

LAB-06: Fibonacci Sequence

Theory:

The Fibonacci sequence, also known as Fibonacci numbers, is defined as the sequence of numbers in which each number in the sequence is equal to the sum of two numbers before it.

The Fibonacci sequence is given as:

Fibonacci sequence = 0, 1, 1, 2, 3, 5, 8, 13, 21...

Here, the third term '1' is obtained by adding the first and second term. (i.e., $0+1 = 1$)

Similarly,

'2' is obtained by adding the second and third term ($1+1 = 2$)

'3' is obtained by adding the third and fourth term ($1+2=3$) and so on.

For example, the next term after 21 can be found by adding 13 and 21.

The Fibonacci sequence of numbers " F_n " is defined using the recursive relation with the seed values $F_0=0$ and $F_1=1$:

$$F_n = F_{n-1} + F_{n-2}$$

Fibonacci sequence using iteration:

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

```
int main() {
    int n, first = 0, second = 1, result, i;
    printf("Give an input upto you want to print sequence : \n");
    scanf("%d", & n);
    printf("Fibonacci Series is:\n");
    for (i = 0; i < n; i++) {
        if (i <= 1)
            result = i;
        else {
            result = first + second;
            first = second;
            second = result;
        }
        printf("%d ", result);
    }
    return 0;
}
```

Fibonacci sequence using recursion:

```
#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("Please give an input upto you want to print sequence : \n");

scanf("%d", & n);

int i;

printf("Fibonacci sequence of %d: ", n);

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

    printf("%d ", fibonacci(i));

}

return 0;

}

```

Fibonacci sequence using tail recursion:

```

#include <stdio.h>

int fib(int n, int a, int b) {
    if (n == 0)
        return a;
    else if (n == 1)
        return b;
    else
        return fib(n - 1, b, a + b);
}

int main() {
    int n = 9, a = 0, b = 1;
    printf("%d ", fib(n, a, b));
    return 0;
}

```

LAB-07: Greatest Common Divisor (GCD)

GCD using iteration:

```
include<stdio.h>
```

```
#include<conio.h>
```

```
int gcd(int, int);
```

```
int main() {
```

```
    int a, b;
```

```
    int gcd1;
```

```
    printf("Enter the first number: ");
```

```
    scanf("%d", &a);
```

```
    printf("\nEnter second number:");
```

```
    scanf("%d", &b);
```

```
    gcd1 = gcd(a, b);
```

```
    printf("The gcd of two numbers %d and %d is: %d", a, b, gcd1);
```

```
}
```

```
int gcd(int a, int b) {
```

```
    int remainder;
```

```
    if (a == 0) {
```

```
        return (b);
```

```
    } else if (b == 0) {
```

```
        return (a);
```

```
    } else {
```

```
        while (b != 0) {
```

```
            remainder = a % b;
```

```
            a = b;
```

```
            b = remainder;
```

```
        }
```

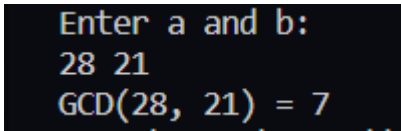
```
        return (a);
```

```
    }
```

```
}
```

GCD using recursion:

```
#include <stdio.h>
int gcd(int x, int y) {
    if (y == 0)
        return x;
    else
        gcd(y, x % y);
}#int main()
{
    int a, b, g;
    printf("Enter a and b:\n");
    scanf("%d%d", &a, &b);
    g = gcd(a, b);
    if(g < 0)
    {
        g *= -1;
    }
    printf("GCD(%d, %d) = %d\n", a, b, g);
    return 0;
}
```

A screenshot of a terminal window showing the execution of the GCD program. The text is displayed in a monospaced font on a dark background. It shows the prompt 'Enter a and b:', followed by the user input '28 21', and the final output 'GCD(28, 21) = 7'.

```
Enter a and b:
28 21
GCD(28, 21) = 7
```

LAB-08: Tower of Hanoi (TOH)

Tower of Hanoi (TOH) using iteration:

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

struct Stack {
    unsigned capacity;
    int top;
    int * array;
};

// function to create a stack of given capacity.
struct Stack * createStack(unsigned capacity) {
    struct Stack * stack =
        (struct Stack * ) malloc(sizeof(struct Stack));
    stack -> capacity = capacity;
    stack -> top = -1;
    stack -> array =
        (int * ) malloc(stack -> capacity * sizeof(int));
    return stack;
}

// Stack is full when top is equal to the last index
int isFull(struct Stack * stack) {
    return (stack -> top == stack -> capacity - 1);
}

// Stack is empty when top is equal to -1
int isEmpty(struct Stack * stack) {
    return (stack -> top == -1);
}

