COMPUTER SCIENCE
12

(MS Access and C)

CHAPTER 13: Functions in C

Topics

- Function
- Types of Functions
- Function Declaration
- Function Definition
- Difference between Function declaration and Definition
- Function Call
- Passing Parameters to Functions
- Difference between Formal and Actual Parameters
- Returning Value from Function
- Local Variable
- Global Variable
- Difference between Local and Global Variable

Topics(continued)

- Functions without Arguments
- Functions that Return a Value and Accept Arguments

Functions

- A function is named block of code that performs a specific task or action
- Functions are like building blocks
- A program may contain many functions
- Each function has a unique name
- The statements written in a function are executed when it is called by its name
- Functions provide structured programming approach
  - Breaking a large program into smaller pieces or modules
  - Modules in C are called **functions**
  - Pieces are more manageable than one large program
  - Pieces can be independently implemented and tested

Benefits of using Functions

- Easier to Code
- Easier to Modify
- Easier to Maintain & Debug
- Reusability
- Less Programming Time

Types of Functions

User-defined Function

- Function written by the programmer
- User-defined function has a unique name
- A program may contain many user-defined functions
- These functions are written according to the exact need of the user

Built-in or Predefined Functions

- Available as a part of language.
- Also known as **Library functions**.
- These functions are ready-made programs.
  - Make programming faster and easier.
- These functions are defined in different header files.
  - **printf()** and **scanf()** functions are defined in **stdio.h**
  - **getch()**, **getche()**, and **clrscr()** functions are defined in in **conio.h**
  - The **sqrt()** function is defined in the **math.h** header file.
- C provides many built-in functions to perform mathematical calculations, input /output and many other useful operations**.**
  - **printf()** is used to send formatted output to the screen (display output on the screen).
  - The **sqrt()** function calculates the square root of a number.

Function Declaration or Function Prototype

- **Function declaration or Function prototype** is a model of a function.
- It provides information to the compiler about the structure of the function to be used in program.
- The compiler must know the following about a function before it is called:
  - **Name** - the name of the function. Function names follow same rules as variable names
  - **Return type** – data type of the value the function returns to the part of the program that called it
  - **Parameters**- Values that are provided to a function when the function is called.
- Function prototype usually placed at the beginning of source file just before the main() function.

Syntax

Return-type Function-name (parameters);

The parameters are given in two ways:

- Only data types of the parameters are written in prototype as follows:

int add(int, int);

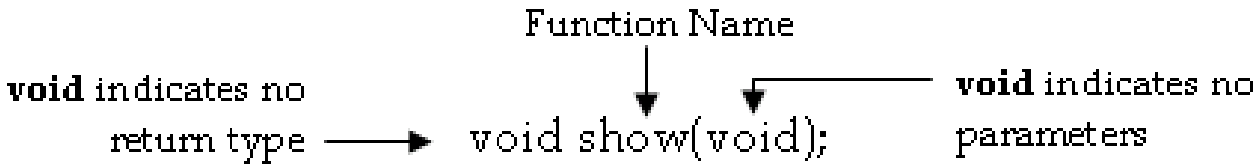
- Both data types and names of parameters are written in prototype as follows:

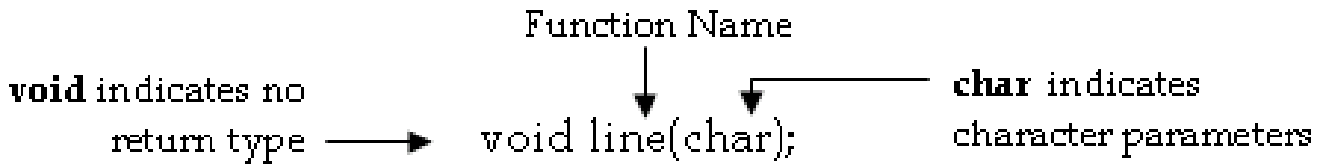
int add(int a, int b);

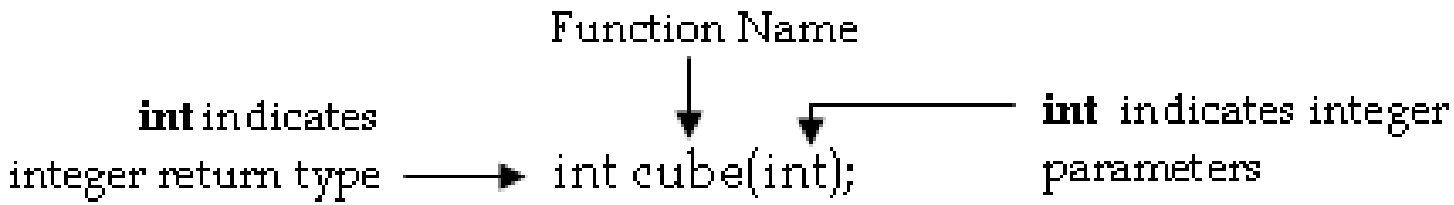
Remember!
Forgetting the semicolon at the end of function prototype is a syntax error

