Programming in C

By

Dr Ramkumar Krishnamoorthy

kramdharma@gmail.com

Functions

Unit III

Contents

Function

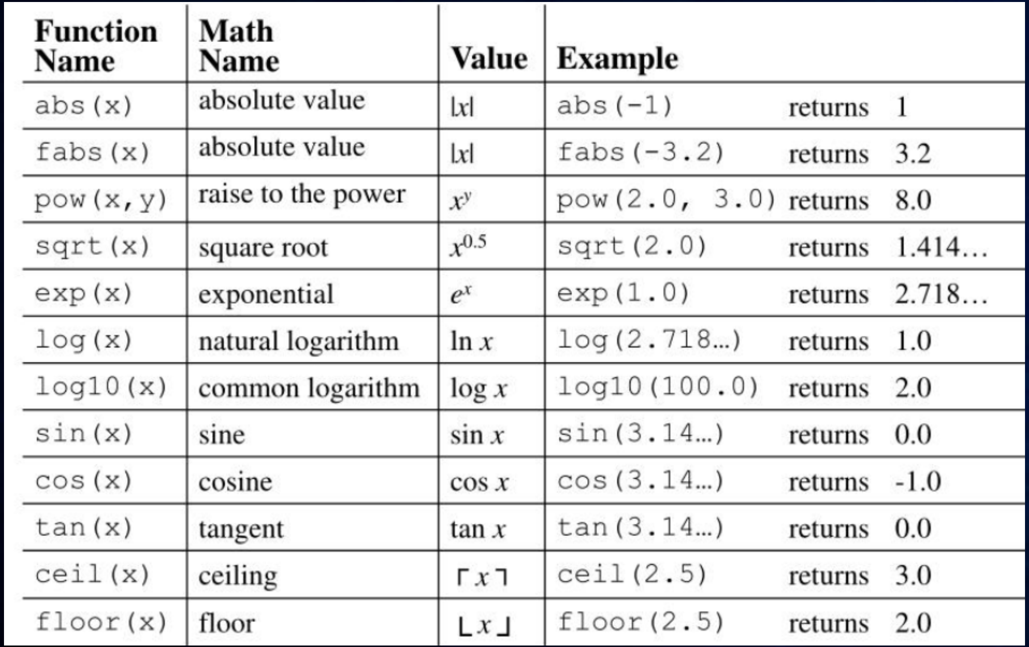
- A function is a named unit of a group of program statements designed to perform a specific task and returns a single value.

- **C language supports two types of functions. They are**
- **i. Built-in or library functions.**
- **ii. User defined functions**

- **Uses of Functions**
- **Repetition – Avoids rewriting**
- **Universal use – Global access**
- **Simple solver – Complexity tasks can be solved**
- **Efficiency – once declaration multiple use**

Built-in Function

A function defined by the C compiler
It is also called readymade functions/predefined function
This type of functions already defined, so user can make use directly
Ex: sqrt(), ceil(), floor(), abs(), pow
In order use above functions user must include **math.h** header file



Function Name	Math Name	Value	Example
`abs(x)`	absolute value	$\lvert x \rvert$	`abs(-1)` returns 1
`fabs(x)`	absolute value	$\lvert x \rvert$	`fabs(-3.2)` returns 3.2
`pow(x,y)`	raise to the power	x^y	`pow(2.0, 3.0)` returns 8.0
`sqrt(x)`	square root	$x^{0.5}$	`sqrt(2.0)` returns 1.414...
`exp(x)`	exponential	e^x	`exp(1.0)` returns 2.718...
`log(x)`	natural logarithm	$\ln x$	`log(2.718...)` returns 1.0
`log10(x)`	common logarithm	$\log x$	`log10(100.0)` returns 2.0
`sin(x)`	sine	$\sin x$	`sin(3.14...)` returns 0.0
`cos(x)`	cosine	$\cos x$	`cos(3.14...)` returns -1.0
`tan(x)`	tangent	$\tan x$	`tan(3.14...)` returns 0.0
`ceil(x)`	ceiling	$\lceil x \rceil$	`ceil(2.5)` returns 3.0
`floor(x)`	floor	$\lfloor x \rfloor$	`floor(2.5)` returns 2.0

