# Structured Programming Language
# PGDIT 101

Professor Dr. Mohammad Abu Yousuf

# Functions

# What is a function?

- A function is a self-contained program segment that carries out some specific, well-defined task. Every C program consists of one or more functions.
- One of these functions must be called main.
- Execution of the program will always begin by carrying out the instructions in main. Additional functions will be subordinate to main, and perhaps to one another.
- A function will carry out its intended action whenever it is accessed (i.e., whenever the function is "called") from some other portion of the program. The same function can be accessed from several different places within a program.

- A function definition has two principal components: the first line (including the argument declarations), and the body of the function.

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

- **Return Type:** A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.
- **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.
- **Parameters**: A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This parameter is referred to as actual parameter or argument.

# Defining a Function

```
#include<stdio.h>
int twice( int x)
{
   x=x+x;
   return x;
}
int main()
{
   int x=10,y;
   y=twice(x);
   printf("%d,%d\n",x,y);
}
```

- Example:

```
int  twice(int x)
{
   x=x+x;
   return x;
}
```

```
/* function returning the max between two numbers */
int max(int num1, int num2)
{
   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

- Information is returned from the function to the calling portion of the program via the ***return*** statement.
- In general terms, the ***return*** statement is written as

  return ***expression;***
- The value of the expression is returned to the calling portion of the program. The expression is optional. If the expression is omitted, the r e t u r n statement simply causes control to revert back to the calling portion of the program, without any transfer of information.

# Local Variables

- Local Variables

```
int func1 (int y)
{
   int a, b = 10;
   float rate;
   double cost = 12.55;
   .......
}
```

- Those variables declared “within” the function are considered “local variables”.
- They can only be used inside the function they were declared in, and not elsewhere.

# Calling a Function

- While creating a C function, we give a definition of what the function has to do. To use a function, we will have to call that function to perform the defined task.
- When a program calls a function, program control is transferred to the called function. A called function performs defined task, and when its return statement is executed or when its function-ending closing brace is reached, it returns program control back to the main program.
- A function can be accessed (i.e., called) by specifying its name, followed by a list of arguments enclosed in parentheses and separated by commas.

# Calling a Function (Cont…)

- The arguments appearing in the function call are referred to as actual arguments, in contrast to the formal arguments that appear in the first line of the function definition. (They are also known simply as arguments, or as actual parameters.)

```
#include<stdio.h>
int  twice(int x) {
   x=x+x;
   return x;
 }
int  main() {
   int x=10,y;
   y=twice(x);  // calling the function
   printf("%d,%d\n",x,y);
}
```

```c
#include <stdio.h>

/* function declaration */
int max(int num1, int num2);

int main ()
{
   /* local variable definition */
   int a = 100;
   int b = 200;
   int ret;

   /* calling a function to get max value */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}
int max(int num1, int num2)
{
   /* local variable declaration */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

# Function prototype

- Many programmers prefer a "top-down" approach, in which main appears ahead of the programmer-defined function definition.
- In such situations the function access (within main) will precede the function definition. This can be confusing to the compiler, unless the compiler is first alerted to the fact that the function being accessed will be defined later in the program. **A function prototype** is used for this purpose.
- Function prototypes are usually written at the beginning of a program, ahead of any programmer-defined functions (including **main).**

**return_type function_name (parameter list )**

# Example: Function Prototype

```
#include<stdio.h>
int  twice(int x) {
   x=x+x;
   return x;
 }
int  main() {
   int x=10,y;
   y=twice(x);
   printf("%d,%d\n",x,y);
}
```

```
#include<stdio.h>
int  twice(int x) ; // Prototype
int  main() {
   int x=10,y;
   y=twice(x);
   printf("%d,%d\n",x,y);
}
int  twice(int x) {
   x=x+x;
   return x;
 }
```

# Different aspects of function calling

function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

- function without arguments and without return value
- function without arguments and with return value
- function with arguments and without return value
- function with arguments and with return value

# Different aspects of function calling

//function without arguments and without return value

```
1.#include<stdio.h>
2.void printName();
3.void main ()
4.{
5.    printf("Hello ");
6.    printName();
7.}
8.void printName()
9.{
10.    printf(“CSE");
11.}
```

Output:
Hello CSE

//function without arguments and with return value

```
1.#include<stdio.h>
2.int sum();
3.void main()
4.{
5.    int result;
6.    printf("\nGoing to calculate the sum:");
7.    result = sum();
8.    printf("%d",result);
9.}
10.int sum()
11.{
12.    int a,b;
13.    printf("\nEnter two numbers");
14.    scanf("%d %d",&a,&b);
15.    return a+b;
16.}
```

