Session Outline

- What is Function
- Advantages of using function
- Types of function
- Function Prototype
- Defining function
- Calling Function
- Return Statement
- Local and global Variables
- Categories Of Function
- Recursion

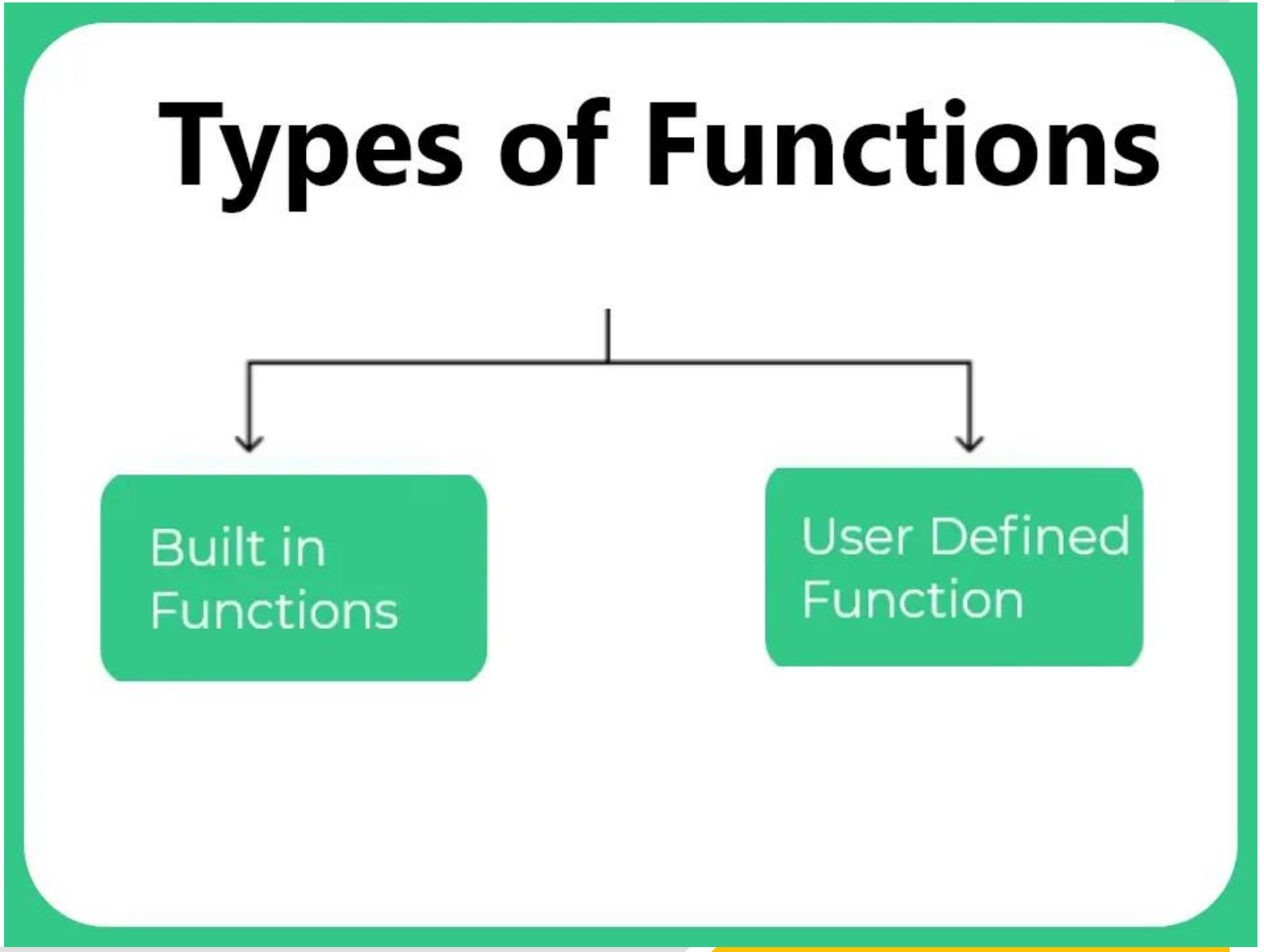
What is function

- A **function** is a group of statements that together perform a task.
- Every c program has at least one function called **main()**

Advantages of Using Function

- The use of functions makes a program more **readable**. It's frequently difficult to read a large program. Breaking the code down into smaller functions keeps the program structured, understandable, and reusable.
- The function can be **reused countless** times after it is defined.
- Using a function, it is possible to **reduce the size of a program** by calling and using the function at different places in the program.
- Functions help in code **modularity**, which means that the entire code is divided into separate blocks, each of which is self-contained and performs a different task. This makes each block implementation and debugging much easier.

