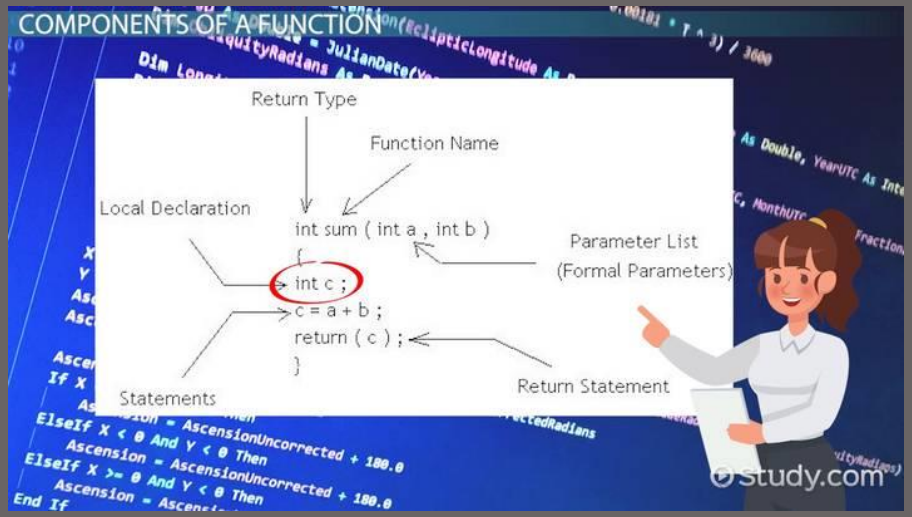
Functions in C

Lecture 07 – ICT1132



Piyumi Wijerathna
Department of ICT
Faculty of Technology

Overview

- Introduction to Functions
- Function definition
- Function declaration
- Invoking functions
- Placement of functions in the program
- Arguments and Parameters
- Passing Arguments to a function
- Rules of writing functions
- Recursive functions

Introduction to Functions

- Real world **problems are very large and complex** and difficult to implement at once.
- The best way to develop and maintain large programs is to **construct them as sub-problems / modules**.
- **Break** the initial problem into **modules.**
- In C language these **modules are called functions**.
- **Implement the modules** as functions.

What is a function

- A function is a block of code that performs a specific task.
- We can divide up our code into many separate functions.
- Every C program has at least one function, which is the **main().**

Types of functions in C

- **Standard Library functions(predefined / built-in functions)**
  - ✓ Functions included in header files.
  - ✓ Handle tasks such as mathematical computations, String handling, I/O Processing etc.

  Ex: printf() comes from stdio.h to display the output on the screen.

- **User-defined functions**
  - ✓ Functions created by the user.
  - ✓ Can create many user defined functions in a program.

Benefits of using functions

- **Easy Understanding**

  Dividing complex problem into small components makes program easy to understand.

- **Code Re-usability**

  same function can be used in any program without writing the same code again.