Built-in Function Simple Examples

```
#include<stdio.h>
#include<conio.h>
void main()
{
            int a=36, b;
            clrscr();
           b=sqrt(a);
            printf("%d", b);
            getch();

}
Output = 6
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
            int x=90,y=60, z=45;
            clrscr();
            printf("%d", sin(x));
            printf(%f", cos(y));
            printf("%d", tan(z));
            getch();

}
Output = 1   0.5   1
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
            float x=56.79;
            printf("%f", ceil(x));
            printf("%f", floor(x));

}

Output = 57   56
```

Builtin Function – Example

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a=10,b,q,r,s,m,n;
float x=12,y,z,p,t,u,v,o;
clrscr();
b=sqrt(a);
y=cos(x);
z=sin(x);
p=tan(x);
q=abs(-17.63);
r=floor(15.18);
s=ceil(15.1);
t=1/tan(x);
u=1/sin(x);
v=1/cos(x);
m=pow(a,2);
n=pow(a,3);
o=log(x);
printf("Sqrt value:%d\n",b);
printf("Cos value:%f\n",y);
printf("Sin value:%f\n",z);
printf("Tan value:%f\n",p);
 printf("Abs value:%d\n",q);
printf("Floor value:%d\n",r);
printf("Ceil value:%d\n",s);
printf("Cot value:%f\n",t);
printf("Cosec value:%f\n",u);
printf("Sec value:%f\n",v);
printf("Pow value:%d\n",m);
printf("Powc value:%d\n",n);
printf("Log value:%f\n",o);
getch();
                    }
```

Output:

```
Sqrt value:3
Cos value:0.843854
Sin value: -0.536573
Tan value:-0.635860
Abs value:17
Floor value:15
Ceil value:16
Cot value: -1.572673
Cosec value: -1.863680
Sec value:1.185039
Pow value:100
Powc value:1000
Log value:2.484907
```

User Defined Functions

- A function defined by the user
- User can give any name to the function
- User cant set the keyword as the function name

Benefits of Functions

- Programs with functions are compact thus decreasing the number of lines of code to a very large extent.
- Debugging the code is much simpler because errors can be localized and corrected easily.
- Functions increase the readability of the program and helps in proper and good documentation.
- Many other programs may use a function, thus helping us to create our own libraries.

User Defined Functions

- Syntax

```
Returntype function name(list of arguments)
{
        set of statements;
        return exp;
}
```

- Function Parts/Components
  - Function Declaration
  - Function Call
  - Function Body/Definition
- Types
  - No return value no arguments
  - No return type with arguments
  - Return type with arguments
  - Return type without arguments

User Defined Functions

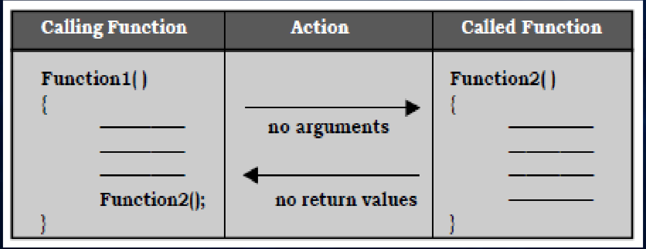
- **Type1: No Return Type without Arguments**
- **Definition: This function did not accept any arguments and not return any values**

Example Program

```
#include<stdio.h>
#include<conio.h>
void demo();        //function declaration
void main()
{
        clrscr();
        printf("\n Before function:");
        demo();   //function calling
        printf("\n After function");
        getch();
}
void demo() //function body
{
        printf("\n *********");
        printf("\n *********");
}
```

Output:

```
Before function
**********
**********
After function
```



User Defined Functions

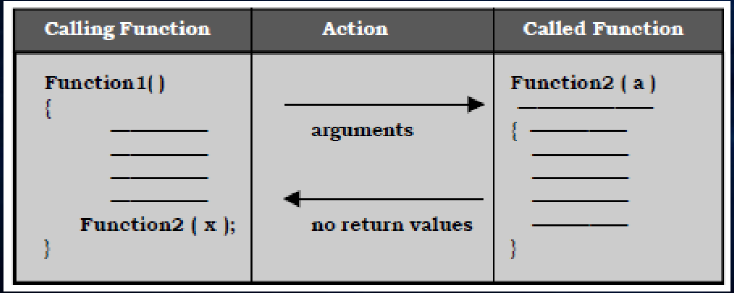
- **Type2: No Return Type with Arguments**
- **Definition: This function takes one or more(some) number of arguments and not return any values**

Example Program