Types of function



Types of function

1. Build in Function /Library Functions:

- These functions are grouped together and placed in common folder called library.
- E.g: printf(),scanf()

2.User Defined Functions:

- These functions are created by user according to programming need are called as user defined functions.
- E.g: Function to perform addition of two int numbers

Library (Built In) Functions:

Header Files	Functions Defined
stdio.h	Printf(), scanf(), getchar(), putchar(), gets(), puts(), fopen(), fclose()
conio.h	Clrscr(), getch()
Ctype.h	Toupper(), tolower(), isalpha()
Math.h	Pow(), sqrt(), cos(), log()
Stdlib.h	Rand(), exit()
String.h	Strlen(), strcpy(), strupr()

Elements of User Defined Functions

1. Function Prototype
2. Function Call
3. Function Definition

Function Prototype

- It is defined in the beginning before the function call is made.
- **Syntax:**

return_type name_of_function(list_of_arguments);

E.g: int sum(int a ,int b);

Function Call

- Function call can be made by specifying name and list of arguments enclosed in parenthesis and separated by comma.
- If there are no arguments empty parenthesis are placed after function name.
- **Syntax:**

name_of_function(list_of_arguments);

- If function return a value then function call can be written as :

c=sum(x,y);

Function arguments and parameters

- **Arguments** are also called as **actual parameters**.
- **Arguments** are written within parenthesis at the time of **function call**.
- **Parameters** are also called as **Formal parameters**.
- **Parameters** are written within parenthesis at the time of **function definition**.

Function Definition

- It is dependant on program module.
- We can write actual logic into function definition
- First line of the function is called function declaration and rest line inside {} are called function body.

Return statement

- It is the last statement of the function that return certain value where the function is called.
- **Syntax:**

return(variable name or constant);

- E.g: return(c);

```
1  int a = 10, b = 5, c;
2
3  int product(int x, int y);
4
5  int main(void)
6  {
7      c = product(a,b);
8
9      printf("%i\n",c);
10
11     return 0;
12 }
13
14 int product(int x, int y)
15 {
16     return (x * y);
17 }
```

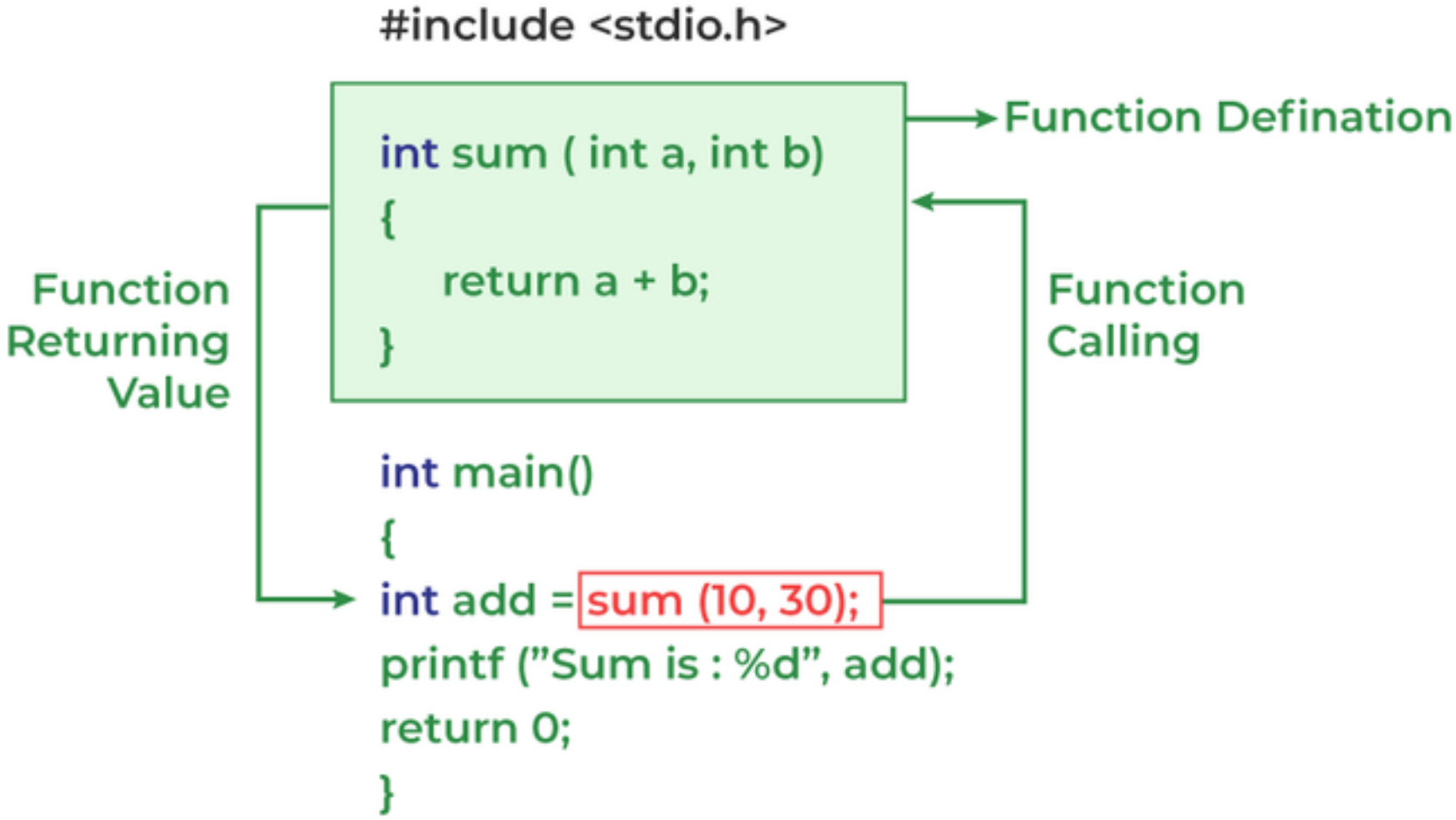
Function Prototype - int is the return type and int x and int y are the function arguments