# Different aspects of function calling

//function with arguments and without return value

```
1.#include<stdio.h>
2.void sum(int, int);
3.void main()
4.{
5.    int a,b,result;
6.    printf("\nGoing to calculate: ");
7.    printf("\nEnter two numbers:");
8.    scanf("%d %d",&a,&b);
9.    sum(a,b);
10.}
11.void sum(int a, int b)
12.{
13.    printf("\nThe sum is %d",a+b);
14.}
```

//function with arguments and with return value

```
1.#include<stdio.h>
2.int sum(int, int);
3.void main()
4.{
5.    int a,b,result;
6.    printf("\nGoing to calculates:");
7.    printf("\nEnter two numbers:");
8.    scanf("%d %d",&a,&b);
9.    result = sum(a,b);
10.   printf("\nThe sum is : %d",result);
11.}
12.int sum(int a, int b)
13.{
14.    return a+b;
15.}
```

# C Library Functions

Library functions are the inbuilt function in C that are grouped and placed at a common place called the library. Such functions are used to perform some specific operations.

| SN | Header file | Description |
|---|---|---|
| 1 | stdio.h | This is a standard input/output header file. It contains all the library functions regarding standard input/output. |
| 2 | conio.h | This is a console input/output header file. |
| 3 | string.h | It contains all string related library functions like gets(), puts(),etc. |
| 4 | stdlib.h | This header file contains all the general library functions like malloc(), calloc(), exit(), etc. |
| 5 | math.h | This header file contains all the math operations related functions like sqrt(), pow(), etc. |
| 6 | time.h | This header file contains all the time-related functions. |
| 7 | ctype.h | This header file contains all character handling functions. |

# Call by value and Call by reference

- In call by value method, the value of the actual parameters is copied into the formal parameters.

- In call by reference, the address of the variable is passed into the function call as the actual parameter.

# Call by value

- When a single value is passed to a function via an actual argument, the value of the actual argument is copied into the function. ***Therefore, the value of the corresponding formal argument can be altered within the function, but the value of the actual argument within the calling routine will not change.*** This procedure for passing the value of an argument to a function is known as **passing by value**.

# Call by value

```
1.#include<stdio.h>
2.void change(int num) {
3.    printf("Before adding value inside function num=%d \n",num);
4.    num=num+100;
5.    printf("After adding value inside function num=%d \n", num);
6.}
7.int main() {
8.    int x=100;
9.    printf("Before function call x=%d \n", x);
10.  change(x);//passing value in function
11.  printf("After function call x=%d \n", x);
12.  return 0;
13.}
```

Ouptput:
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=100

# Call by value

```
1.#include <stdio.h>
2.void swap(int , int); //prototype of the function
3.int main()
4.{
5.    int a = 10;
6.    int b = 20;
7.    printf("Before swapping the values in main a = %d, b = %d\n",a,b);
8.    swap(a,b);
9.    printf("After swapping values in main a = %d, b = %d\n",a,b);
10.}
11.void swap (int a, int b)
12.{
13.    int temp;
14.    temp = a;
15.    a=b;
16.    b=temp;
17.    printf("After swapping values in function a = %d, b = %d\n",a,b);
```

Output:
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 10, b = 20

# Example: Passing by value

```
#include <stdio.h>

void modify(int a);       /* function prototype */

main()
{
    int a = 2;

    printf("\na = %d  (from main, before calling the function)", a);
    modify(a);
    printf("\n\na = %d  (from main, after calling the function)", a);
}

void modify(int a)
{
    a *= 3;
    printf("\n\na = %d  (from the function, after being modified)", a);
    return;
}
```

# Example: Passing by value (Cont..)

- The original value of a (i.e., a = 2) is displayed when main begins execution. This value is then passed to the function modify, where it is multiplied by **3 and the new value displayed. Note that it is the *altered value of the formal argument*** that is displayed within the function. Finally, the value of a within main (i.e., the actual argument) is again displayed, after control is transferred back to main from modify.

- When the program is executed, the following output is generated.

  a = 2 (from main, before calling the function)

  a = 6 (from the function, after being modified)

  a = 2 (from main, after calling the function)

# Call by reference

- In call by reference, the address of the variable is passed into the function call as the actual parameter.

- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.

# Call by reference

```
1.#include<stdio.h>
2.void change(int *num) {
3.    printf("Before adding value inside function num=%d \n",*num);
4.    (*num) += 100;
5.    printf("After adding value inside function num=%d \n", *num);
6.}
7.int main() {
8.    int x=100;
9.    printf("Before function call x=%d \n", x);
10.   change(&x);//passing reference in function
11.   printf("After function call x=%d \n", x);
12.return 0;
13.}
```

Output:

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=200
```

# Difference between call by value and call by reference

| No. | Call by value | Call by reference |
|---|---|---|
| 1 | A copy of the value is passed into the function | An address of value is passed into the function |
| 2 | Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters. | Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters. |

Thank you