```
#include<stdio.h>
#include<conio.h>
void demo(int,int); //function declaration
void main()
{
        int x=10,u=10;
        clrscr();
        printf("\n Before function:");
        demo(x,u); //function calling
        printf("\n After function");
        getch();
}
void demo(int a,int b) //function definition
{
        int c=a+b;
        printf("\n %d",c);
}
```

Output:

```
Before function
20
After function
```



Calling Function	Action	Called Function

Function1( )
{
____
____
____
____
Function2 (x);
}
	arguments →
	← no return values
		Function2 (a)
		________
		{ ________
		________
		________
		________
		________
		}

User Defined Functions

- Type3: Return Type without Arguments
- Definition: This function takes no arguments and return some values based on its data types

Example Program

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int add(); //function declaration
        int r;
        clrscr();
        r=add(); //function calling
        printf("\n Addition%d:",r);
        getch();
}
        int add() //function definition
        {
                int a,b,c;
                a=1,b=2;
                c=a+b;
                return(c);
        }
```

Output:

```
Addition: 3
```

User Defined Functions

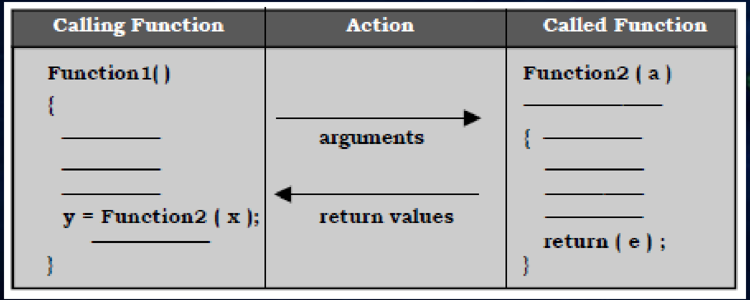
- **Type4: Return Type with Arguments**
- **Definition: This function takes one or more (some) number of arguments and return some values based on its data types**

Example Program

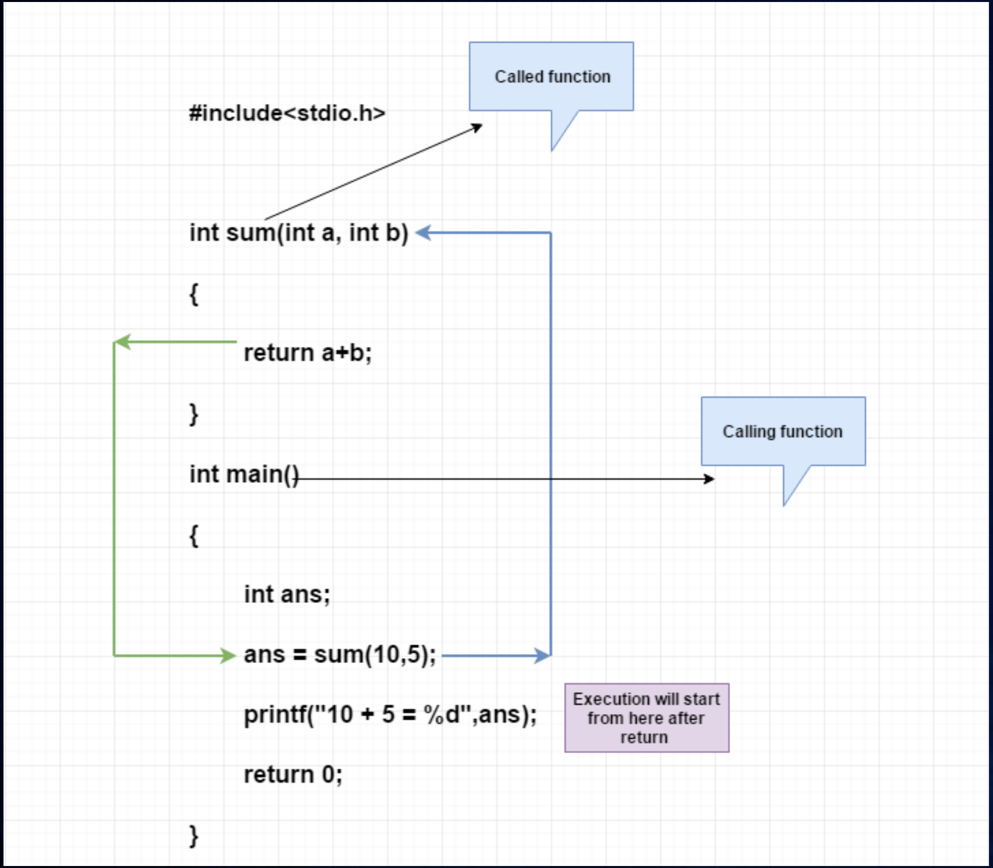
```
#include<stdio.h>
#include<conio.h>
int demo(int,int); //function declaration
void main()
{
        int x=10,u=5;
        clrscr();
        printf("\n Before function:");
        printf("\n After function:%d",demo(x,u)); //function calling
        getch();
}
int demo(int a,int b) //function definition
{
        int c=a-b;
        return c;
}
```