Main Function - int is always the return type and there are no arguments, hence the (void). Curly braces { } mark the start and end of the main function

Function call - product(a,b); a and b are global variables the function is passed. Here the values returned by the function are assigned to the variable c

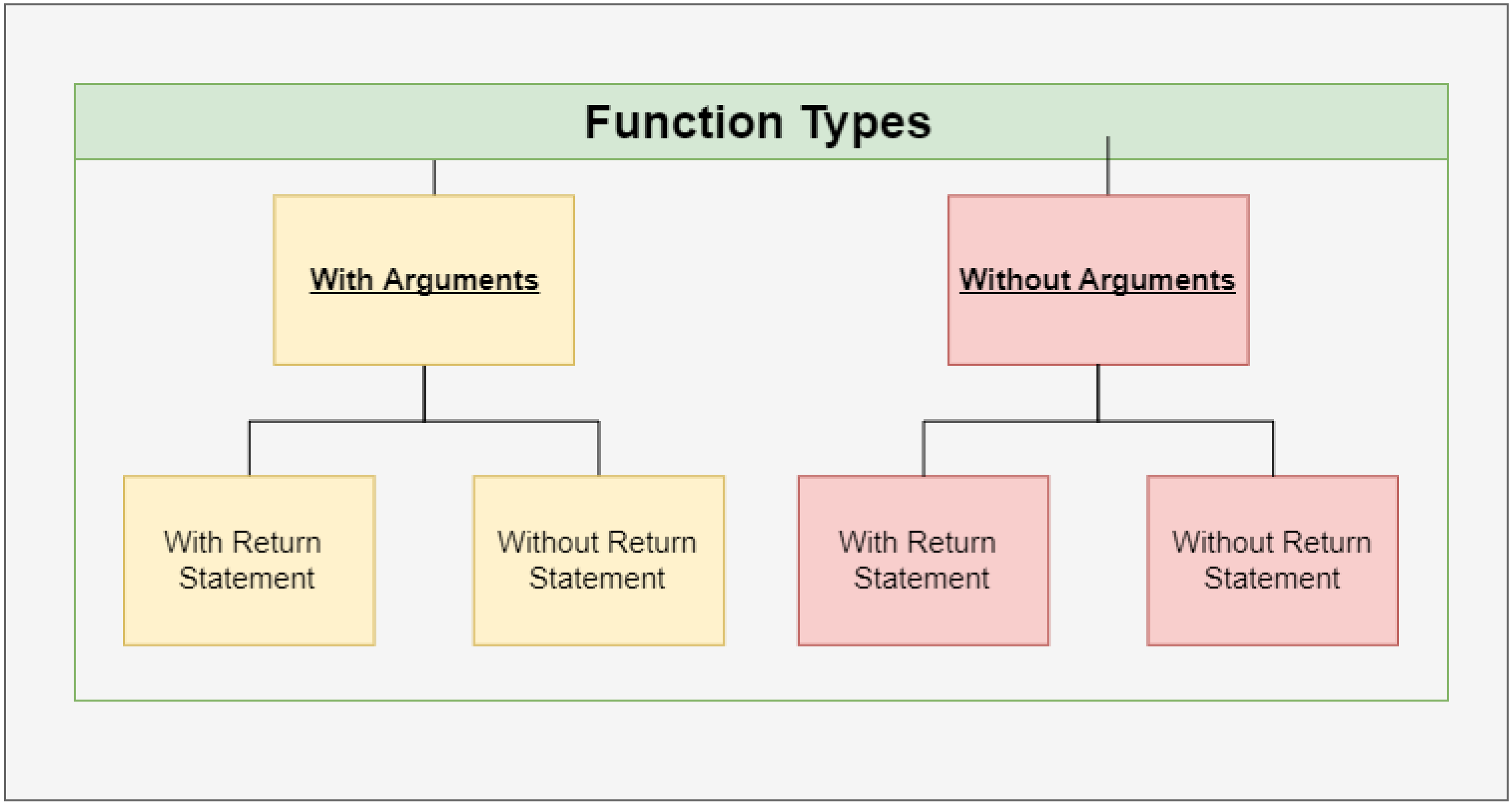
Function Definition - contains the function statement return(x * y); the function returns x times y to the main function where it was called. Curly braces { } mark the start and end of the function

