Unit III- Part 04
(Looping Statements in C)

Lecture Slide

AS2023

Objectives

By the end of this session, students will be able to:

- Explain all variants of looping statement
- Write a program using those iteration statements
- Choose the appropriate looping statement for the given problem
- Break statement
- Continue statement

Looping Statements

- A loop is defined as a block of statements which are repeatedly executed for certain number of times
- Terms in loops
  - **Initialization:** it is the first step in which starting and final value to the loop variable is assigned.
  - **Loop control variable:** variable used in the loop
  - **Increment or decrement:** it is a numerical value added to or subtracted from the variable in each round of the loop
- A Statement or a sequence of statements are executed until the conditions for the termination of the loop is satisfied
- Program loop consist of two segments
  - **Body of the loop** (consist of statements that are required to executed repeatedly)
  - **Control statement** (test certain conditions and directs the execution of body of the loop)

Looping Statements Cont.

- Types:
  - Entry controlled (pre-test loop)
  - Exit controlled (post-test loop)
- Entry controlled:
  - Test condition is evaluated before executing the statements inside the body of the loop
  - ***for*** and ***while*** loop
- Exit controlled loop
  - Statements inside the Body of the loop is executed before the evaluation of test conditions
  - ***do while*** loop
- **Home assignment:** Write the differences between exit and entry controlled loop

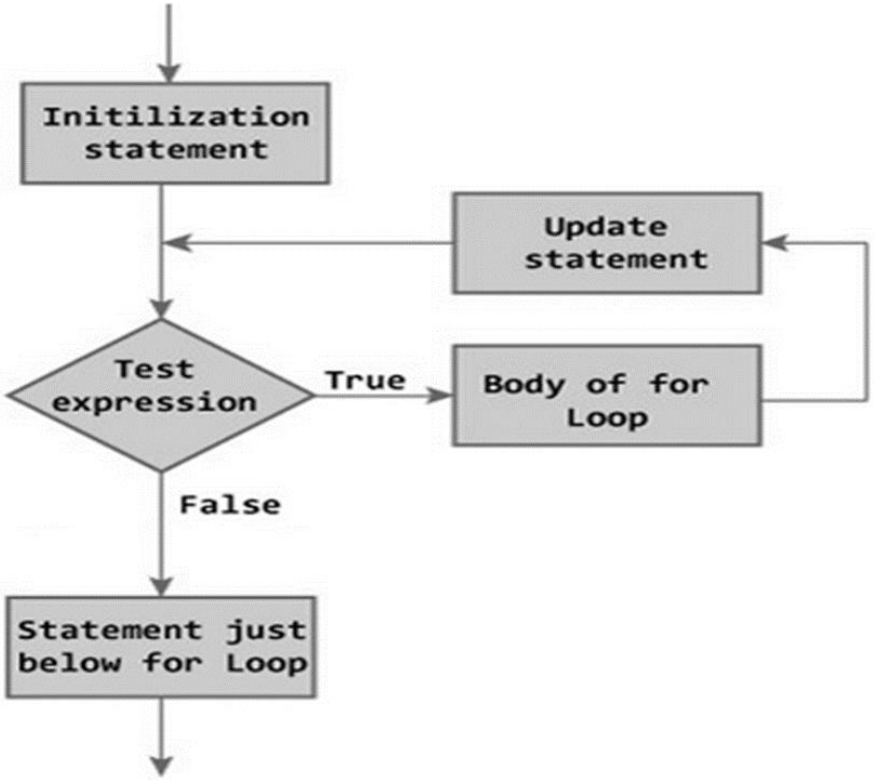
The *for* Statement

- **for** loop is an *entry-controlled loop*
- The general form of the for loop is

```
for(initialization; test-condition; increment/decrement){
        body of the loop (you can write your code here)
}
```

- Initialization is done first using assignment statements
- Value of the loop control variable is tested using test condition & Test condition is a relational expression
- When the body loop is executed, the controls is transferred back to ***for*** after the evaluation of last statement in the loop. The value of control variable is updated and new value is checked using test condition. If true, then body of the loop is executed, otherwise loop get terminated & control is transferred to the statement that immediately follows ***for*** loop