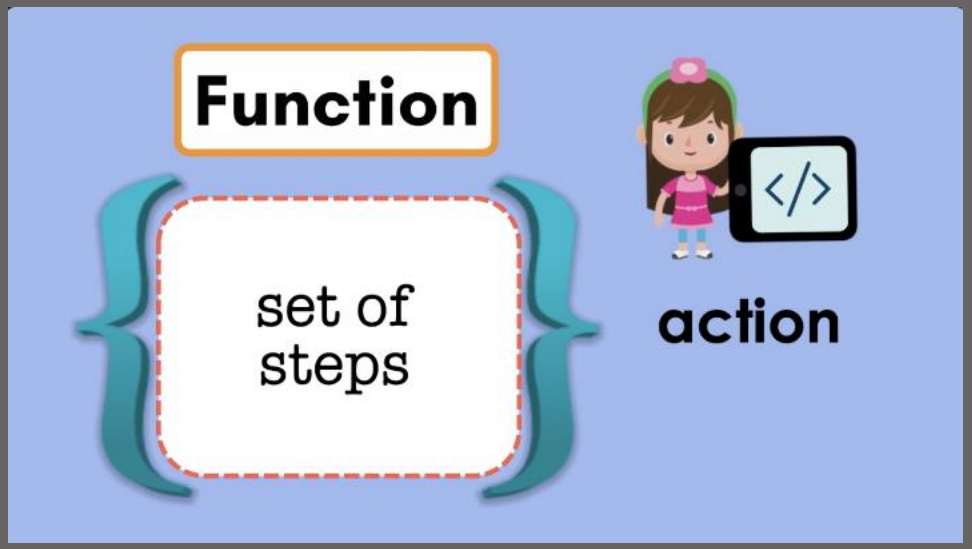
- **Easy developing and maintaining**

  In case of large programs with thousands of code lines, editing and debugging can be divided among many programmers.

- **Reduces the size of the code**

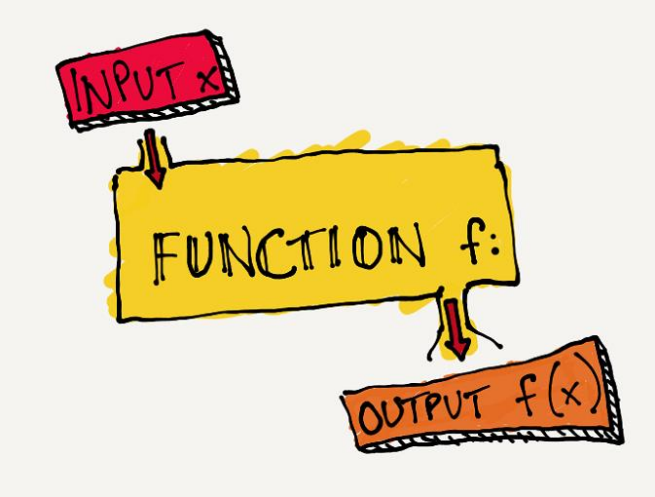
  duplicate set of statements are replaced by function calls.

C function definition, function declaration and function call



There are 3 aspects in each C function. They are,

- Function definition.
- Function declaration or prototype.
- Function call.



1. Defining a function in C

- Gives a definition of what the function has to do.
- A **function definition** consists of a **function header** and a **function body.**

```
return_type function_name( parameter list )
{
        body of the function
}
```

The function header consists of

- Return type
- Function name
- Parameter List

Signature of the function

- **Function name + Parameter List** called as the **signature of the function.**

```
int findSquare(int a)
{
        int sq;
        sq = a*a;
        return sq;
}
```

Return type

Parameter

Function Name

Body

Return-type

- Function can return **a value** when it is calling.
- The return_type is the **data type** of the value the function returns (int, char, float etc.)
- Some functions perform its operations **without returning a value.** In this case, the return_type is **"void".**

Function returns a value

```
int findSquare(int a)
{
        int sq = a*a;
        return sq;
}
```

Function do not return a value

```
void findSquare(int a)
{
        int sq = a*a;
        printf("sq=%d",sq);
}
```

Function-name

- Function name is the name of the function and should be unique in a program.
- It is an identifier, therefore must follow the same rules of defining identifiers in C.

Note: Memorize about rules of identifiers in C.

Parameter-list

- Parameter list contains variables names along with their data types.
- These are kind of **inputs for the function** and giving an input is done when calling the function.
- These values are referred to as **arguments.**
- The parameter list refers to the **type**, **order**, and **number of parameters** of a function.
- Parameters are **optional**; that is, a function may contain no parameters.

The Body of a function

- Body implementation done when **defining the function.**
- It contains a collection of statements that define what the function does.
- The body is enclosed within curly braces { } and can consists of three parts.

1. local variable declaration.
2. function statement/s that performs the tasks of the function.
3. A return statement

   that return a value.

Type of the return value should be same as the return type of the function.

