: 3
Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C
-------------------------------------------------
Total number of steps: 7
```