Output:

```
Before Function
After Function: 5
```



Example



```
#include<stdio.h>

int sum(int a, int b)
{
      return a+b;
}
int main()
{
      int ans;
      ans = sum(10,5);
      printf("10 + 5 = %d",ans);
      return 0;
}
```

Factorial Using Functions - Example

```
#include<stdio.h>
#include<conio.h>
int Fact(int);
void main()
{

        int n;
        clrscr();
        printf("\n Enter the Number for N");
        scanf("%d", &n);
        printf("\n The Factorial Value is:%d", Fact(n));
        getch();
}
int Fact(int K)
{
        int Fact=1, i;
        for(i=1; i<=K; i++)
        {
                Fact = Fact * i;
        }
        return Fact;
}
```

```
Output:
Enter the Number for N: 5
The Factorial Value is: 120
```

Call by Value Function

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int x=10,y=5;
        void demo(int,int);
        clrscr();
        printf("\n\n Before swapping  X:%d,Y:%d",x,y);
        demo(x,y);
        getch();
}
void demo(int a,int b)
{
        int temp;
        temp=a;
        a=b;
        b=temp;
        printf("\n\n After swapping X:%d,Y:%d",a,b);
}
```

Output:

```
Before swapping X:10,   Y:5
After swapping    X:5,   Y:10
```

Call by Reference Function

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int x=10,y=5;
        void demo(int *,int *);
        clrscr();
        printf("\n Address are Before swapping  X:%u,Y:%u",&x,&y);
        demo(&x, &y);
        getch();
}
void demo(int *a,int *b)
{
        int *temp;
        temp=a;
        a=b;
        b=temp;
        printf("\n Address are After swapping X:%u,Y:%u\n",a,b);
}
```

Output:

```
Before swapping X:65524
                Y:65522
After swapping   X:65522
                 Y:65524
```

Functions with Arrays

- Arrays can be used with functions

```
#include<stdio.h>
#include<conio.h>
int avg(int[]);
void main()
{
		int marks[5]={98, 99, 100, 96, 94};
		clrscr();
		printf("\n Average is: %d", avg(marks));
		getch();
}
int avg(int sum[])
{
		int i, cal=0;
		for(i=0;i<5;i++)
		{
				cal = cal + sum[i];
		}
		return cal/5;
}
```

Output:

```
Average  is : 97
```

Example program for argc and argv

- It is possible to pass some values from the command line to user's C programs when they are executed. These values are called command line arguments and many times they are important for user program especially when they want to control the program from outside.
- The command line arguments are handled using main() function arguments where
- **argc** refers to the number of arguments passed, and
- **argv[]** is a pointer array which points to each argument passed to the program

Example program for argc and argv

```
#include <stdio.h>
 int main (int argc, char *argv[])
{
      int i=0;
      printf("\ncmdline args count=%s", argc);

      printf("\nexe name=%s", argv[0]);
      for (i=1; i< argc; i++)
     {
             printf("\narg%d=%s", i, argv[i]);
     }
    printf("\n");
    return 0;
 }
```

