First year CPS Module 4 (Function and Recursion)

1. **What is function in C? What are the different types of function? Explain the elements of user defined function.**
2. **With an example, explain the categories of User defined function.**
3. **With an example, explain the following:**
   i) Call by value ii) Call by reference

4. **Compare the following:**
   a. Actual parameter and formal parameter
   b. Local variable and global variable
   c. Built in function and user defined function
   d. Call by value and call by reference
5. **What is recursion? What are the properties of recursion? Develop a C program to generate first n Fibonacci numbers using recursion**
6. **Using Recursion Develop C program :**
   a. To find GCD of two number
   b. To convert binary to decimal
   c. To find factorial of n
7. **Briefly explain different types of Storage class used in C**

What is function in C? What are the different types of function? Explain the elements of user defined function.

What is function?

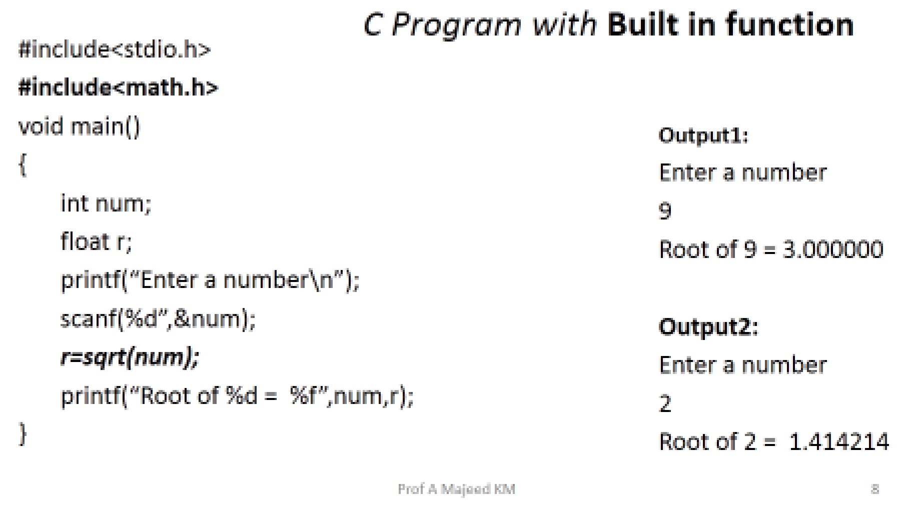
- *A function is a group of statements that together perform a specific task(Sub Program)*

Advantages of using function*:*

  - Reduces the size of the program
  - Easy to detect and correct error
  - Reduces programming complexity
  - Improves reusability

Different types of functions

- **Built in functions(Pre-defined function)**
  - Functions that are already written, compiled and their definitions are grouped and stored in header files. Eg. sqrt(), abs()



C Program with **Built in function**

```
#include<stdio.h>
#include<math.h>
void main()
{
     int num;
     float r;
     printf("Enter a number\n");
     scanf(%d",&num);
     r=sqrt(num);
     printf("Root of %d =  %f",num,r);
}
```

Output1:

```
Enter a number
9
Root of 9 = 3.000000
```

Output2:

```
Enter a number
2
Root of 2 =  1.414214
```

Prof A Majeed KM 8

- **User defined function**
  - Functions that are written by the user to carry on some task.

  C Program to find square of number

```
#include <stdio.h>
int square(int a);   //function prototype or function declaration
main()
{
  int x, sqr;
  printf("Enter a number: \n");
  scanf("%d", &x);
  sqr = square (x);    //function call
  printf("Square = %d\n", sqr);
}                  //end main
int square (int a)    //function definition
{
  int s;
  s = a*a;
  return s;  //returns the square value s
}          //end function square
```

Elements of user defined functions: There are three elements. They are:

Function Declaration, Function Definition, and function call

- **Function Declaration/Function prototype:** If you are defining function **after main()** function then it is **mandatory to declare function** at global declaration section. If you define function before main function then function declaration is optional.

Syntax: It consist of return type followed by function name followed by parameter list and terminated with semicolon

return_type function_name(parameter list);

Ex:

int square (int);

Here, int before function name indicates that this function returns integer value to the calling function while int inside parentheses indicates that this function will receive an integer value from calling function.

- **Function Definition:** A function definition provides the actual body of the function.

Syntax:

return_type function_name (Parameter list) //Function header

{ //Function body

Local variable declaration;

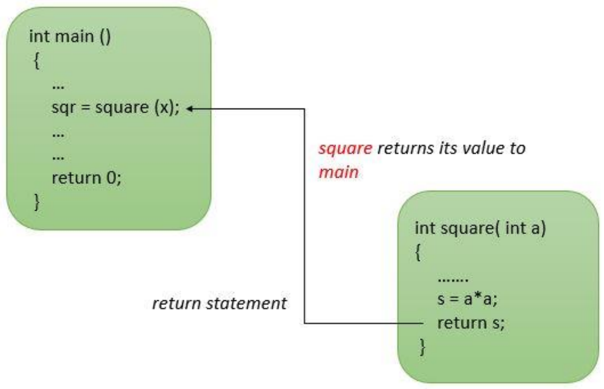
Statement(s);

Return statement;

}

It consists of a function header and a function body. The function_name is an identifier. The return_type is the data type of value which will be returned to a calling function. Some functions performs the desired task without returning a value which is indicated by **void as a return _type.**

Function body consist of Local variable declaration, Statement(s), and Return statement All definitions and statements are written inside the body of the function. Return statement returns the value and transfer control to the calling function.