Example:

```
int addition(int x, int y)
{
        int add;
        add = x+y;
        return add;
}
```

2. Function declaration / Prototype

- functions should be **declared before the first call** within a C program (same as variables).
- Informs compiler about the return type, function name and parameters.

Syntax:

```
return-type function name (parameter list);
```

Example:

```
int addnum( int num1, int num2);
```

- Parameter names are not important in function declaration, only their type is required.

```
int addnum( int , int);
```

Cont.

- Function declaration is required when,

  Define the function after the main function of the program.

- In such case, you should declare the function at the top of the file before calling the function in the main.

```
#include<stdio.h>
int addsum(int x, int y);
int main(){
        ---------
        --------
        //function calling
        --------
}

int addsum(int x, int y)
{
        return x+y;
}
```

```
#include<stdio.h>
int addsum(int x, int y)
{
        return x+y;
}

int main(){
        ---------
        --------
        //function calling
        --------
}
```

Function Prototype Examples

- int addnumbers (int num1, int num2);
- float F_To_C (int Fahrenheit);
- void instructions (void); → void instructions ( );
- int findAvg (int count, float sum);

The easiest way to build a function prototype is to copy the function header and place a semicolon at the end of it.

3. Function Call/ Invocation

To use a function, have to call that function to perform the defined task.

```
#include <stdio.h>

void printMessage();

int main()
{
   printMessage();
   return 0;
}
```

Function Declaration

Calling Function

Defining the called Function

```
void printMessage()
{
   printf("**********\n”;
   printf(“* Hello  *“\n);
   printf("**********\n“);
}
```

Invoking Functions

- Functions can be invoked, by using their **name followed by the argument list**, which is enclosed in parentheses.
- The return value of the function can be stored into a **variable of the same type as the return type of the function.**

```
Eg: add = addnumbers( 10, 20 );
```

- Functions which **do not return values** (sometimes referred as procedures or void functions) can be invoked by using **only the function name and its argument list** as a statement in the program.
- If a function returns a value and is invoked in this manner the value is discarded.

```
Eg: printMessage();
```

Calling functions from another C program

- **Syntax:** #include “fileName.c”

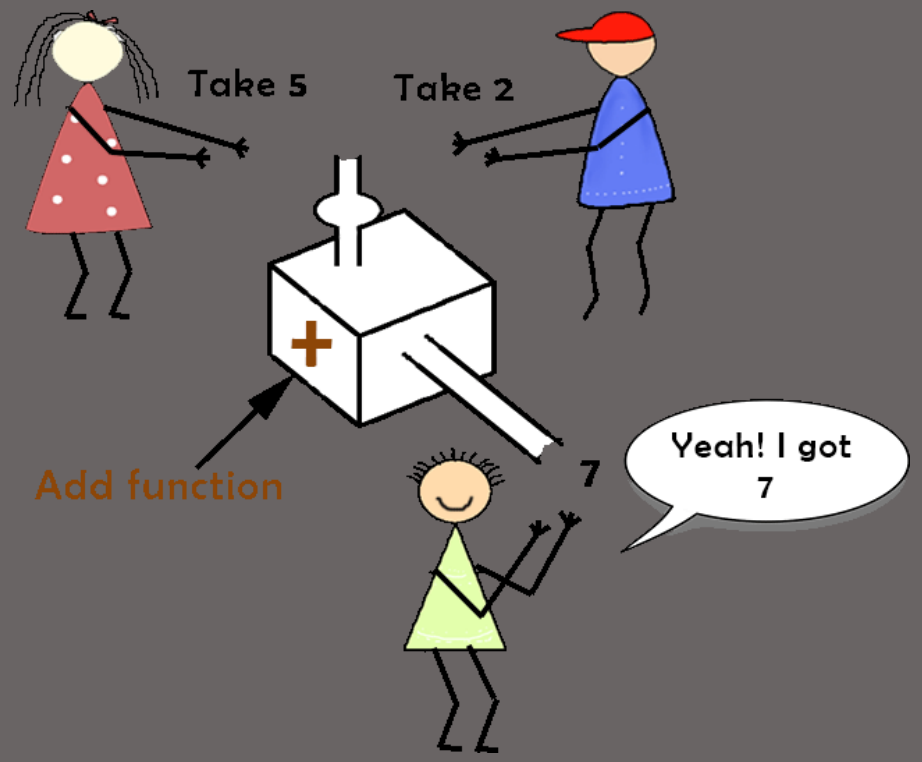
- Function defined in ExtFunc.c File

```
void callFunc(float x)
{

        printf("Func%f",x);
}
```