The *for* Statement

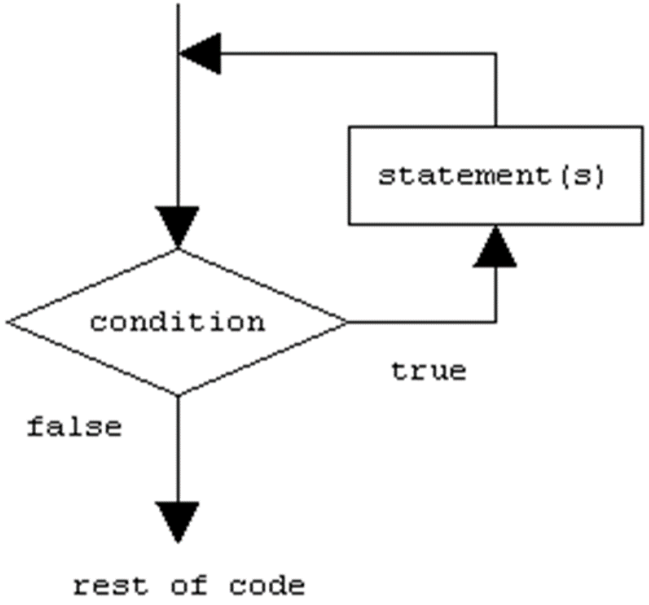


- **Example:** Write a program to find the sum of first **n** natural numbers where n is entered by user.

The *while* Statement

- **while** loop is another *entry-controlled loop*
- The general form of the while loop is

```
initialization;
while( test-condition){
        body of the loop
(you can write your code here
        increment/decrement;
}
statement-x;
```

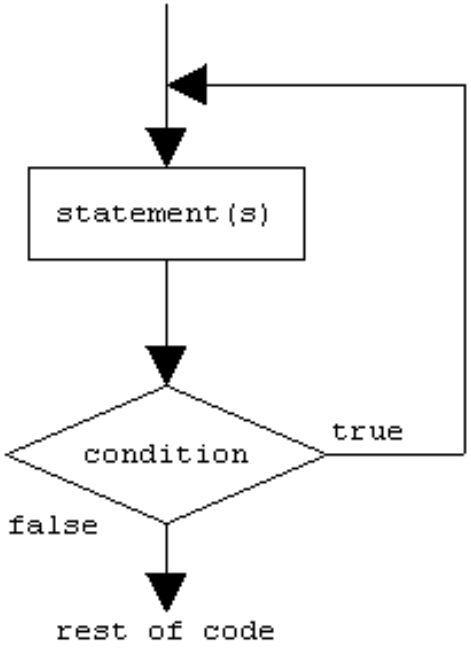


- Example: WAP to compute n!

The *do while* Statement

- The general form of the do-while loop is

```
initialization;
do{
    body of the loop;
    Increment/decrement;
}
while( test-condition);
statement-x;
```



- Do while loop is an exit-controlled loop
- **Example:** WAP to print the multiplication of n number from 1-15.

CLASS ACTIVITY

Question 1:

Write a program to Print the given series using do-while loop.

10 9 8 7 6 5 4 3 2 1 0

CLASS ACTIVITY

Question 2:

Write a program to count all odd numbers and even numbers from 1-25 using for loop.

CLASS ACTIVITY

Question 3:

Write a program to print the multiplication table of user entered number using while loop.

Break Statement

- The break statement ends the loop immediately when it is encountered. Its syntax is:

```
break;
```

- The break statement is almost always used with `if...else` statement inside the loop

Break Statement (cont.…)

```
while (testExpression) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

```
do {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (condition to break) {
        break;
    }
    // codes
}
```

Example: Break Statement

```c
#include <stdio.h>
int main() {
        int i;
        double number, sum = 0.0;
        for (i = 1; i <= 10; ++i) {
                printf("Enter a n%d: ", i);
                scanf("%lf", &number);
                // if the user enters a negative
                //number, break the loop
                if (number < 0.0) {
                        break;
                }
                sum += number; // sum = sum + number;
        }
        printf("Sum = %.2lf", sum);
        return 0;
}
```

Output

```
Enter a n1: 2.4
Enter a n2: 4.5
Enter a n3: 3.4
Enter a n4: -3
Sum = 10.30
```

Continue Statement

- The continue statement skips the current iteration of the loop and continues with the next iteration.

```
continue;
```

- The `continue` statement is almost always used with the `if...else` statement.

Continue Statement (Cont..)

