SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23ITT101- PROBLEM SOLVING & C PROGRAMMING

I YEAR /I SEMESTER

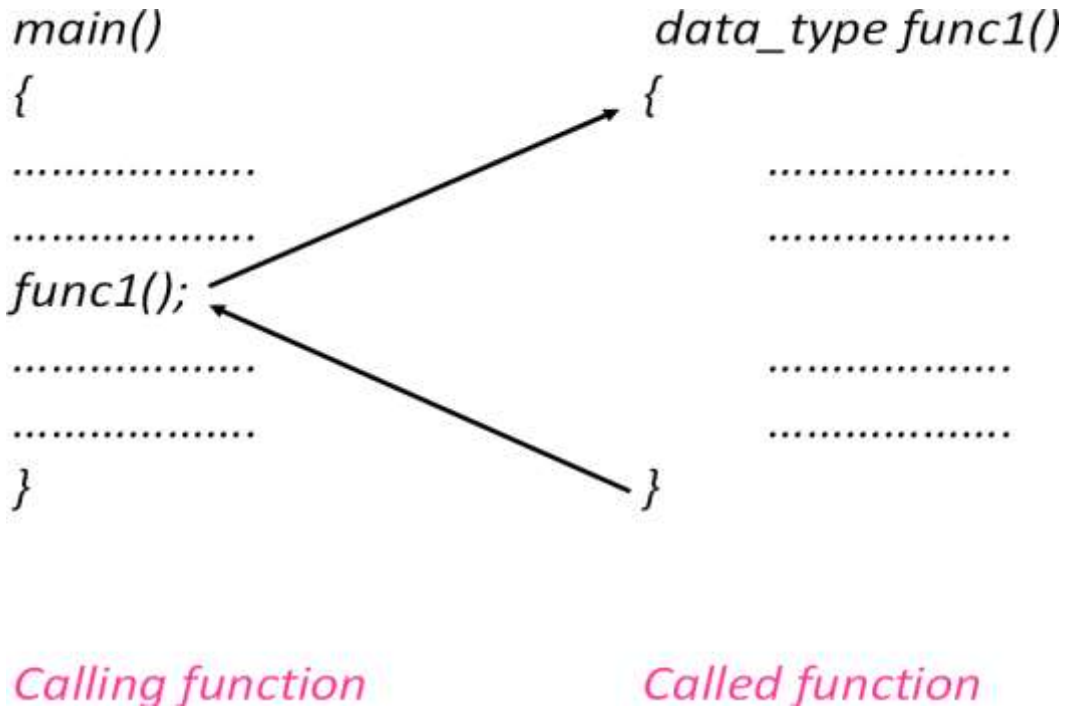
Unit IV – FUNCTIONS AND POINTERS

Topic : User Defined Function

Topics to be covered

- Function definition
- Function parameter
- Function body
- Return
- Function call

User Defined Function



DEFINITION OF FUNCTIONS

- A function definition, also known as function implementation shall include the following elements;
  - Function name;
  - Function type;
  - List of parameters;
  - Local variable declaration;
  - Function statements; and
  - A return statement.
- All six elements are grouped into two parts, namely,
  - Function header (First three elements) and
  - Function body (Second three elements)

THE FORM OF C FUNCTION

return-type function_name(arg-type name-1,...,arg-type name-n) ---- **Function header**

Parameter list

```
{
declarations;
statements;
return(expression);
}
```

Function Body

- The first line **function_type function_name(parameter list)** is known as **the function header.** **r**eturn-type is the function-type
- The statements within the opening and the closing brace constitute **the function body.**

FUNCTION DEFINITION

Function Header

- The function header consists of three parts: the function type (also known as return type), the function name and formal parameter list.
- Semicolon is not used at the end of the function header

Name and Type

- The function type specifies the type of value (like float or double) that the function id expected to return to the program calling the function
- If the return type is not explicitly specified, C assume it as an integer type.
- If the function is not returning anything then we need to specify the return type as void
- The function name is any valid C identifier and therefore ,just follow the same rules of formation as other variable names in C

FORMAL PARAMETER LIST

- The parameter list declares the variables that will receive the data sent by the calling program.
- They serve as input data to the function to carry out the specified task.
- They represent actual input values, they are often referred to as formal parameters.
- These parameters can also be used to send values to the calling programs

FORMAL PARAMETER LIST

- The parameter is known as arguments.
  - float quadratic (int a, int b, int c) { ….. }
  - double power (double x, int n) { ….. }
  - int sum (int a, int b) { ….. }
- There is no semicolon after the closing parenthesis
- The declaration parameter variables cannot be combined

FORMAL PARAMETER LIST

- To indicate that the parameter list is empty, we use the keyword void between the parentheses as in

```
void printline (void)
{
…
}
```

- Many compiler accept an empty set of parentheses

```
void printline()
```

- It is good to use void to indicate a nil parameter list

FUNCTION BODY

- The function body contains the declarations and statements necessary for performing the required task. The body enclosed in braces, contains three parts,
  - Local declarations that specify the variables needed by the function
  - Function statements that perform the task of the function
  - A return statement that returns the value evaluated by the function

- If a function does not return any value, we can omit the return statement.