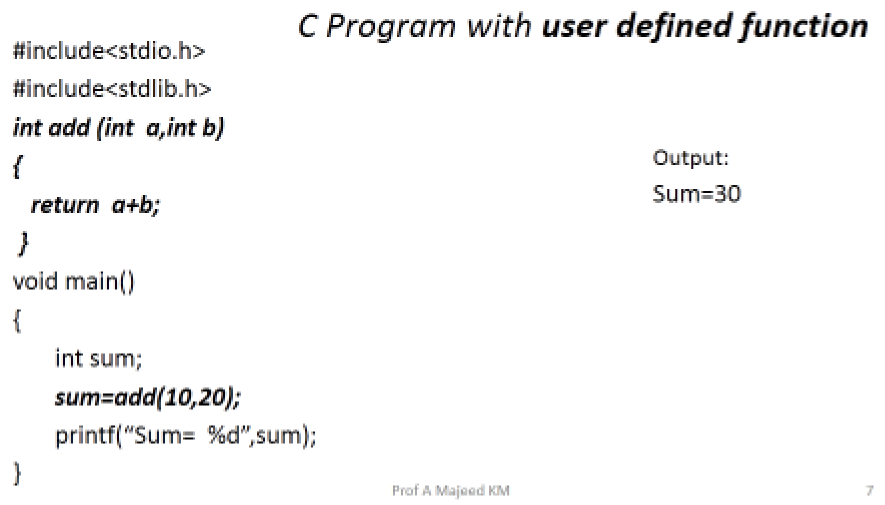


Example: Function to calculate square of a number

```
 int square(int a)   // header
 {
   int s; //local variable declaration
   s=a*a;  // statement
   return s;  //return statement
 }
```

- **Function Call:** Calling a function which is already defined
  Syntax: function_name followed by parameter list, and terminated by semicolon
  Function_name(parameter list);
  Ex:
  square (x);

Write a C program to find sum of two numbers using function



C Program with ***user defined function***

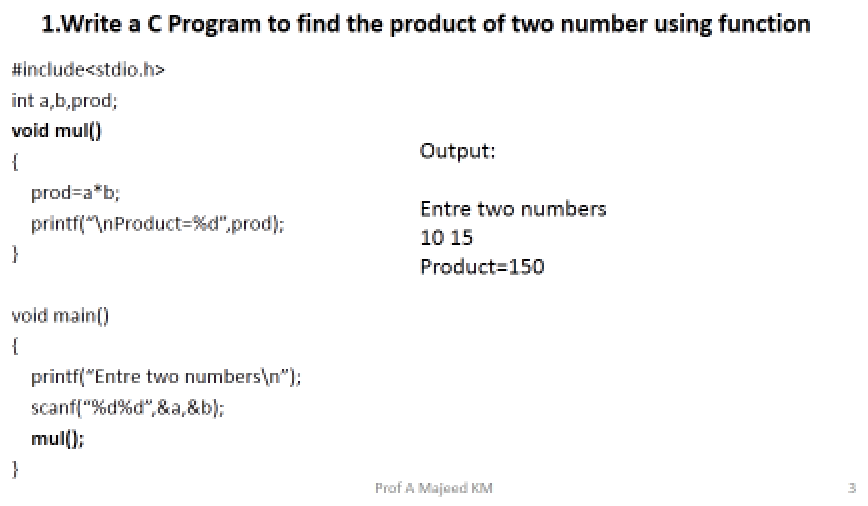
```
#include<stdio.h>
#include<stdlib.h>
int add (int  a,int b)
{
  return  a+b;
 }
void main()
{
     int sum;
     sum=add(10,20);
     printf("Sum=  %d",sum);
}
```

```
Output:
Sum=30
```

Prof A Majeed KM 7

Exercises:

1. **Write a C Program to find the product of two number using function**
2. **Write a C Program to find the Areas of circle and rectangle using two different functions**
3. **Write a C Program to find the values of sin(x) and cos(x) using built in function**
4. **Write a C Program to find largest of two number using function**



1.Write a C Program to find the product of two number using function

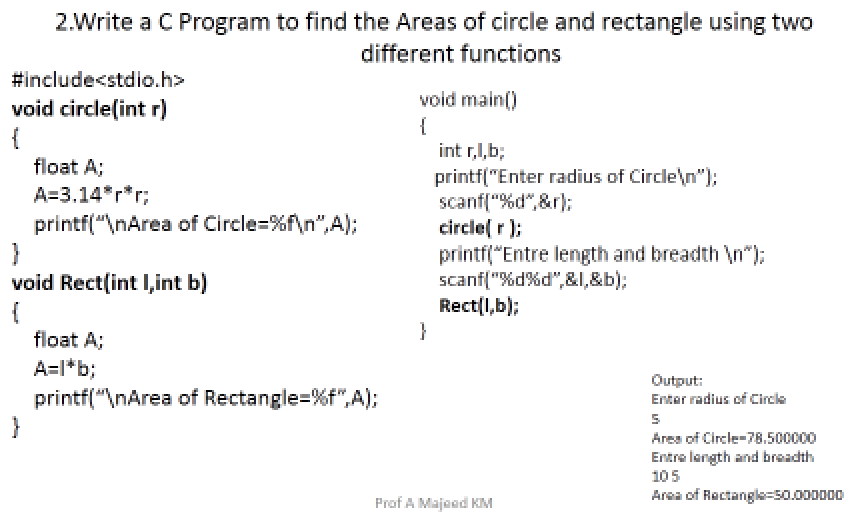
```
#include<stdio.h>
int a,b,prod;
void mul()
{
   prod=a*b;
   printf("\nProduct=%d",prod);
}

void main()
{
   printf("Entre two numbers\n");
   scanf("%d%d",&a,&b);
   mul();
}
```