```
while (testExpression) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

```
do {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
} while (testExpression);
```

```
for (init; testExpression; update) {
    // codes
    if (testExpression) {
        continue;
    }
    // codes
}
```

Example: Continue Statement

```
#include <stdio.h>
int main() {
        int i;
        double number, sum = 0.0;
        for (i = 1; i <= 10; ++i) {
                printf("Enter a n%d: ", i);
        scanf("%lf", &number);
                if (number < 0.0) {
                        continue;
                }
                sum += number; // sum = sum + number;
        }
        printf("Sum = %.2lf", sum);
        return 0;
}
```

Nested Loop

- Nested loops consist of an **outer loop** with one or more **inner loops**
- The loops should be properly indented
  - Enables the reader to easily determine which statements are contained within each *for* statement
- ANSI C allows up to 15 levels of nesting

Nested Loop

Example:

```
for(i=1;i<=10;i++)
{
    for(j=1;j<=20;j++)
    {


    }
}
```

Outer loop

Inner loop

The above loop will run for 10*20 iterations

Class Activity 1

- Write a Program to print the following using nested for loop.

```
1 1 1 1 1 1
2 2 2 2 2 2
3 3 3 3 3 3
4 4 4 4 4 4
5 5 5 5 5 5
6 6 6 6 6 6
7 7 7 7 7 7
```

Class Activity 2

Write a Program to print the following using nested for loop.

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

Home Assignment

- Write a program for all the home assignment questions of unit II (part 02).

Thank you

Unit IV – Part 01 (Function)

Lecture Slide

AS2023

Objectives

By the end of this session, students will be able to:

- Define function
- Differentiate between predefined and user defined function
- Explain the need for function
- Identify the characteristics of function
- Identify and explain elements of function
- Solve problem using user defined functions
- Use appropriate return type
- Explain and implement user defined functions

Function

- Block of statements to perform a specific task
- A function is known with various names like a method or a sub-routine or a procedure, modules, subprograms and so on.
- As always, a function is a module of code that takes information in (parameters), does some computation, and (usually) returns a new piece of information based on the parameter information.
- Can be classified either as library functions/inbuilt functions or user defined function
- **add( )** is a example of user-defined functions and **scanf( ) & printf( )** are examples of *library function*

Need for function

- Subprograms are Easier to understand, debug and test
- Facilitates TOP-DOWN modular programming
- The length of the source program can be reduced
- Easier to locate and isolate faulty function for further investigation
- A function may be used by many other programs

Multi-functioned Program

- Once a function has been designed and packed, it can be treated as a '***black box***'
- Consider a set of statements as shown below

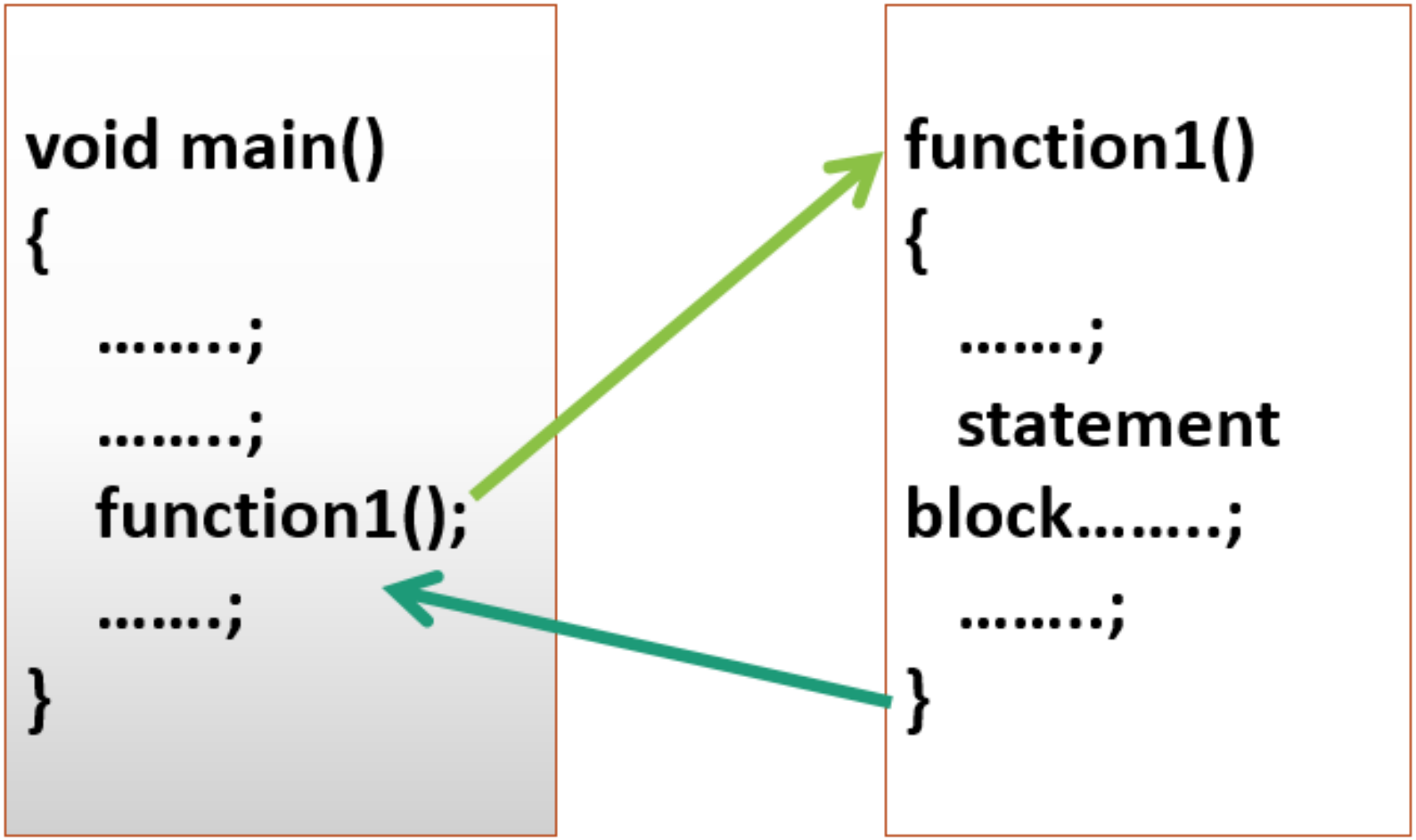
```
void printline(void){
   int i;
   for(i=1; i<40; i++){
      printf("_");
   }
}
```