// Function to add an item to stack. It increases
// top by 1
void push(struct Stack * stack, int item) {
    if (isFull(stack))
        return;
    stack -> array[++stack -> top] = item;
}
```

```

// Function to remove an item from stack. It
// decreases top by 1
int pop(struct Stack * stack) {
    if (isEmpty(stack))
        return INT_MIN;
    return stack -> array[stack -> top--];
}

//Function to show the movement of disks
void moveDisk(char fromPeg, char toPeg, int disk) {
    printf("Move the disk %d from \'%c\' to \'%c\'\\n", disk, fromPeg, toPeg);
}

// Function to implement legal movement between
// two poles
void moveDisksBetweenTwoPoles(struct Stack * src, struct Stack * dest, char s, char d) {
    int pole1TopDisk = pop(src);
    int pole2TopDisk = pop(dest);

    // When pole 1 is empty
    if (pole1TopDisk == INT_MIN) {
        push(src, pole2TopDisk);
        moveDisk(d, s, pole2TopDisk);
    }

    // When pole2 pole is empty
    else if (pole2TopDisk == INT_MIN) {
        push(dest, pole1TopDisk);
        moveDisk(s, d, pole1TopDisk);
    }

    // When top disk of pole1 > top disk of pole2
    else if (pole1TopDisk > pole2TopDisk) {
        push(src, pole1TopDisk);
        push(src, pole2TopDisk);
        moveDisk(d, s, pole2TopDisk);
    }

    // When top disk of pole1 < top disk of pole2
    else {
        push(dest, pole2TopDisk);
        push(dest, pole1TopDisk);
        moveDisk(s, d, pole1TopDisk);
    }
}

```

```
}  
}
```

//Function to implement TOH puzzle

```
void tohIterative(int num_of_disks, struct Stack * src, struct Stack * aux, struct Stack * dest)
```

```
{
```

```
    int i, total_num_of_moves;
```

```
    char s = 'J', d = 'K', a = 'L';
```

```
    //If number of disks is even, then interchange
```

```
    //destination pole and auxiliary pole
```

```
    if (num_of_disks % 2 == 0) {
```

```
        char temp = d;
```

```
        d = a;
```

```
        a = temp;
```

```
    }
```

```
    total_num_of_moves = pow(2, num_of_disks) - 1;
```

```
    //Larger disks will be pushed first
```

```
    for (i = num_of_disks; i >= 1; i--)
```

```
        push(src, i);
```

```
    for (i = 1; i <= total_num_of_moves; i++) {
```

```
        if (i % 3 == 1)
```

```
            moveDisksBetweenTwoPoles(src, dest, s, d);
```

```
        else if (i % 3 == 2)
```

```
            moveDisksBetweenTwoPoles(src, aux, s, a);
```

```
        else if (i % 3 == 0)
```

```
            moveDisksBetweenTwoPoles(aux, dest, a, d);
```

```
    }
```

```
}
```

// Driver Program

```
int main() {
```

```
    // Input: number of disks
```

```
    unsigned num_of_disks;
```

```
    printf("Enter number of disks:");
```

```
    scanf("%d", & num_of_disks);
```

```
    struct Stack * src, * dest, * aux;
```



```

// Create three stacks of size 'num_of_disks'
// to hold the disks
src = createStack(num_of_disks);
aux = createStack(num_of_disks);
dest = createStack(num_of_disks);

tohIterative(num_of_disks, src, aux, dest);
return 0;
}

```

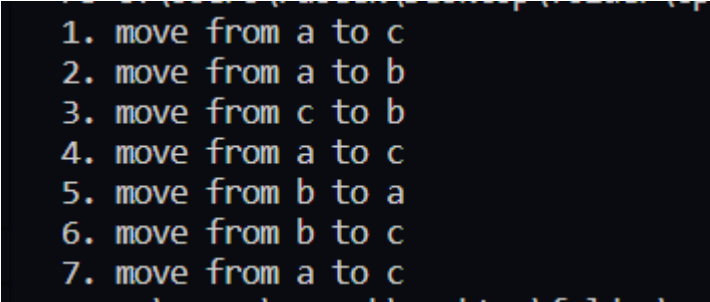
Tower of Hanoi (TOH) using recursion:

```

#include <stdio.h>
int counter = 1;
void hanoi(int n,char source,char dest,char aux){
    if (n == 0){
        return;
    }
    hanoi(n-1,source,aux,dest);
    printf("%d. move from %c to %c\n",counter++,source,dest);
    hanoi(n-1,aux,dest,source);
}

int main(){
    hanoi(3,'a','c','b');
}

```



```

1. move from a to c
2. move from a to b
3. move from c to b
4. move from a to c
5. move from b to a
6. move from b to c
7. move from a to c

```