- Function calling into Ex1.c file

```
#include<stdio.h>
#include "ExtFunc.c"

void main(){
        callFunc(5.5);
}
```

How User Defined Functions Work?



- The execution of a C program begins from the **main()** function.

- When the compiler founds the **functionName();** inside the main function, control of the program jumps to the function
  void functionName()

- And, the compiler starts executing the codes inside the user-defined function.

- After execution within the function, the control of the program jumps to statement after the statement,
  functionName();

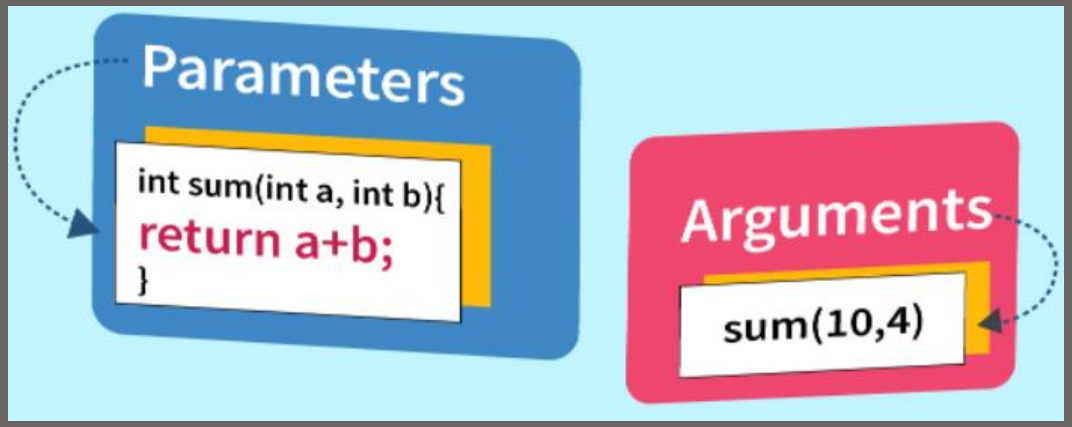
```
#include <stdio.h>

void functionName()
{
    ... .. ...
    ... .. ...
}

int main()
{
    ... .. ...
    ... .. ...

    functionName();

    ... .. ...
    ... .. ...
}
```

Placement of functions in the Program

- **Function definitions** are placed after or before the main program.
- **Function declarations/prototypes** are generally placed either above the main program or at the top of the main program, before the variable declarations.
- The placement of the **function invocation**, depends on the place you want the function to be executed.

```
#include<stdio.h>

//void callFunc(float);

void main(){

        void callFunc(float);
        callFunc(5.5);
}

void callFunc(float x){

        printf("Func%f",x);
}
```

Arguments and Parameters



- The term **argument (Actual Parameter)** refers to the **values passed** into the function. The arguments appear in the **invocation** of the function.

arguments

```
int main(){
  int add,num1=10,num2=20;
  add = addnumbers(num1, num2);
  //add = addnumbers(10, 20);
  return 0;
}
```

- The term **parameter (Formal Parameter)** refers to the **variables declared** in the function heading and used by the function to hold the information passed to it.