Output:

```
Entre two numbers
10 15
Product=150
```

Prof A Majeed KM 3



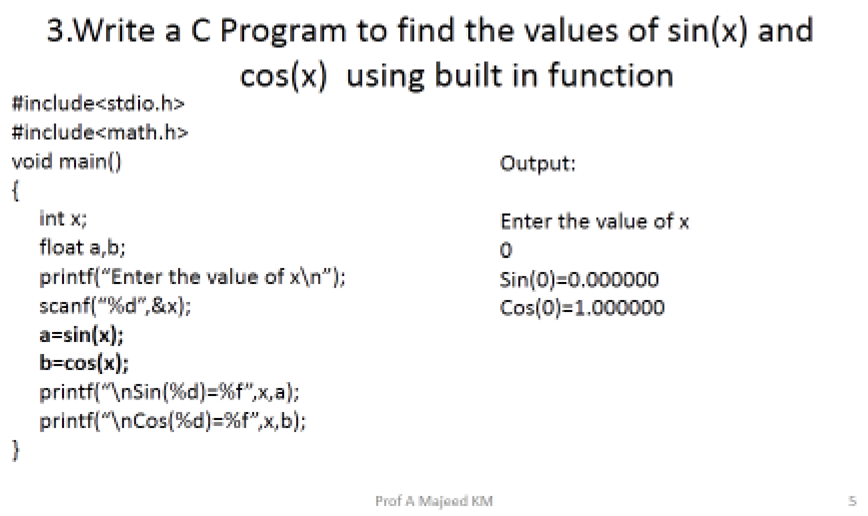
2.Write a C Program to find the Areas of circle and rectangle using two different functions

```
#include<stdio.h>
void circle(int r)
{
   float A;
   A=3.14*r*r;
   printf("\nArea of Circle=%f\n",A);
}
void Rect(int l,int b)
{
   float A;
   A=l*b;
   printf("\nArea of Rectangle=%f",A);
}
```

```
void main()
{
   int r,l,b;
  printf("Enter radius of Circle\n");
   scanf("%d",&r);
   circle( r );
   printf("Entre length and breadth \n");
   scanf("%d%d",&l,&b);
   Rect(l,b);
}
```

```
Output:
Enter radius of Circle
5
Area of Circle=78.500000
Entre length and breadth
10 5
Area of Rectangle=50.000000
```

Prof A Majeed KM



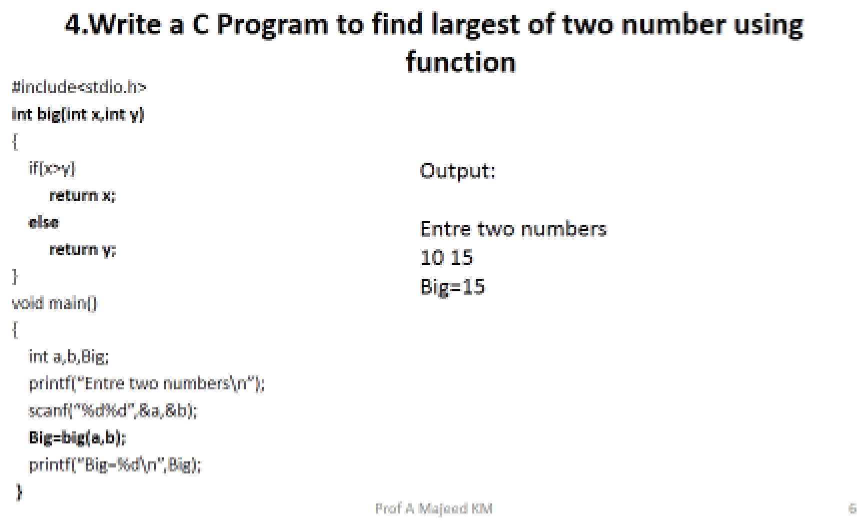
3.Write a C Program to find the values of sin(x) and cos(x) using built in function

```
#include<stdio.h>
#include<math.h>
void main()
{
    int x;
    float a,b;
    printf("Enter the value of x\n");
    scanf("%d",&x);
    a=sin(x);
    b=cos(x);
    printf("\nSin(%d)=%f",x,a);
    printf("\nCos(%d)=%f",x,b);
}
```

Output:

```
Enter the value of x
0
Sin(0)=0.000000
Cos(0)=1.000000
```

Prof A Majeed KM 5



4.Write a C Program to find largest of two number using function

```
#include<stdio.h>
int big(int x,int y)
{
   if(x>y)
      return x;
   else
      return y;
}
void main()
{
   int a,b,Big;
   printf("Entre two numbers\n");
   scanf("%d%d",&a,&b);
   Big=big(a,b);
   printf("Big=%d\n",Big);
}
```

Output:

```
Entre two numbers
10 15
Big=15
```

Prof A Majeed KM 6

Compare Actual Parameter and formal Parameter (or Actual argument Vs Formal argument)

The argument/parameter which is **used in function call is known as actual argument**/ actual parameter.

The argument/Parameter which is **used inside in function definition is known as formal argument**/formal parameter.