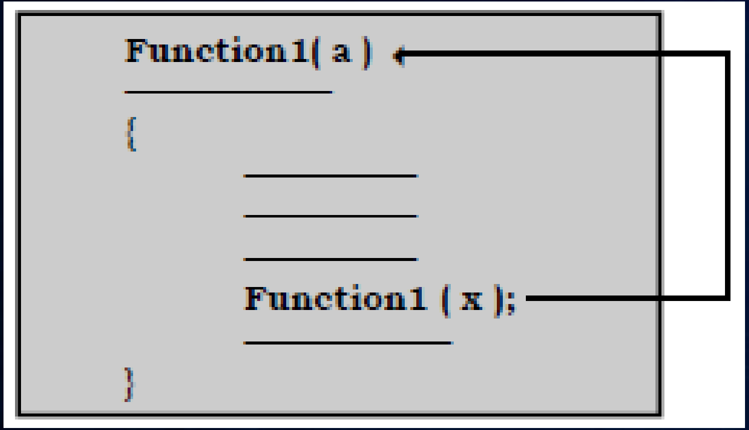
Output

```
$ ./cmdline_basic test1 test2 test3 test4 1234 56789
cmdline args count=7
 exe name=./cmdline_basic
 arg1=test1
 arg2=test2
 arg3=test3
 arg4=test4
 arg5=1234
 arg6=56789
```

Note: It holds value ‘7’ (in which one argument is executable name and ‘6’ are arguments passed to program

Recursive Function

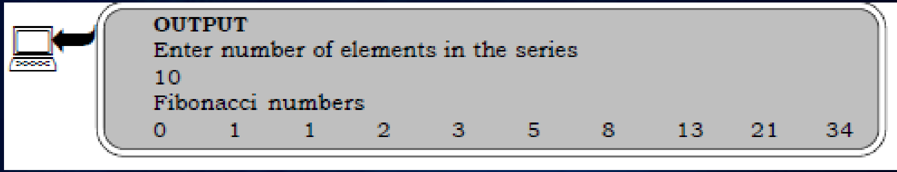
- In many situations it is possible for users to have functions that call itself directly or indirectly again and again. Such functions are termed as recursive functions and the process is termed as recursion.
- Many algorithm and mathematical definitions are naturally described recursively
- A function is called by itself which is known as '**Recursive function**'



```
Function1( a )
____________
{
        __________
        __________
        __________
        Function1 ( x );
        ____________
}
```

Recursive Function for Fibonacci Series – Example

```
#include <stdio.h>
int fibo( int );
void main( )
{
            int i,n;
            printf("Enter number of elements in the series \n");
            scanf("%d",&n);
            printf("\nFibonacci numbers\n\n");
            for(i=1;i<=n;i++)
            printf("%d\t",fibo(i)); /* Function call */
}
/* Function to find fibonacci numbers */
int fibo(int k)
{
            if(k ==1)
            return(0);
            else if(k==2)
            return(1);
            else
            return(fibo(k-1)+fibo(k-2));
}
```



```
OUTPUT
Enter number of elements in the series
10
Fibonacci numbers
0    1    1    2    3    5    8    13    21    34
```

Tower of Hanoi – Recursion Application Example

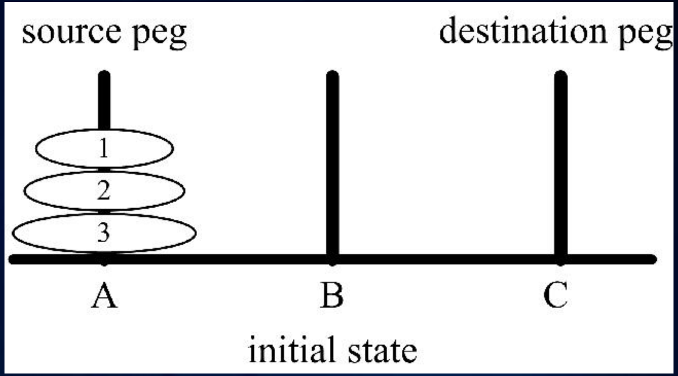
- The Tower of Hanoi (also called the Tower of Brahma or Lucas' Tower and sometimes pluralized) is a mathematical game or puzzle.

Rules

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.
- No disk may be placed on top of a smaller disk.