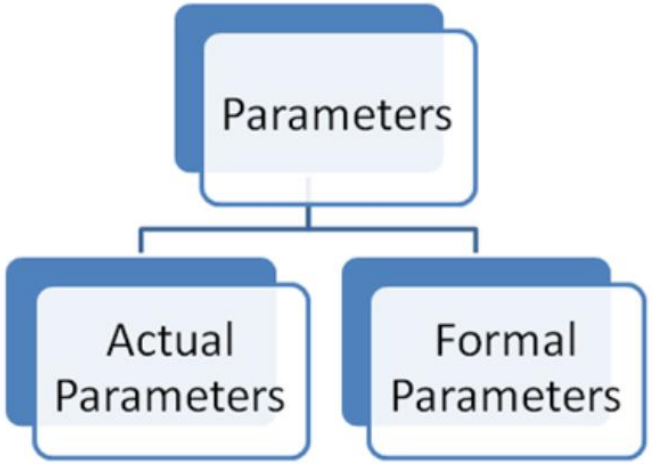
parameters

```
int addnumbers(int num1, int num2);
```

```
class Addition
{
        int sum( int p, int q )
        {
            return p+q;
        }
        public static void main()
        {
                Addition ad=new Addition();
                int a=10,b=20;
                int c=sum( a,b );
                System.out.print("Result is " + c);
        }
}
```

Formal Parameter

Actual Parameter



Arguments and Parameters

- The first argument matches with the first parameter, the second argument matches the second parameter, and so on.
- The argument and the parameter do not have to have the same name.
- The argument may be a constant, a variable, or an expression, when using pass by value.
- The parameter must be a variable.

```
void calValue (int a, float b)
{
        //function body
}

void main(){
        const int x;
        float y;
        -------------
        calValue(x, y);
}
```

Example 01

Write a program to find and print the maximum between two numbers using a function.

```c
#include <stdio.h>

void max(int num1, int num2);  //Declaration/Prototype

void main(){

    int x,y;

    printf("Enter first number: \n");
    scanf("%d", &x);
    printf("Enter second number: \n");
    scanf("%d", &y);

    max(x,y);  // Invoking the function
}

void max(int num1, int num2)  // Defining the function
{
    int ans;   // local variable
        if (num1 > num2)
            ans = num1;
        else
            ans = num2;

        printf("Maxmum Number: %d", ans);
}
```

Modify the program to return the maximum number from the function.

```c
#include <stdio.h>

int max(int num1, int num2);  //Declaration/Prototype

void main()
{
  int x, y, result;

  printf("Enter first number: \n");
  scanf("%d", &x);
  printf("Enter second number: \n");
  scanf("%d", &y);

  result = max(x,y);                // Invoking the function
  printf("Maxmum No: %d", result);
}

int max(int num1, int num2)     // Defining the function
{
    int ans;
                                  // local variable
    if (num1 > num2)
        ans = num1;
    else
        ans = num2;

    return ans;
}
```

Example 02

Write a C program to input any number from user and find cube of the given number using function. **Input the number inside the function.**

Example:
Input any number: 5
Output: 125

```c
#include <stdio.h>

float num;  //Global variable

// Function to find cube of any number
float cube()
{
    //float num;

    printf("Enter any number: ");
    scanf("%f", &num);

    return (num * num * num);
}

int main()
{
    float c;

    c = cube();   //Function invoking
    printf("Cube of %f is %f\n", num, c);

    return 0;
}
```

Modify the same program to input the number inside main function.

```c
#include <stdio.h>

// Function to find cube of any number
float cube(float num)
{
    return (num * num * num);
}

int main()
{
    float num;
    float c;

    printf("Enter any number: ");
    scanf("%f", &num);

    c = cube(num);    //Function invoking
    printf("Cube of %f is %f\n", num, c);

    return 0;
}
```