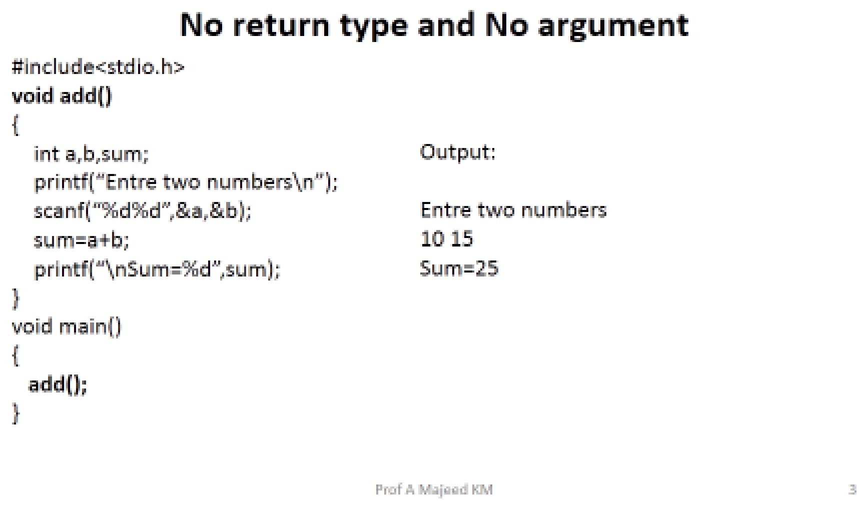
Example: In this example given below, x is actual argument and a is formal argument

```
#include <stdio.h>
int square(int a);   //function prototype or function declaration
main()
{
  int x, sqr;
  printf("Enter a number: \n");
  scanf("%d", &x);
  sqr = square (x);    //function call : x is actual argument
  printf("Square = %d\n", sqr);
}                    //end main
int square (int a)    //function definition: a is formal argument
{
  int s;
  s = a*a;
  return s;  //returns the square value s
}           //end function square
```

Types of user defined functions: (With an example, explain the categories of User defined function.)

There are mainly four types of user defined functions in C. They are

- **C function: Based on argument and return values**
  - **Function with No return type and No argument**
  - **Function with No return type ,but argument**
  - **Function with return type ,but No argument**
  - **Function with both return type and argument**
- **Function with No return type and No argument**
  Function with **no argument means** the called function does not receive any data from calling function and **Function with no return value** means calling function does not receive any data from the called function. So **there is no data transfer between calling and called function.**



No return type and No argument

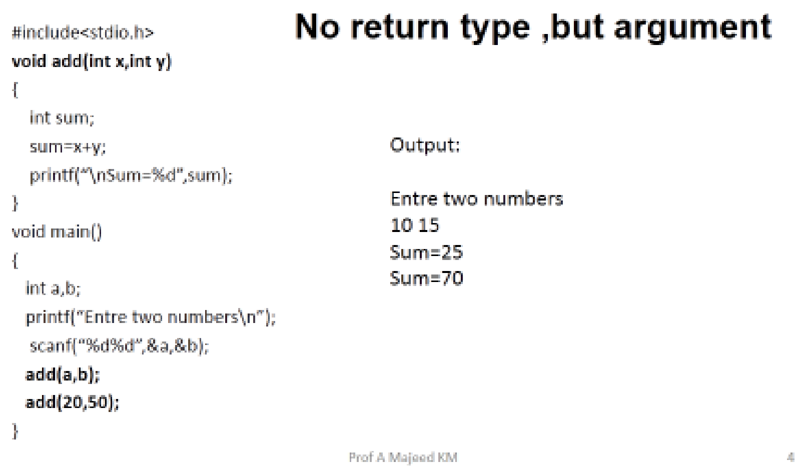
```
#include<stdio.h>
void add()
{
   int a,b,sum;
   printf("Entre two numbers\n");
   scanf("%d%d",&a,&b);
   sum=a+b;
   printf("\nSum=%d",sum);
}
void main()
{
  add();
}
```

```
Output:

Entre two numbers
10 15
Sum=25
```

Prof A Majeed KM 3

- **Function with No return type ,but argument:**
  Here function will **accept data from the calling function** as there are arguments, however, since **there is no return type nothing will be returned to the calling program**. So it's a one-way type communication.



No return type ,but argument

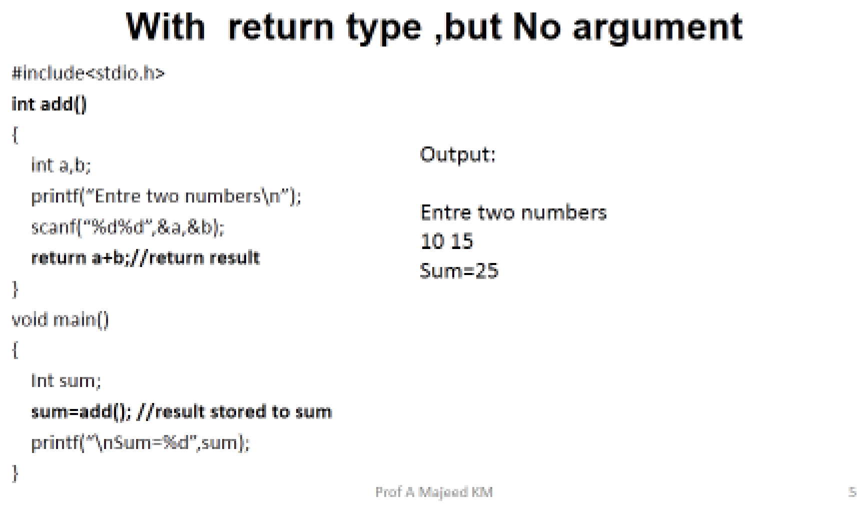
```
#include<stdio.h>
void add(int x,int y)
{
   int sum;
   sum=x+y;
   printf("\nSum=%d",sum);
}
void main()
{
  int a,b;
  printf("Entre two numbers\n");
   scanf("%d%d",&a,&b);
  add(a,b);
  add(20,50);
}
```

```
Output:

Entre two numbers
10 15
Sum=25
Sum=70
```