Multi-functioned Program

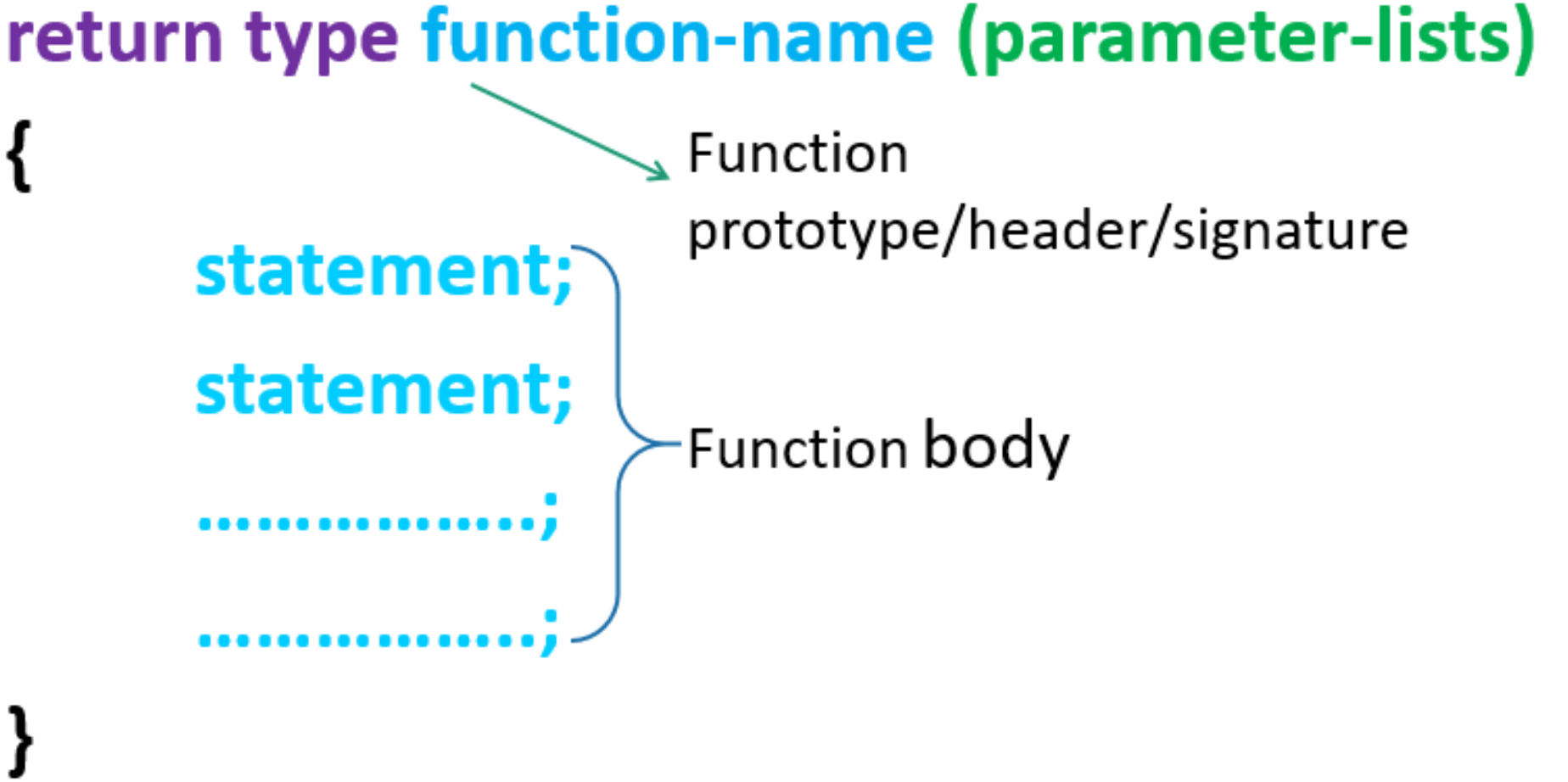
- The function can be used in the program as

```
void printline(void); /*declaration*/
int main( ){
        printline( );
        printf("This illustrates the use of C function\n");
        printline( );
}
void printline(void){
        int i;
        for(i=1; i<40; i++){
                printf("_");
        }
}
```

