Learn Basic Recursion:

Problem 1: Introduction to Recursion - Understand Recursion by printing something N times.

What is Recursion?

It is a phenomenon when a function calls itself indefinitely until a specified condition is fulfilled.

If there is no condition to stop the recursive calls, the calls will run indefinitely until the stack runs out of memory (stack overflow)

What is stack overflow in recursion?

Whenever recursion calls are executed, they're simultaneously stored in recursion stack where they wait for the completion of the recursive function. A recursive function can only be completed if a base condition is fulfilled and the control returns to the parent function.

But, when there is no base condition given for a particular recursive function, it gets called indefinitely which results in stack overflow i.e, exceeding the memory limit of

the recursion stack and hence the program terminates giving a segmentation fault error.

Base condition

It is a condition that is written in a recursive function in order for it to get completed and not to run infinitely. After encountering the base condition, the function terminates and returns back to its parent function simultaneously.

Problem 2: Print Name N times using Recursion.

Solution

```
public void printNTimes(int N) {
    if (n==0)                    // base condition
    { return ;
     }

     print("santosh")            // printing name
     printNTimes(N-1);           // recursive call
}
```

Problem 3: Print from 1 to N using recursion.

print integers from 1 to N without using

any global variable but by using function parameters.

Solution :

For this problem, we're going to be using a function along with parameters that get incremented with each function call through which we can keep track of the integer count while printing.

Pseudocode :

```
void func(i, n)
{
    if (i>n) return;      // base condition.
    print(i);             // printing number
    func(i+1, n)          // recursive function call
}

main()
{ int n;
    input(n);             // getting 'n'
    func(1, n)            // call recursive function.
}
```

In this pseudocode, we first call the function when the value of i is 1 and then print the

value of i and increment i by 1 inside the parameter of the function and make a call again. But we know this will go on forever as i will be increasing continuosly after every function call. So, to avoid this we put a base condition that if i exceeds n, then simply terminate the current recursive call and return to the previous call.

In this way, all the integers from 1 to N would get printed and as soon as we exceed the count of printing by n, the function terminates.

Approach 2: using backtracking.

In the previous approach, we used forward recursion but in this approach, we will be using backward recursion. The only change from the previous approach here will be that the print line would be kept after the function call inside the recursive function contrary to the previous approach.

pseudocode:

```
void func (i, N)
{
  if (i< 1) return;
  func (i-1, N);
   Print (i) ;
}

main c)
{   int  n;
    input (n);
    func (n, n);
}
```

We can clearly see, We first call the function when the value of i is N and then make a call again inside this function for printing (n-1) integers and after this we print N. But we know that this will go on forever as i will be decreasing continuosly after every function call. So, to avoid this we put a base condition that if i is less than 1, then simply terminate the current recursive call and return to the previous call).

Problem 4: Print from N to 1 using Recursion.

Solution:

you can use the same approach as used in the solution of Problem 3.

Problem 5: Sum of first N natural numbers.

Given a number 'N', find out the sum of first N natural numbers.

Solution:

Approach 1: Using for loop.

Algorithm:

1. Initialize a variable sum = 0;
2. use a for loop (i=1; i<n; i++)
3. each iteration of loop, sum = sum + i;
4. return sum.

T.C. = O(n)

S.C = O(1)

Approach 2: Using formula.

We can use the formula for the sum of N number $N(N+1)/2$.

Approach 3: Using Recursion

There are two ways of calculating the sum

of first N natural Numbers:

- Parameterized way
- Functional way.

1) Parameterized way:

In this approach instead of using a global variable for calculating the sum, we pass the sum in the parameters of the function each time we add an integer to it during the function call. The sum gets incremented by an ith integer and i get decremented by 1 in each function call. At the end when i becomes less than 1, we simply return the calculated sum until that point.

pseudocode.

```
void func(i, sum)
 {
    if (i<1)
    { print(sum);
      return;
    }
   func(i-1, sum+i)
   }
```

```
mainc) {
 input (n);
  func (n,0)
 }
```

2. Functional way:

This approach is a lot simpler than the parameterized way.

Sum(N) = N + Sum(N-1);

pseudocode:

```
int  func(n) {
       if (n==0)
       { return 0;
         }

 return  n + func(n-1);
  }

 main()
 {  input (n);
     func (n);
  }
```

Problem 6: Factorial of a number: Iterative & Recursive.

Given a number x, print its factorial.

To obtain the factorial of a number, it has to be multiplied by all the whole numbers preceding it.

More precisely $x! = x*(x-1)*(x-2)\ldots 1$.

Note :- x is always a positive number.

Solution :-

You can use the same approach as given in Problem 5.

pseudo code

1.
```
int factorialN(int N) {
  int factorial = 1;
for (int i=1; i<=n; i++)
   { factorial = factorial * i ;
    }
return factorial;
 }
```

2.
```
int factorial(int n) {
   if (n==1)
    { return 1;}
return n * factorial(n-1) ;
  }
```

Problem 7: Reverse an array.

You are given an array, the task is to reverse an array and print it.

Solution:

Approach 1: Iterative approach:

Create another array of same size.

start a for loop with (i=0; i<n; i++)

result[i] = nums[n-1-i];

as you can see, for first element in result array we are assigning the last index of given array by doing n-1-i. as i will increase, the index of given array will decrease by 1 each time until 0.

T.C. = O(n) // n iterations.

S.C. = O(n) // Result Array of size (n).

Approach 2: Space optimized iterative method.

Unlike the previous method we use the same array to obtain the result.

1. Keep a pointer P_1 at first index and another P_2 at last index of the array.
2. Swap the elements pointed by P_1 and P_2, Past swapping increment P_1 and decrement P_2.

3. This process is repeated for only the first n/2 elements where n is the length of array.

Approach 3: Recursive Method.

The recursive method has an approach almost similar to the iterative one. The approach has been broken down into following steps for simplicity.

1. Create a function that takes an array, start index and end index as parameters.
2. swap the elements present at the start & end.
3. The portion of the array left to be reversed is arr[start+1, end-1]. Make a recursive call to reverse the rest of the array. while calling recursion pass start+1, ends-1 as parameters for the shrunk array Repeat step 2.
4. continue recursion as long as the 'start<end' condition is satisfied. This is the base case for our recursion.

Approach 4:

C++ and java have inbuit functions to reverse an array.

Problem 8: check if the given string is pallindrome or not.

Solution: you can use methods given in Problem 7 to reverse the string. and check.

Problem 9: Print Fibonacci series upto Nth term.

Solution:

Approach 1: As we know fib(i) = fib(i-1) + fib(i-2). simply iterate and go on calculating the ith term in the series.

1. Take an array say fib of size n+1. The 0th term and 1st term are 0 and 1 respectively. So fib(0) = 0 and fib(1) = 1
2. Now iterate from 2 to n and calculate fib(n) fib(n) = fib(n-1) + fib(n-2).
3. Then print fib(0) + fib(1) ..... fib(n).

T.C. = O(n)

S.C = O(n)

Approach 2: Recursive approach

Fibonacci(N) = fibonacci(n-1) + fibonacci(n-2)

T.C. = 2^n

S.C = O(n)