Prof A Majeed KM 4

- **Function with return type ,but No argument**
  Function with **no arguments means called function does not receive any data** from calling function and function with return **value means one result will be sent back to the calling function.**



With return type ,but No argument

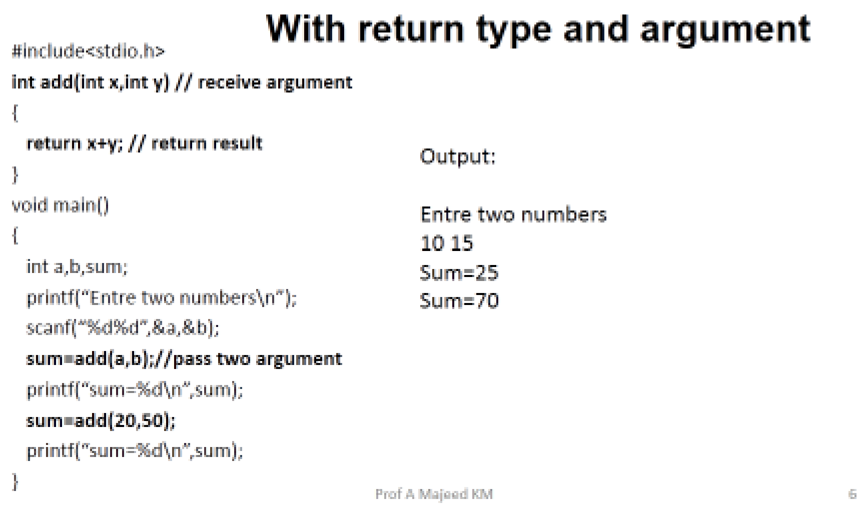
```
#include<stdio.h>
int add()
{
   int a,b;
   printf("Entre two numbers\n");
   scanf("%d%d",&a,&b);
   return a+b;//return result
}
void main()
{
   Int sum;
   sum=add(); //result stored to sum
   printf("\nSum=%d",sum);
}
```

Output:

```
Entre two numbers
10 15
Sum=25
```

Prof A Majeed KM 5

- **Function with both return type and argument**
  Function with arguments and return type means **both the calling function and called function will receive data from each other**. It's like a dual communication.



With return type and argument

```
#include<stdio.h>
int add(int x,int y) // receive argument
{
  return x+y; // return result
}
void main()
{
  int a,b,sum;
  printf("Entre two numbers\n");
  scanf("%d%d",&a,&b);
  sum=add(a,b);//pass two argument
  printf("sum=%d\n",sum);
  sum=add(20,50);
  printf("sum=%d\n",sum);
}
```

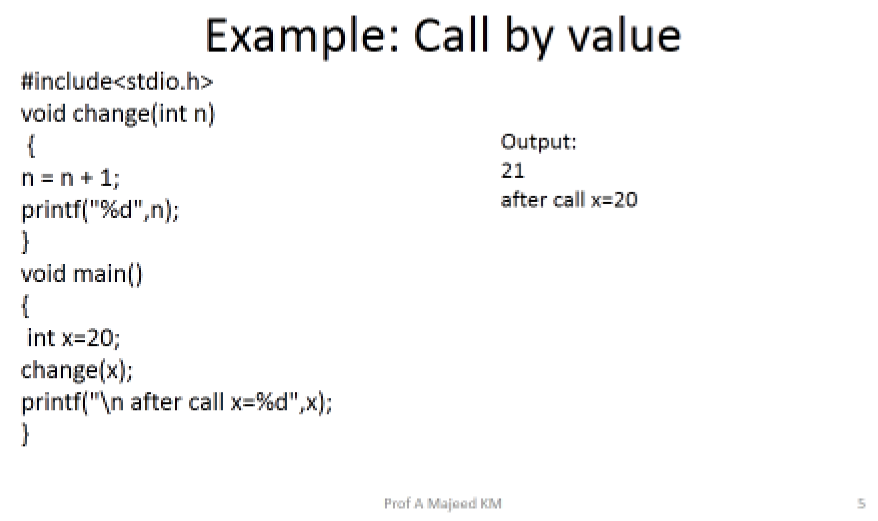
Output:

```
Entre two numbers
10 15
Sum=25
Sum=70
```

Prof A Majeed KM 6

*With an example, explain i) Call by value ii) Call by reference (*Explain parameter passing techniques in C)

- **Call by value or pass by value**: It is one of the parameter passing techniques in C
    - In this method, the value contained in the actual argument is passed to the formal argument.
    - In other words, a copy of the actual argument is passed to the function.
    - Hence, if the formal argument is modified within the function, **the change is not reflected in the actual argument at the calling place**