Example 03

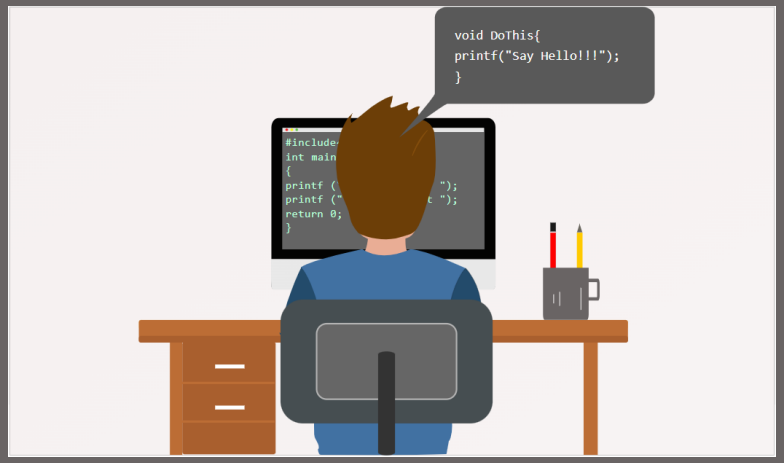
Write a C program to find diameter, circumference and area of a circle using functions. (Reads radius of the circle from user).

Example:
Input radius: 10
Output diameter: 20
Output circumference: 62.83
Output area: 314.16

Functions declarations

```
double diameter (double radius);
double circumference (double radius);
double area (double radius);
```

Rules of Writing functions



Rules of writing functions

1. C program is a collection of one or more functions.

```
void main()
{
        addition();
        subtraction();
        multiplication();
}
```

2. A function gets called when the function name is followed by a semicolon.

```
void main( )
{
        display( ) ;
}
```

3. A function is defined when function name is followed by a pair of braces in which one or more statements may be present.

```
void display( )
    {
        statement 1 ;
        statement 2 ;
        statement 3 ;
    }
```

4. Any function can be called within any other function(even the main).

```
void main( ){
    message( ) ;
}
void message( ){
    printf ( "\nWe are going to call main" ) ;
    main( ) ;
    addSum( );
}
```

5. A function can be called any number of times.

```
void main(){
    message( );
    message( );
}
void message(){
    printf("\nLearning C is very Easy");
}
```

6. The order in which the functions are defined in a program and the order in which they get called need not to be same.

```
void main( ){
    message1( ) ;
    message2( ) ;
}
void message2( ){
    printf ( "\n I am First Defined But Called Later " ) ;
}
void message1( ){
    printf ( "\n I am Defined Later But Called First ") ;
}
```

7. A function can call itself (Process of Recursion)

```
int fact(int n)
{
        if(n==0)
                return (1);
        else
                return (n * fact(n-1));
}
```

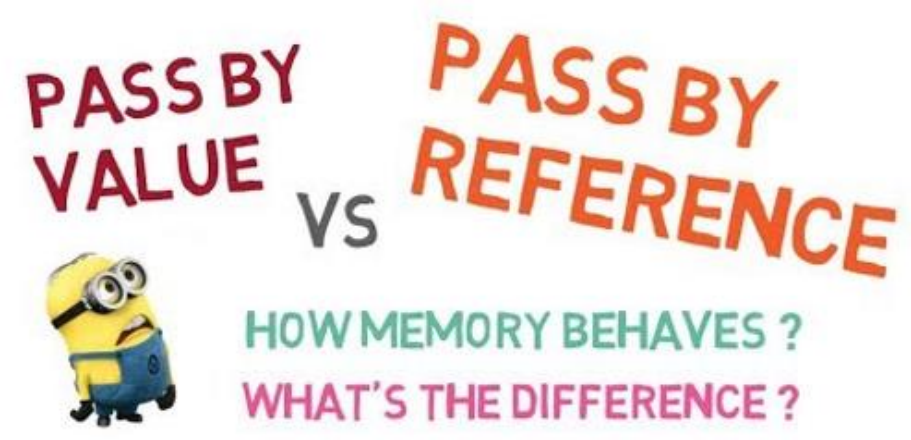
8. C does not support Function Overloading. So **two functions with same name are not allowed**.

