Programming for problem solving using C

123ES

Ms. Urvashi Gohil
Assistant Professor

Birla Vishvakarma Mahavidhyalaya (BVM) Engineering College
Department of Computer Engineering

March 17, 2025

Outline of the course :

Introduction

Types of Function

Recurrsion

Introduction

About Function

Parameters : Formal Parameters
Arguments: Actual Parameters

Main→calling function
User defined fucnt→ called function

Step-1 function declaration
Step-2 int main() → call the function
Step-3 function definition

Return type→ What value function is returning (int, float, char, double, void)

When call the funct in main→ it will find the definition→ then it match the definition with declaration (return type, funct name, parameters)→if match trun the funct→Closes braces, control return to the main

Type-1 Without Return type without argument

abc > C fun.c > sum()

```c
#include<stdio.h>
void sum(); //func declaration
int main(){
    sum(); // fun calling
}
void sum(){ // fun defination
    int a, b;
    printf("Enter a & b:");
    scanf("%d %d", &a, &b);
    printf("The sum of a & b is: %d", a+b);
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\code1\abc\output> cd 'f:\code1\abc\output'
PS F:\code1\abc\output> & .\'fun.exe'
Enter a & b:3 2
The sum of a & b is: 5
PS F:\code1\abc\output>
```

Type-2 With Return type without argument

abc > C w.c > sum()

```c
#include<stdio.h>

int sum();

int main(){
   int s= sum();
   printf("%d",s);
    }

    int sum(){
      int num1, num2, sum;
      printf("Enter num1 and num2:");
      scanf("%d %d" , &num1, &num2);
      sum = num1+num2;
      return sum;
   }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Enter num1 and num2:5 6
11
PS F:\code1\abc\output>
```

Why Function declaration is imp

abc > C fu.c > funct()

```
1   #include<stdio.h>
2   int main(){
3       char ch;
4       ch = funct();
5       printf("%c",ch);
6   }
7
8   char funct(){
9       char c;
10      printf("Enter the char:");
11      scanf("%c",c);
12      return c;
13  }
```

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
has type 'int' [-Wformat=]
   11 |     scanf("%c",c);
      |            ~^  ~
      |             |  |
      |             |  int
      |             char *
```

abc > C fu.c > funct()

```
1   #include<stdio.h>
2   char funct();   //declare funct
3   int main(){
4       char ch;
5       ch = funct();
6       printf("%c",ch);
7   }
8   char funct(){ //define funct
9       char c;
10      printf("Enter the char:");
11      scanf("%c",c);
12      return c;
13  }
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\code1> cd 'f:\code1\abc\output'
PS F:\code1\abc\output> & .\'fu.exe'
Enter the char:D
PS F:\code1\abc\output>
```

It takes int as an implicit return type, if we don't declare funct with char

Type-3 Without Return type with argument

abc > C fu.c > main()

```c
#include<stdio.h>
void sum(int, int);
int main(){
    sum(4,5);
    printf("\nhello function!");
}
void sum(int a, int b){
    printf("The sum of a & b is: %d ", a+b);
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\code1\abc\output> cd 'f:\code1\abc\output'
PS F:\code1\abc\output> & .\'fu.exe'
The sum of a & b is: 9
hello function!
PS F:\code1\abc\output>
```

Type-3 Without Return type with argument- Array as parameter

abc > C fu.c > ...

```c
#include<stdio.h>
void display(int arr[], int size);
int main(){
  int arr[] = { 3, 4, 5, 6};
  display(arr,4);
}
void display(int arr[], int size){
    for (int i= 0;i<size; i++){
        printf("%d ", arr[i]);
    }
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\code1\abc\output> cd 'f:\code1\abc\output'
PS F:\code1\abc\output> & .\'fu.exe'
3 4 5 6
PS F:\code1\abc\output>
```

Type-3 Without Return type with argument- String as parameter


```
11  #include<string.h>
12  void display(char ch[]);
13  int main(){
14      char str[100];
15
16      display(str);
17  }
18  void display(char ch[]){
19      printf("Enter the string:");
20      gets(ch);
21      puts(ch);
22  }
23
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\code1\abc\output> cd 'f:\code1\abc\output'
PS F:\code1\abc\output> & .\'fu.exe'
Enter the string:BVM Engineering
BVM Engineering
PS F:\code1\abc\output>
```

Type-4 With Return type with argument

abc > C fu.c > main()

```c
#include<stdio.h>
int sum(int, int);
int main(){
    int s ;
    s = sum(4,3);
    printf("The sum is : %d", s);
}
int sum(int a, int b){
    return a+b;
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\code1\abc\output> cd 'f:\code1\abc\output'
PS F:\code1\abc\output> & .\'fu.exe'
The sum is : 7
PS F:\code1\abc\output>
```

GCD

abc > C gcdfun.c > main()