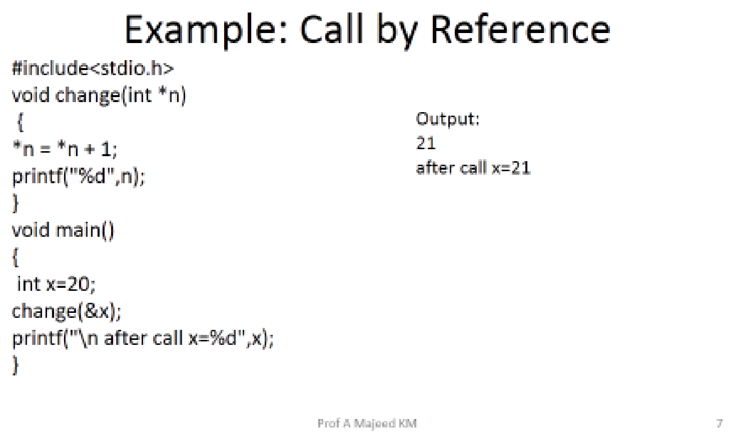
Example: Call by value

```
#include<stdio.h>
void change(int n)
 {
n = n + 1;
printf("%d",n);
}
void main()
{
 int x=20;
change(x);
printf("\n after call x=%d",x);
}
```

```
Output:
21
after call x=20
```

Prof A Majeed KM 5

- **Call by reference or pass by reference:** It is an another parameter passing techniques in C
    - Here the reference of the actual argument is passed to the function.
    - As a result, the memory location allocated to the actual argument will be shared by the formal argument.
    - So, **if the formal argument is modified within the function, the change will be reflected in the actual argument at the calling place.**
    - An ampersand symbol (&) is placed in between the data type and the variable in the function header for the reference variable.

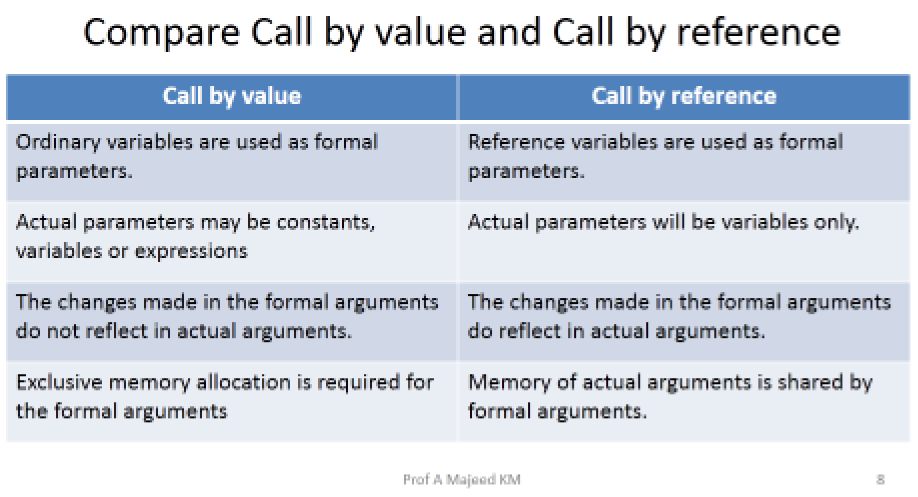


Example: Call by Reference

```
#include<stdio.h>
void change(int *n)
 {
*n = *n + 1;
printf("%d",n);
}
void main()
{
 int x=20;
change(&x);
printf("\n after call x=%d",x);
}
```

```
Output:
21
after call x=21
```

Prof A Majeed KM 7



Compare Call by value and Call by reference

Call by value	Call by reference
Ordinary variables are used as formal parameters.	Reference variables are used as formal parameters.
Actual parameters may be constants, variables or expressions	Actual parameters will be variables only.
The changes made in the formal arguments do not reflect in actual arguments.	The changes made in the formal arguments do reflect in actual arguments.
Exclusive memory allocation is required for the formal arguments	Memory of actual arguments is shared by formal arguments.

Prof A Majeed KM 8

Compare Local variable and global variable

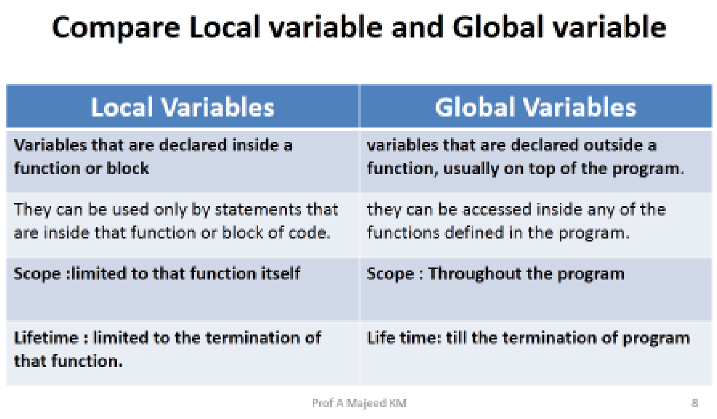
- A *scope* in any programming is a *region of the program where a defined variable can have its existence* and beyond that variable *cannot be accessed*
  - ***Local Variable***
  - ***Global Variable***

Local variable:

- Variables that are declared inside a function are called local variables.
- They can be used only by statements that are inside that function
- Scope of local variable is limited to that function
- Lifetime of local variable limited to the termination of that function.