Function



Basic Structure of Function



Characteristics

- Each function should do only one thing
- Communication between functions is allowed only by a calling function
- A function can be called by one and only one higher module
- No communication can take place directly between modules that do not have *calling-called relationship*
- All modules are designed as *single-entry*, *single-exit* systems using control structures

Elements of Function

- Functions are classified as one of the derived data types in C
- We can define function and use them like a variables in C
- It holds some similarities with variable
  - Both are identifiers and should adhere to the rules for identifiers
  - Both should have type associated with them
  - Both should be declared and defined before their usage in the program
- In order to establish a function, three elements are required that are related to functions
  - Function definition
  - Function call
  - Function declaration

Function Definition

- Is an independent program module that is especially written to implement the requirements of the function
- It is also know as ***function implementation***
- Shall include the following elements

1. Function type
2. Function name
3. List of parameters

Function header

4. Local variable declaration
5. Function statements
6. A return statement

Function body

Function Definition

- General format of function definition

function_type *function_name(parameter list) //Function header*

{

local variable declaration;

executable_statement;

……..

……..

return statement;

}

function body

Function Header

- Consist of three parts as discussed in previous slide
- Type & Name
  - Function types specifies the type of value to be or that function is expected to return to the program calling the function
  - if the return type is not explicitly specified, C will assume that it is an integer type
  - if the function is not returning anything, then we need to specify the return type as **void**
  - it is good programming practice to code return type explicitly even when it is integer type
  - Value return is output produced by the function
  - Function name can be any valid C identifiers
  - Name should be appropriate to the task performed by the function

Function Header

- 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
  - Since they represent the input values, they are often referred to as **formal parameters**
- Examples

```
1. float quadratic(int a, int b){
          function body;
   }
2. double power(double a, int b) {
           function body;
   }
3. int sum(int a, int b){
          function body;
   }
```

Remember:

1.there is no semicolon after the closing parenthesis

2. *every parameters is separated with coma*
3. *declaration of parameters cannot be combined. Example* ***int sum(int a,b)*** *is* ***illegal***

Function Declarations

- Like variables, all functions in C program must be declared before they are invoked
- Function declaration is known as function prototype and takes the following form

```
function_type function_name (parameter list);
```

- If function doesn't take any values and doesn't return any values, its prototype is written as:

```
void display (void);
```

- A function declaration can be placed in two places in a program
  - Write the functions above main function
  - Write the functions prototype before main function

Function body

- Contains the statement block (declarations and executable statements) necessary for performing the required task
- Contains three parts
  - Local variable declaration
  - Functions statement
  - Return statements
- If the function doesn't return any values, we can omit return statements but note to declare function type or return type as **void**
- when the function reaches its return statement, the control is transferred back to the calling program. In absence, the closing brace act as a **void return**

Function body cont..