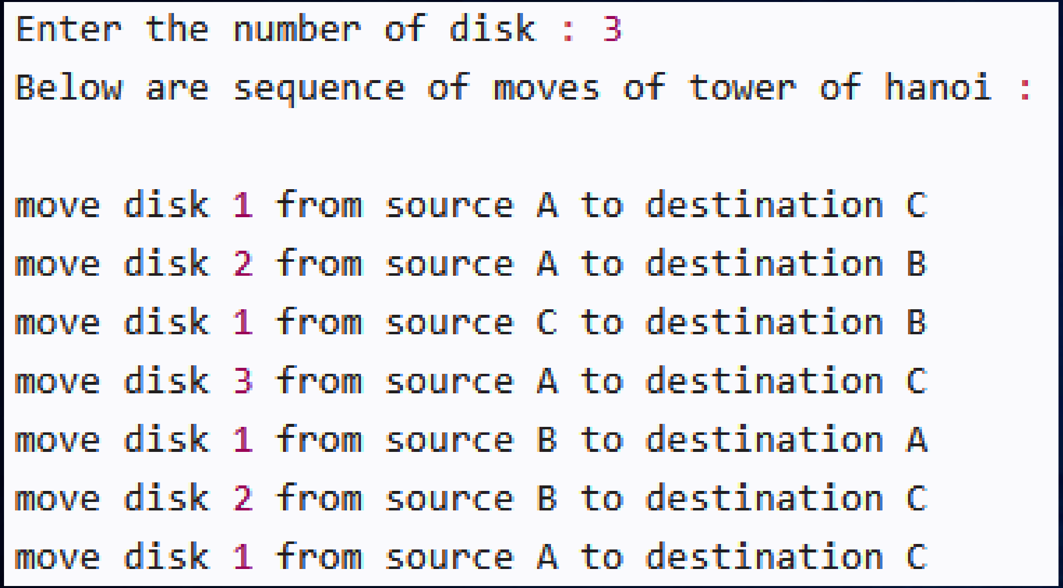
Tower of Hanoi – Recursion Application Example

```c
#include <stdio.h>
#include <conio.h>
void toh(int, char, char, char);
void main()
{
   int n;
   printf("Enter the number of disk : ");
   scanf("%d", &n);
   printf("Here is sequence of moves of tower of hanoi  :\n");
   toh(n, 'A', 'C', 'B');
}
void toh(int no, char source, char destination, char spare)
{
   if (no == 1)
      {
         printf("\n move disk 1 from source %c to destination %c", source, destination);
         return;
      }

     toh(no - 1, source, spare, destination);
     printf("\n move disk %d from source %c to destination %c", no, source, destination);
     toh(no - 1, spare, destination, source);
}
```