(declaring more than one function with the same name & scope)

```c
#include<stdio.h>

//void addSum(float x, float y);
void addSum(float x, float y, float a);

void main(){

        addSum(2.5,3.3,5.5);
}

/*void addSum(float n1, float n2){
        float sum = n1+n2;
        printf("Sum of two numbers=%f",sum);
}*/

void addSum(float n1, float n2, float n3){
        float sum = n1+n2+n3;
        printf("Sum of three numbers=%f",sum);
}
```

If you uncomments the comments, program will generate errors.

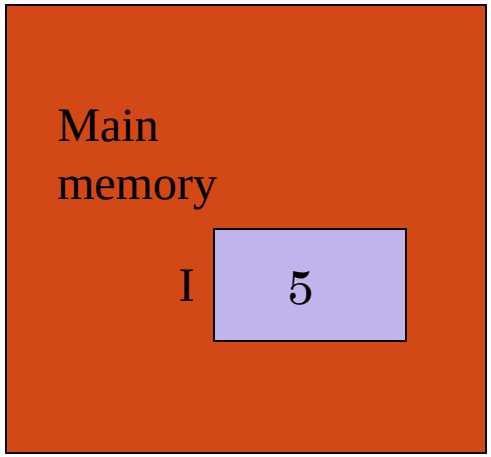
Parameter Passing



1. Pass by Value

- It is a method of passing information to a function whereby the **parameter receives a copy of the value of the argument**.
- Different memory is allocated for both argument and parameter.
- Any changes that the function makes to the parameter are **made to the copy and not to the original** argument.
- Hence, the **original argument values are unchanged,** only the parameters inside function changes.

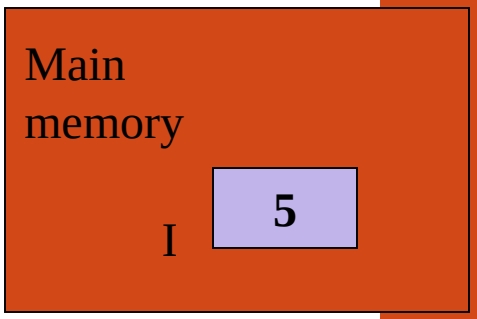
Example (Pass by Value)

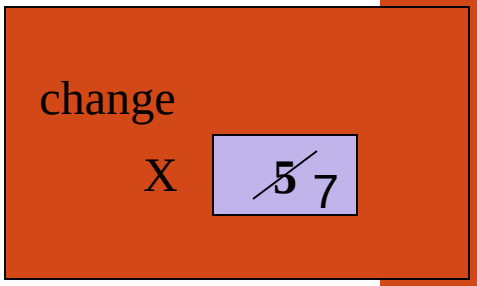



```
void change(int X);

int main(){
   int I=5;
   printf("First time I is %d\n",I);
   change(I);
   printf("Next time I is %d\n",I);
}

void change(int X)
{
 printf("Entering function X is %d\n",X);
 X = 7;
 printf ("Leaving function X is %d\n",X);
}
```

```
First time I is 5
Entering function X is 5
Leaving function X is 7
Next time I is 5
```

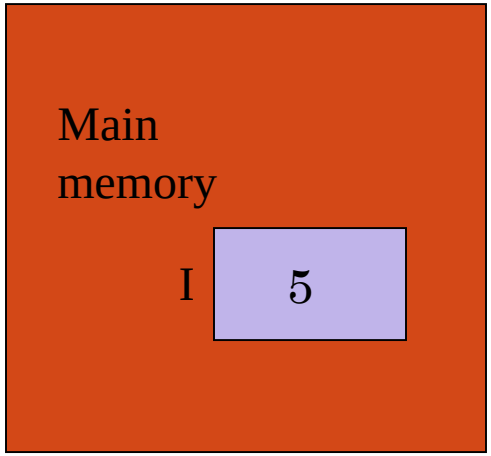
```
void change(int X);
int main(){
   int I=5;
   printf("First time I is %d\n",I);
   change(I);
   printf("Next time I is %d\n",I);
}
void change(int X){
 printf("Entering function X is %d",X);
 X = 7;
 printf ("Leaving function X is %d",X);
}
```






```
First time I is 5
Entering function X is 5
Leaving function X is 7
Next time I is 5
```

```
void change(int X);

int main(){
   int I=5;
   printf("First time I is %d\n",I);
   change(I);
   printf("Next time I is %d\n",I);
}

void change(int X){
 printf("Entering function X is %d",X);
 X = 7;
 printf ("Leaving function X is %d",X);
}
```




```
First time I is 5
Entering function X is 5
Leaving function X is 7
Next time I is 5
```