- Return values and their types
  - Function may or may not send back any values to the calling function
  - If it does, it is done through return statement
  - The function can only return one value per call at most
  - The return statement takes the following form

```
return;
Or
return(expression);
```

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

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

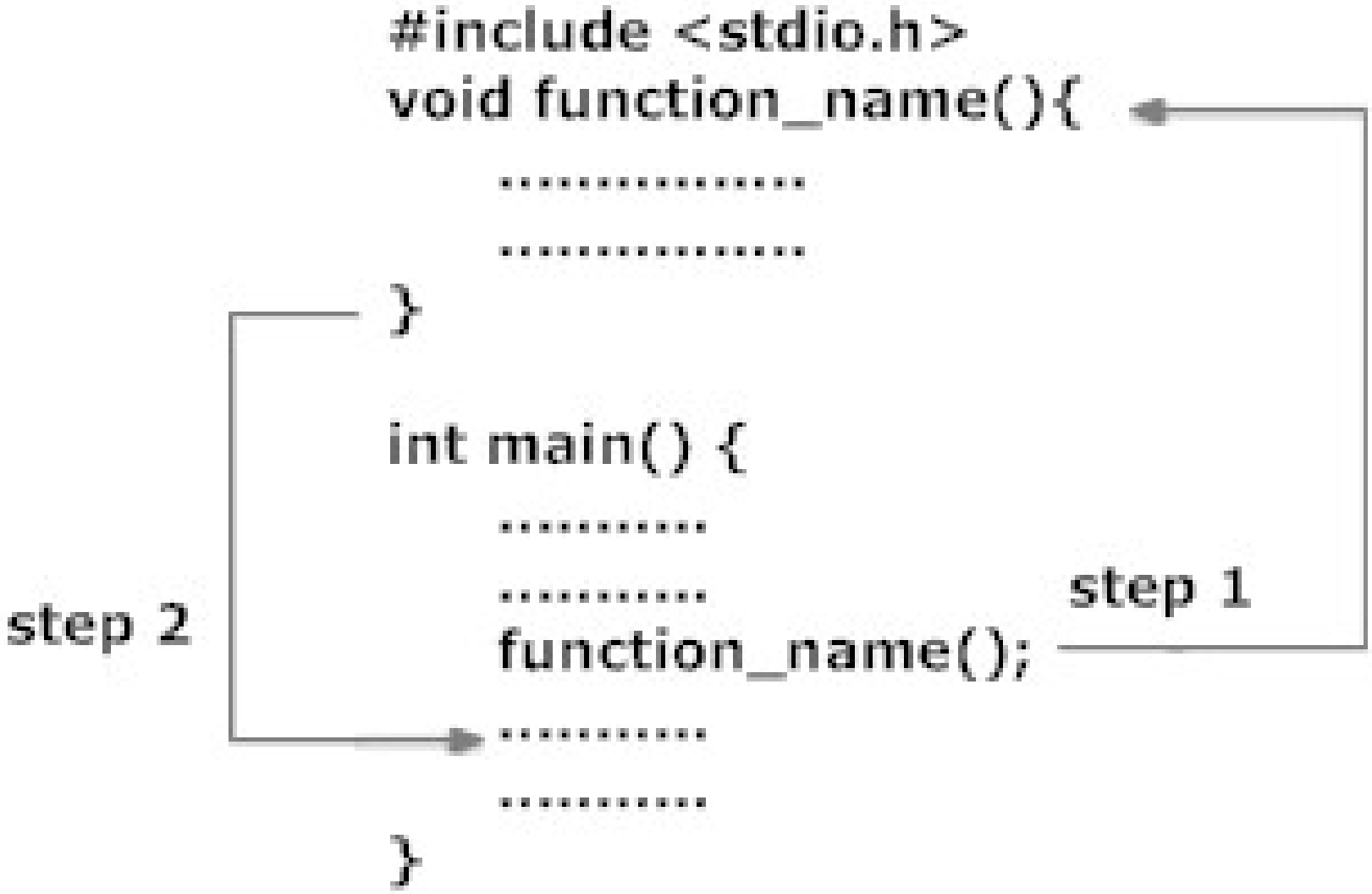
Function call

- A function can be called by simply using the function name followed by list of actual parameters (arguments), if any, in the parenthesis

```
int main( ){
    int y;
    y=mul(10,5); /*function call*/
    printf("%d\n",y);
}
```

- A function which returns a value can be used in expression like any other variable.
- However, a function cannot be used on the right side of an assignment statement
- The actual parameter should match with the formal parameter in type, order and number

How user-defined function work?



Demonstration

- WAP program that uses function to compare two user input number.

Thank you