Working of Function in C



Local and Global Variable

- ***Local Variable:*** The variable whose scope lies inside a function or a block in which they are declared.
- ***Global Variable:*** The variable that exists outside of all functions. It is the variable that is visible from all other scopes.


Function Types
With Arguments
Without Arguments
With Return Statement
Without Return Statement
With Return Statement
Without Return Statement

Function with no argument and no return

```
#include<stdio.h>
#include<conio.h>
void sum();
void main()
{
 sum();
 getch();
}
void sum()
{
 int a,b;
 printf("Enter the two integers");
 scanf("%d%d",&a,&b);
 printf("The sum of the numbers you entered is %d",a+b);
}
```

Function with argument and no return

```
#include<stdio.h>
#include<conio.h>
void sum(int,int);
void main()
{
 int a,b;
 printf("Enter the two integers");
 scanf("%d%d",&a,&b);
 sum(a,b);
 getch();
}
void sum(int a, int b)
{
 printf("The sum of the numbers you entered is %d",a+b);
}
```

Function with no argument and return

```
#include<stdio.h>
#include<conio.h>
int sum();
void main()
{
 int x;
 x=sum();
 printf("The sum of the numbers you entered is %d",x);
 getch();
}
int sum()
{
 int a,b;
 printf("Enter the two integers");
 scanf("%d%d",&a,&b);
 return(a+b);
}
```

Function with argument and return

```
#include<stdio.h>
#include<conio.h>
int sum(int,int);
void main()
{
 int a,b,x;
 printf("Enter the two integers");
 scanf("%d%d",&a,&b);
 x=sum(a,b);
 printf("The sum of the numbers you entered is %d",x);
 getch();
}
int sum(int a, int b)
{
 return(a+b);
}
```

Call By Value

- In **call by value** method of parameter passing, the values of actual parameters are copied to the function's formal parameters.
- There are two copies of parameters stored in different memory locations.
- One is the original copy and the other is the function copy.
- Any changes made inside functions are not reflected in the actual parameters of the caller.

Call By Reference

- In **call by reference** method of parameter passing, the address of the actual parameters is passed to the function as the formal parameters.
- Both the actual and formal parameters refer to the same locations.
- Any changes made inside the function are actually reflected in the actual parameters of the caller.

Recursion in C

Recursion is the technique of making a function call itself.

```
#include<stdio.h>
int sum(int k);
int main() {
  int result = sum(10);
  printf("%d", result);
  return 0;
}
int sum(int k) {
  if (k > 0) {
    return k + sum(k - 1);
  } else {
    return 0;
  }}
```


Thank You
Any Questions?