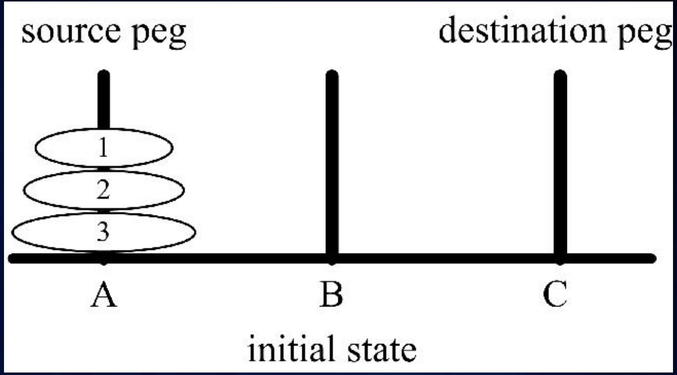


Tower of Hanoi – Recursion Application Example

Output



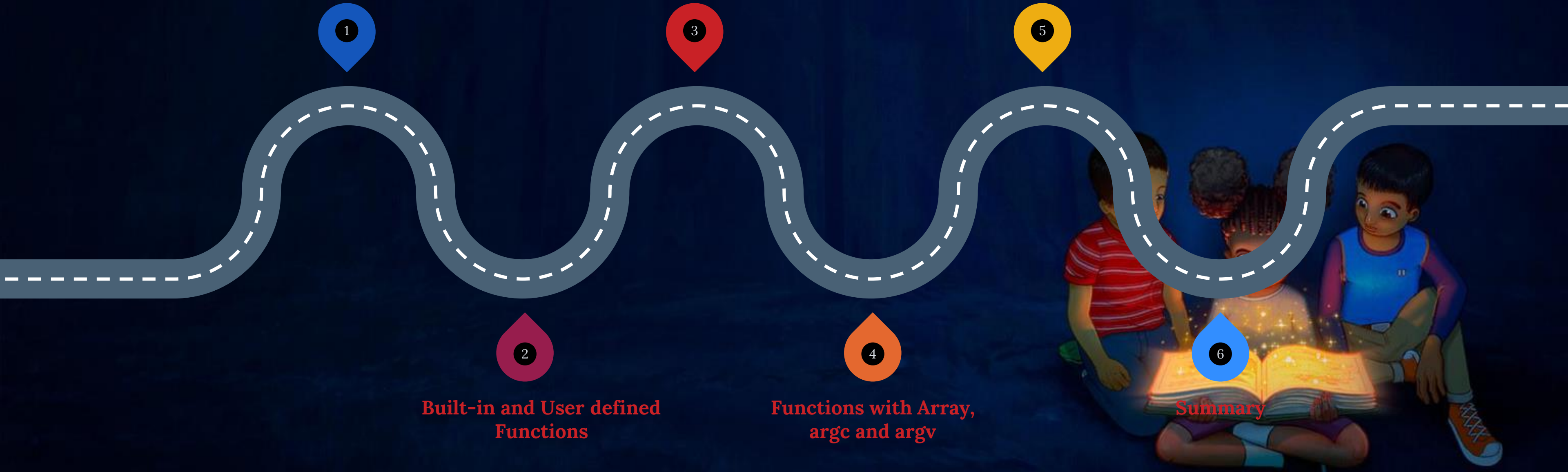
```
Enter the number of disk : 3
Below are sequence of moves of tower of hanoi :

move disk 1 from source A to destination C
move disk 2 from source A to destination B
move disk 1 from source C to destination B
move disk 3 from source A to destination C
move disk 1 from source B to destination A
move disk 2 from source B to destination C
move disk 1 from source A to destination C
```



Credits

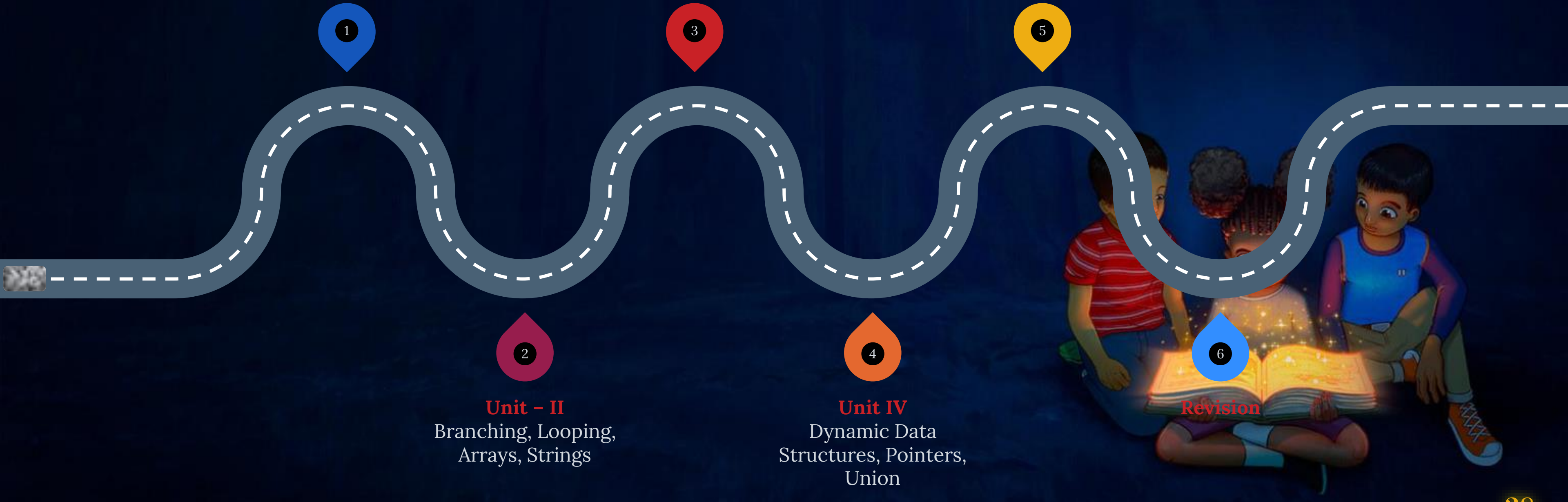
- https://www.thegeekstuff.com/2013/01/c-argc-argv/
- https://www.stechies.com/tower-of-hanoi-c/
- https://dotnettutorials.net/lesson/tower-of-hanoi-using-recursion-in-c/
- https://meeraacademy.com/c-program-for-tower-of-hanoi-using-recursion/
- https://www.tutorialspoint.com/cprogramming/c_command_line_arguments.htm