Example 02

```
#include<stdio.h>

void interchange(int num1, int num2)
{
    int temp;
    temp = num1;
    num1 = num2;
    num2 = temp;
}

int main() {

    int num1=50,num2=70;
    interchange(num1,num2);

    printf("\nNumber 1 : %d",num1);
    printf("\nNumber 2 : %d",num2);

    return 0;
}
```

Output :

Number 1 : 50
Number 2 : 70

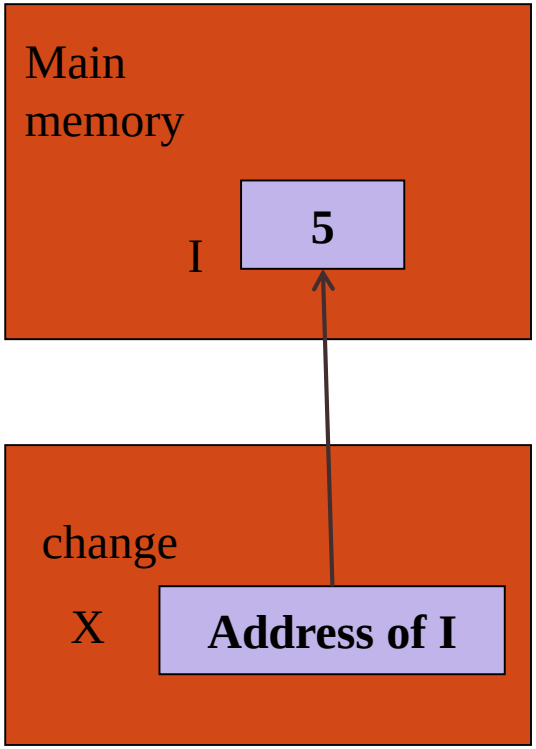
Example 02

```
#include<stdio.h>

void interchange(int num1, int num2)
{
    int temp;
    temp = num1;
    num1 = num2;
    num2 = temp;
    printf("\nInside Function Number 1 : %d",num1);
    printf("\nInside Function Number 2 : %d",num2);
}
int main() {

    int num1=50,num2=70;
    interchange(num1,num2);

    printf("\nNumber 1 : %d",num1);
    printf("\nNumber 2 : %d",num2);

    return 0;
}
```

```
Inside Function Number 1 : 70
Inside Function Number 2 : 50
Number 1 : 50
Number 2 : 70
```

2. Pass by Reference

- Copies the address of the argument into the parameter.
- Any changes that the function makes to the parameter are made to the original argument.



Will be discussed in Pointers Lecture.

Function Recursion

- **Call a function inside the same function**, then it is called a recursive call of the function.

Example:

```
void recursion() {
        recursion();     /* function calls itself */
 }
int main(){
        recursion();
}
```

How does recursion work?

```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```

recursive call

Find the addition of n numbers by recursion:

Without Using Loops

```c
#include <stdio.h>
int getSum(int n);

int main()
{
    int n,sum;
    printf("Enter a positive integer: ");
    scanf("%d", &n);

    //printf("Sum = %d",getSum(n)); or
    sum = getSum(n);
    printf("Sum = %d",sum);

    return 0;
}

int getSum(int n)
{
    int sum;
    if(n > 0)
        sum = n + getSum(n-1);
    else
        sum = n;
return sum;
}
```

Exercise

- Write a C program to calculate the factorial of a given number using a recursive function. (Number should be entered by the user).


Any
Question

THANK YOU… !