Global variable:

- Global variables are declared outside a function, usually on top of the program.
- Global variables hold their values throughout the lifetime of your program and they can be accessed inside any of the functions defined for the program.
- Scope : Throughout the program
- Life time: till the termination of program



Compare Local variable and Global variable

Local Variables	Global Variables
Variables that are declared inside a function or block	variables that are declared outside a function, usually on top of the program.
They can be used only by statements that are inside that function or block of code.	they can be accessed inside any of the functions defined in the program.
Scope :limited to that function itself	Scope : Throughout the program
Lifetime : limited to the termination of that function.	Life time: till the termination of program

Prof A Majeed KM 8

Example for local/global variables

```
#include<stdio.h>
#include<stdlib.h>
int  a=10,b=5;  /*Global Variable*/
int add ()
{
     int s; /* Local variable*/
    s=a+b;
    return  s;
 }
void main()
{
      int sum;  /*Local Variable*/
      sum=add();
      printf("Sum=  %d",sum);
}
```

Briefly explain different types of Storage class used in C

What is a Storage Class?

A storage class represents the visibility and a location of a variable. It tells from what part of code we can access a variable. A storage class is used to describe the following things:

- The variable scope.
- The location where the variable will be stored.
- The initialized value of a variable.
- A lifetime of a variable.
- Who can access a variable?

Thus a storage class is used to represent the information about a variable.

There are total **four types of standard storage classes**. The table below represents the storage classes in 'C'.

Storage class	Purpose
auto	It is a default storage class.
extern	It is a global variable.
static	It is a local variable which is capable of returning a value even when control is transferred to the function call.
register	It is a variable which is stored inside a Register.

Auto storage class:

The variables defined using auto storage class are called as local variables. **Auto stands for automatic storage class**. A variable is in auto storage class by default if it is not explicitly specified. The scope of an auto variable is limited with the particular block only. Once the control goes out of the block, the access is destroyed. This means only the block in which the auto variable is declared can access it.

A **keyword auto is used to define an auto storage class**. By default, an auto variable contains a garbage value. Example, **auto int age**;

Extern storage class:

Extern stands for external storage class. Extern storage class is used when we have global functions or variables which are shared between two or more files.

Keyword **extern** is used to declaring a global variable or function in another file to provide the reference of variable or function which have been already defined in the original file.

The variables defined using an extern keyword are called as global variables. These variables are accessible throughout the program. Notice that the extern variable cannot be initialized it has already been defined in the original file

Ex:

```
#include <stdio.h>
extern i;
main()
{
  printf("value of the external integer is = %d\n", i);
}
```

Static storage class:

The static variables are used within function/ file as local static variables. They can also be used as a global variable

- Static local variable is a local variable that retains and stores its value between function calls or block and remains visible only to the function or block in which it is defined.
- Static global variables are global variables visible **only to the file in which it is declared.**
- Example: **`static int count = 10;`**

Register storage class:

You can use the register storage class when you want to store local variables within functions or blocks in CPU registers instead of RAM to **have quick access to these variables**. For example, "counters" are a good candidate to be stored in the register.

Example: **`register int age;`**

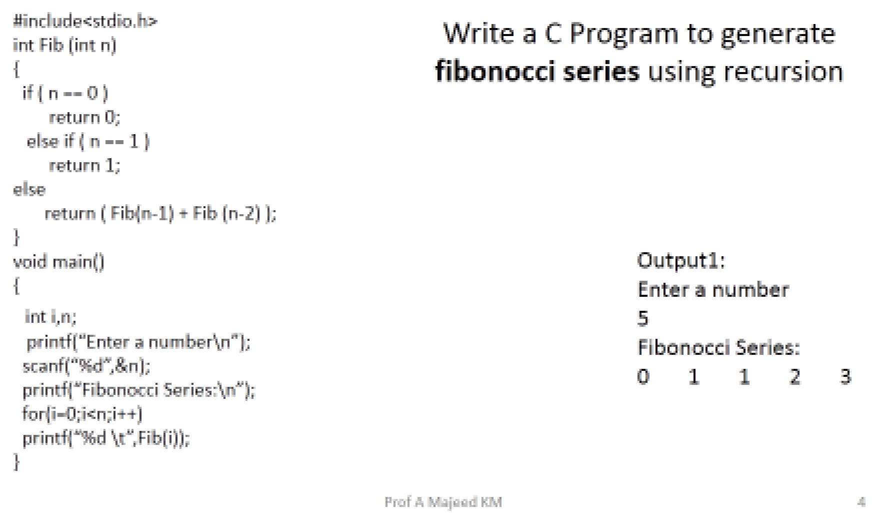
The keyword **register** is used to declare a register storage class. The variables declared using register storage class has lifespan throughout the program.

It is similar to the auto storage class. The variable is limited to the particular block. The only difference is that the variables declared using **register storage class are stored inside CPU registers instead of a memory. Register has faster access than that of the main memory.**

What is recursion? What are the properties of recursion? Develop a C program to generate first n Fibonacci numbers using recursion

- A function that calls itself is known as a recursive function. And, this technique is known as Recursion
- Advantages of Recursion
  - Recursion makes program elegant and cleaner.
- Disadvantages of Recursion
  - Recursions use more memory and are generally slow.
- ***Properties of recursion:***
  - **Base case Property** : There must be at least one condition in recursive function which do not involve the call to recursive routine
  - The invoking of each recursive call must reduce to some manipulation and **must go closer to Base case condition**
- **How to define recursion?**
  - **Syntax:**

```
    return_type  recursion_name( parameter list )
   {
      base case property;
      recursive call;
   }
```