Roadmap for Unit III



1. Introduction to Functions
2. Built-in and User defined Functions
3. Call by Value and Call by Reference
4. Functions with Array, argc and argv
5. Recursion and Tower of Hanoi
6. Summary

Roadmap to Programming in C



Unit I
Computer Based Problem Solvingand Overview of C

Unit – II
Branching, Looping, Arrays, Strings

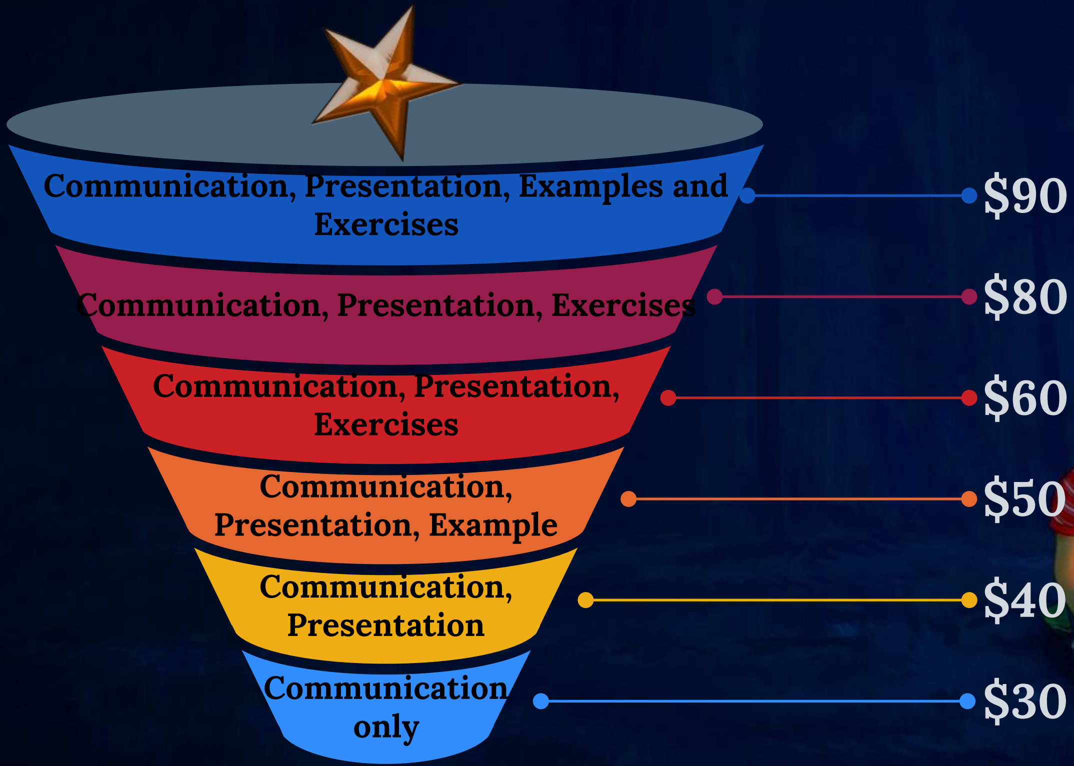
Unit III
Functions and Recursions

Unit IV
Dynamic Data Structures, Pointers, Union

Unit V
File Handling, Preprocessor, Standard Library and Header Files

Revision

Feedback – Rating Star



Communication, Presentation, Examples and Exercises	$90
Communication, Presentation, Exercises	$80
Communication, Presentation, Exercises	$60
Communication, Presentation, Example	$50
Communication, Presentation	$40
Communication only	$30

Thanks!

Any questions?

You can find me at:

- kramdharma@gmail.com