```c
#include<stdio.h>
int findGCD(int, int);
int main(){
    int x, y, result;
    printf("Enter x & Y: ");
    scanf("%d %d", &x, &y);
    result = findGCD(x,y);
    printf("%d", result);
}
int findGCD(int a, int b){
    int temp;
    for(;b!= 0;){
        temp = b;
        b = a %b;
        a = temp;
    }
    return a;
}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\code1\abc\output> cd 'f:\code1\abc\output'
PS F:\code1\abc\output> & .\'gcdfun.exe'
Enter x & Y: 32 128
32
PS F:\code1\abc\output>
```

Prime

abc > C prime.c > main()

```c
#include<stdio.h>
int isPrime(int num);
int main(){
    int number;
    printf("Enter the number:");
    scanf("%d", &number);
    if(isPrime(number)){
        printf("The number is prime");
    }
    else printf("The number is not prime");

}
int isPrime(int num) {
        if (num < 2)
            return 0;

        int i = 2;
        while (i < num) {
            if (num % i == 0)
                return 0; // Found a divisor, not prime
            i++;
        }
        return 1; // No divisors found, number is prime
    }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

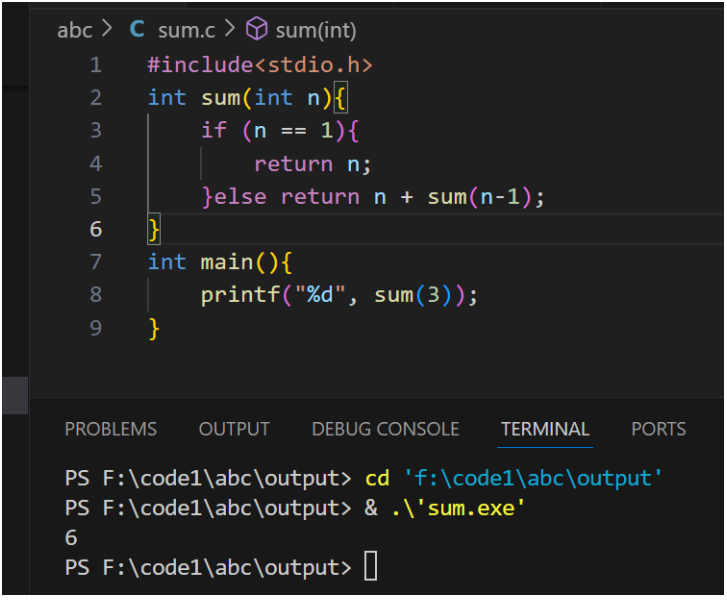
```
PS F:\code1\abc\output> & .\'prime.exe'
Enter the number:23
The number is prime
```

Recursion


```
return_type function_name(parameters) {
    // Base case(s) - the condition where the function stops calling itself
    if (base_condition) {
        // Return some value or perform some action
    }
    // Recursive case(s) - where the function calls itself
    else {
        // Recursive call with modified parameters
        function_name(modified_parameters);
    }
}
```

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

Recursion- sum of n number



Recursion fact example

abc > C chocol.c > main()

```c
#include<stdio.h>
int factorial(int n){
    if(n == 0 || n ==1){
        return 1;
    }else
        return n * factorial(n-1);
}
int main(){
    int num;
    printf("Enter the number");
    scanf("%d", &num);
    int fact = factorial(num);
    printf("%d", fact);
}
```