- Its return type should be specified as void

RETURN VALUES AND THEIR TYPES

A function may or may not send back any value to the calling function

- Done through return statement
- It is possible to send any number of values to the called function
- The called function can only return one value per call

SYNTAX:

return;

or

return (expression);

RETURN VALUES AND THEIR TYPES

- return;
  - Plain return does not return any value
  - Acts as the closing brace of the function
  - The control is immediately passed back to the calling function

EXAMPLE:

```
if(error)  return;
        return (expression);
```

  - Return the value of the expression

RETURN VALUES AND THEIR TYPES

EXAMPLE:

```
mul(x,y)
int x,y;
{
int p;
p = x*y;  return(p);
}
```

- Returns the value of p which is the product of the values of x and y
- The last statement can be combined into one statement as return(x*y);

RETURN VALUES AND THEIR TYPES

- A function may have more than one return statements
- This situation arises when the value returned is based on certain conditions

EXAMPLE:

```
if(x <= 0)  return(0);  else  return(1);
```

RETURN VALUES AND THEIR TYPES

- Return type of data:

– All function by default return int type data

– We can force a function to return a particular type of data by using type specifier in the function header

EXAMPLE:

```
double product(x,y)

float sqr_root(p)
```

- When a value is returned, it is automatically cast to the function's type

RETURN VALUES AND THEIR TYPES

```
/*  C program to find SUM of two integer
Numbers using User Define Functions.*/

#include<stdio.h>

/*function declarations*/
int     sumint(int,int); /*to get sum*/
int main()
{
    int n1,n2;
    int sum;
    float avg;

    printf("Enter the first integer number: ");
    scanf("%d",&n1);

    printf("Enter the second integer number: ");
    scanf("%d",&n2);
```

```
     /*function calling*/
    sum=sumint(n1,n2);
  printf("Number1: %d, Number2: %d\n",n1,n2);
    printf("Sum: %d\n",sum);
     return 0;
}

/*function definitions*/
/* Function     : sumint
 * Arguments    : int,int - to pass two integer values
 * return type  : int - to return sum of values
*/
int sumint(int x,int y)
{
    /*x and y are the formal parameters*/
    int sum;
    sum=x+y;
    return sum;
}
```

CALLING A FUNCTION

- A function can be called by simply using the function name in a statement
- When the function encounters a function call, the control is transferred to the function mul(x,y)

```
main()
{
int p;
p = mul(10,5);
printf("%d\n", p);
}
int mul(int x,int y)
{
int p;          /*local variables*/
p = x * y;/* x = 10, y = 5*/
return(p);
}
```

CALLING A FUNCTION

- The function is executed and the value is returned and assigned to p
- A function which returns a value can be used in expressions like any other variables

```
printf("%d\n", mul(p,q));
 y = mul(p,q) / (p+q);
If (mul(m,n) > total)  printf("large");
```

CALLING A FUNCTION

- A function cannot be used on the right side of an assignment statement

```
mul(a,b) = 15;
```

- A function that does not return any value may not be used in expression; but can be called to perform certain tasks specified in the function

```
main()
{
printline();
 }
```

FUNCTION DECLARATION

- A function declaration consists of four parts
  - Function type (return type)
  - Function name
  - Parameter list
  - Terminating semicolon

Format

  - Function-type function-name (parameter list);

- Very similar to the function header line expect the terminating semicolon.
  - int mul(int m, int n); /* Function prototype */

FUNCTION DECLARATION

- Equally acceptable forms of declarations are
  - int mul (int, int);
  - mul (int a, int b);
  - mul (int, int);

- When a function does not take any parameters and does not return any value, its prototype is written as:

void display (void);

FUNCTION DECLARATION

- A prototype declaration may be placed in two places in a program.
  - Above all the functions (including main) (global prototype)
  - Inside a function definition (local prototype)

- It is good to declare prototypes in the global declaration section before main.

- It adds flexibility, provides an excellent quick reference to the functions used in the program, and enhances documentation

Summary

- Functions are modules in C. Programs combine user-defined functions with library functions.
- Function definition includes function name, arguments, body of the function that includes executable statements and an option return statement if function is returning any value.
- A function call is provided with function name and arguments (data) in the calling part of the program. Two types of calling functions include call-by-value and call-by-reference.


Thank
you