Write a C Program to generate **fibonocci series** using recursion

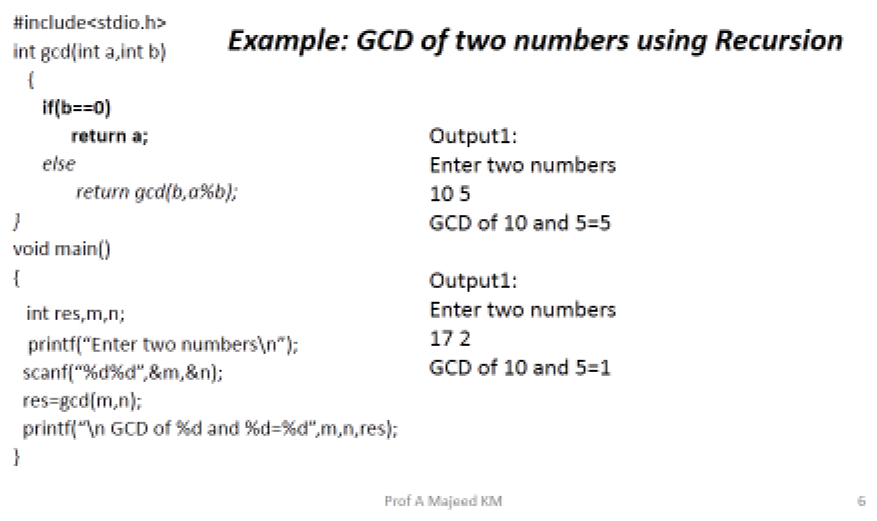
```
#include<stdio.h>
int Fib (int n)
{
  if ( n == 0 )
       return 0;
   else if ( n == 1 )
       return 1;
else
      return ( Fib(n-1) + Fib (n-2) );
}
void main()
{
  int i,n;
  printf("Enter a number\n");
  scanf("%d",&n);
  printf("Fibonocci Series:\n");
  for(i=0;i<n;i++)
  printf("%d \t",Fib(i));
}
```

```
Output1:
Enter a number
5
Fibonocci Series:
0    1    1    2    3
```

Prof A Majeed KM 4

Using Recursion Develop C program:

a. To find GCD of two number
b. To convert binary to decimal
c. To find factorial of n



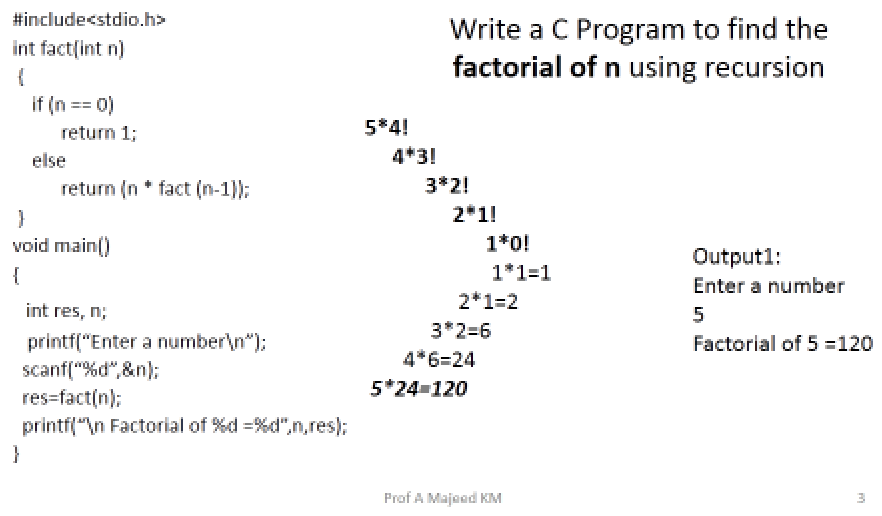
Example: GCD of two numbers using Recursion

```
#include<stdio.h>
int gcd(int a,int b)
  {
    if(b==0)
        return a;
    else
         return gcd(b,a%b);
}
void main()
{
  int res,m,n;
  printf("Enter two numbers\n");
 scanf("%d%d",&m,&n);
 res=gcd(m,n);
 printf("\n GCD of %d and %d=%d",m,n,res);
}
```

```
Output1:
Enter two numbers
10 5
GCD of 10 and 5=5

Output1:
Enter two numbers
17 2
GCD of 10 and 5=1
```

Prof A Majeed KM 6



Write a C Program to find the **factorial of n** using recursion

```
#include<stdio.h>
int fact(int n)
 {
   if (n == 0)
       return 1;
   else
       return (n * fact (n-1));
 }
void main()
{
  int res, n;
  printf("Enter a number\n");
 scanf("%d",&n);
 res=fact(n);
 printf("\n Factorial of %d =%d",n,res);
}
```

```
5*4!
   4*3!
      3*2!
         2*1!
            1*0!
             1*1=1
          2*1=2
       3*2=6
    4*6=24
5*24=120
```

```
Output1:
Enter a number
5
Factorial of 5 =120
```

Prof A Majeed KM 3

Program: Binary to decimal convertor

```
#include <stdio.h>

int convert(int num)
{
  if (!(num / 10)) return (num); //Base case property
   return (num % 10 + convert(num / 10) * 2); //Recursive call for convert()
  }

int main()
{
   int n;
   printf("Enter a binary number: ");
   scanf("%d", &n);
   printf("%d in binary = %d in decimal", n, convert(n));
   return 0;
}
```

Output 1:

:

```
Enter a binary number: 1010

1010 in binary = 10 in decimal
```

Output 2:

```
Enter a binary number: 11111111

11111111 in binary = 255 in decimal
```