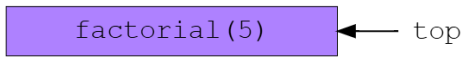
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\code1\abc\output> cd 'f:\code1\abc\output'
PS F:\code1\abc\output> & .\'chocol.exe'
Enter the number 5
120
PS F:\code1\abc\output>
```

Recursion fact example

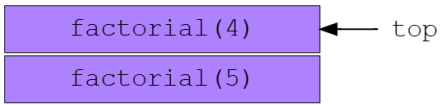
The first call to the function factorial()
lies at the bottom of the stack.

This is also the function that will return at the very end.



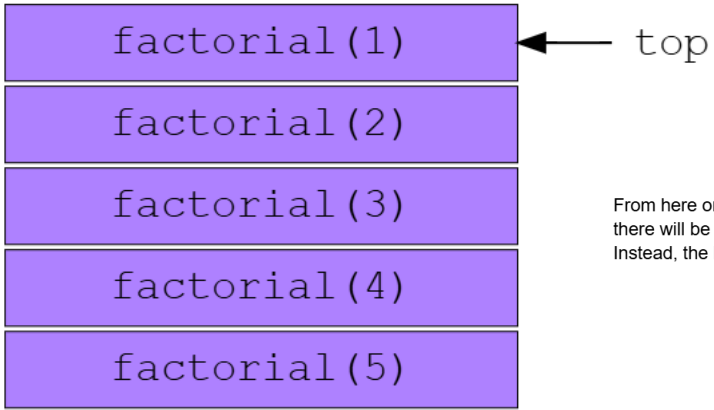
Recursion fact example

factorial(5) function calls another instance of the same function, but this time target - 1 is passed to the function i.e., factorial(4)



Recursion fact example

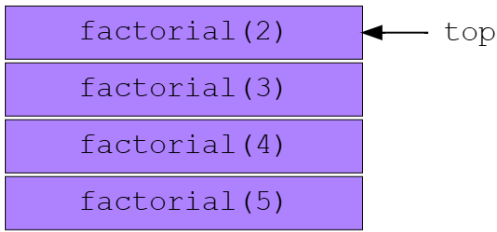
Each time a function calls another instance of the function it is pushed at the top of the stack.



From here onwards, i.e., after factorial(1) is called there will be no further function calls.
Instead, the base case is satisfied and the function returns 1.

Recursion fact example

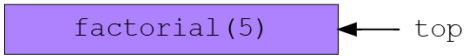
Each function will now return its respective value.



Recursion fact example

This is the beauty of recursion.

Each child function call returns result to its parent function.
In the end the result is accumulated and returned.



Recursion ProblemStatement

The Deadly Sin

Meliodas and Ban are fighting over chocolates. Meliodas has X chocolates, while Ban has Y. Whoever has lesser number of chocolates eats as many chocolates as he has from the other's collection. This eatfest war continues till either they have the **same number of chocolates**, or atleast one of them is left with **no chocolates**. Can you help Elizabeth predict the total no of chocolates they'll be left with at the end of their war?

Input Format

- First line will contain T, number of testcases. Then the testcases follow.
- Each testcase contains of a single line of input, which contains two integers X, Y, the no of chocolates Meliodas and Ban have, respectively.

Output Format

For each testcase, output in a single line the no of chocolates that remain after Ban and Meliodas stop fighting.

Recursion ProblemStatement

abc > C chocol.c > main()

```c
#include <stdio.h>

int chocolateWar(int X, int Y) {
    if (X == 0 || Y == 0 || X == Y) {
        return X + Y;
    }
    return (X > Y) ? chocolateWar(X - Y, Y) : chocolateWar(X, Y - X);
}

int main() {
    int X, Y;
    printf("Enter chocolates for Meliodas (X) and Ban (Y): ");
    scanf("%d %d", &X, &Y);

    printf("Total chocolates left: %d\n", chocolateWar(X, Y));

}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS F:\code1\abc\output> & .\'chocol.exe'
Enter chocolates for Meliodas (X) and Ban (Y): 10 15
Total chocolates left: 10
PS F:\code1\abc\output>
```

Recursion - indirect


```
3   int n=1;
4   void odd()
5   {
6       if(n <= 10)
7       {
8           printf("%d ", n+1);
9           n++;
10          even();
11      }
12  }
13  void even()
14  {
15      if(n <= 10)
16      {
17          printf("%d ", n-1);
18          n++;
19          odd();
20      }
21  }
22  int main()
23  {
24      odd();
25  }
```

```
PS F:\code1\abc\output> & .\'oddeven.exe'
2 1 4 3 6 5 8 7 10 9
PS F:\code1\abc\output>
```

Recursion - indirect

abc > C indirect.c > main()

```c
#include <stdio.h>
void fun1(int a)
{
    if (a > 0)
    {
        printf("%d ", a); fun2(a - 1);
    }
}
void fun2(int b)
{
    if(b > 0)
    {
        printf("%d ", b);fun1(b - 2);
    }
}
int main ()
{
    int p = 13; fun1 (p);
}
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
13 12 10 9 7 6 4 3 1
PS F:\code1\abc\output>
```

Consider the following C program:

```c
#include <stdio.h>
void fX();
int main() {
   fX();
   return 0;}
```

```c
void fX() {
   char a;
   if ((a = getchar()) != '\n')
       fX();
   if (a != '\n')
       putchar(a);}
```

Assume that the input to the program from the command line is 1234 followed by a newline character. Which one of the following statements is CORRECT?

2. What will be the output of the following C program segment?
Char inchar = 'A';

```
Switch ( inchar ) {
case 'A' : printf ("Choice A\ n") ;
case 'B' :
case 'C' : printf (“Choice B”) ;
case 'D' :
case 'E' :
default : printf ( " No Choice" ) ; }
```

A. **No Choice**
B. **Choice A**
C. **Choice A**
Choice B No Choice
D. **Program gives no output as it is erroneous**

GATE CSE 2012

Consider the following C program. Assume parameters to a function are evaluated from right to left.

```
#include <studio.h>

int g(int p) { printf("%d", p); return p; }

int h(int q) { printf("%d", q); return q; }

void f(int x, int y) {

   g(x);

   h(y);

}

int main() {

   f(g(10),h(20));

}
```

Mid solution