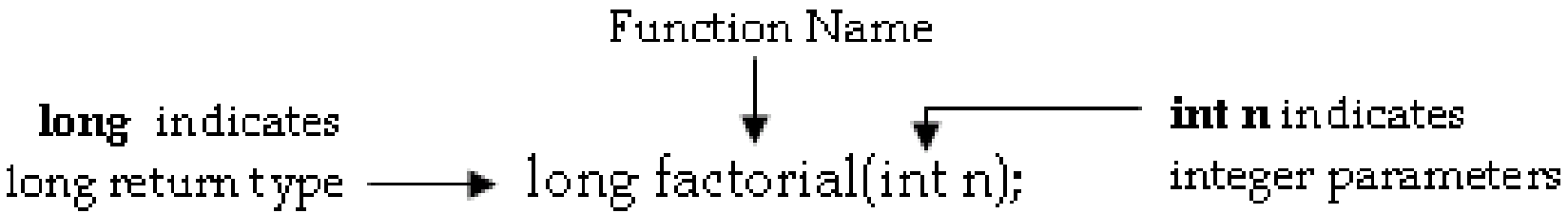
Remember!
Specifying a function parameters of the same type as double a, b instead of double a, double b results in a compilation error

Examples









Function definition

A set of statements that explains what a function does is called **function definition**.

The function definition can be written at the following places:

- Before main() function
- After main() function
- In a separate file

The function definition consists of two parts:

Remember!
Placing a semicolon after right parenthesis enclosing the parameter list of function definition is a syntax error

1. Function Header

- The first line of function definition is known as function header.
- It is similar to function prototype
- The only difference is that it is not terminated with semicolon.
- The number of parameters and sequence parameters in function header and function prototype must be same.

Syntax

```
Return-type Function-name (parameters)      <-- Function header
{
  statement 1;
  statement 2;
        :
        :                                   <-- Function body
  statement N;
}
```

2. Function Body

- The set of statements which are executed inside the function is known as function body.
- The body of function appears after function declaration and the statements are written in curly braces { }

Difference between Function Definition & Declaration

Function Definition	Function Declaration
1. It consists of different statements to perform a particular task.	1. It consists of single statement that informs the compiler about a function.
2. It cannot be written in the body of main() function.	2. It can be written in the body of main() function.
3. It must be written before or after the main() function.	3. It is not required if function definition appears before the main() function.

Function Call

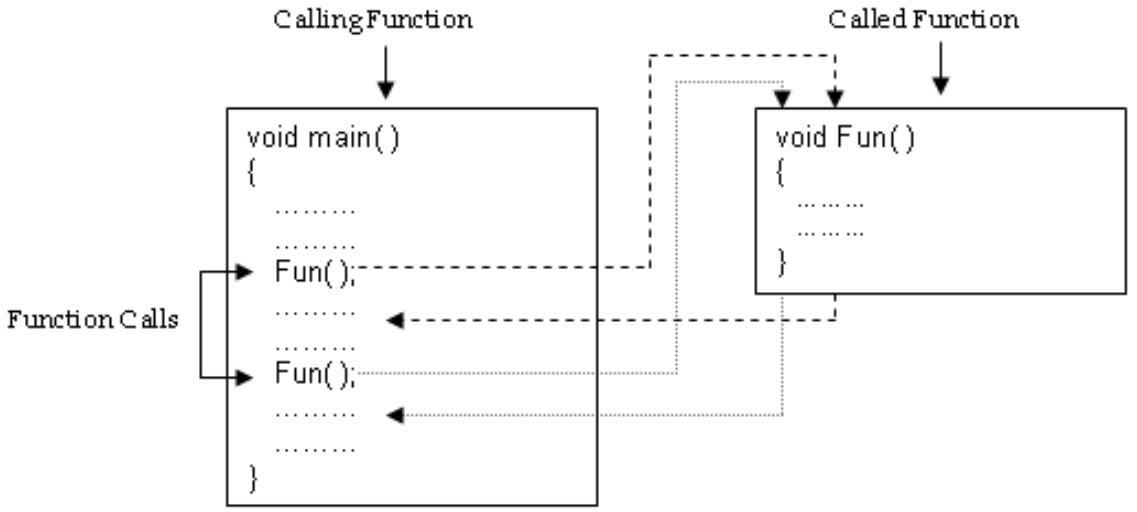
- The statement that activates a function is known as **function call**.
- To call a function, use the function name followed by () and ;

```
fun();
```

- The following steps take place when a function is called:
  - The control moves to the function that is called
  - All statements in the function body are executed
  - The control returns back to the calling function
- When the control returns back to the calling function,

the remaining statements in the calling function are executed

- main is automatically called when the program starts
- main can call any number of functions
- Functions can call other function



Program 13.1

Write a program that displays a message "Programming makes life interesting" on screen using function.

```
#include <stdio.h>
void show()
{
   printf("Programming makes life interesting.");
}

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

Output:
Programming makes life interesting.

Program 13.2

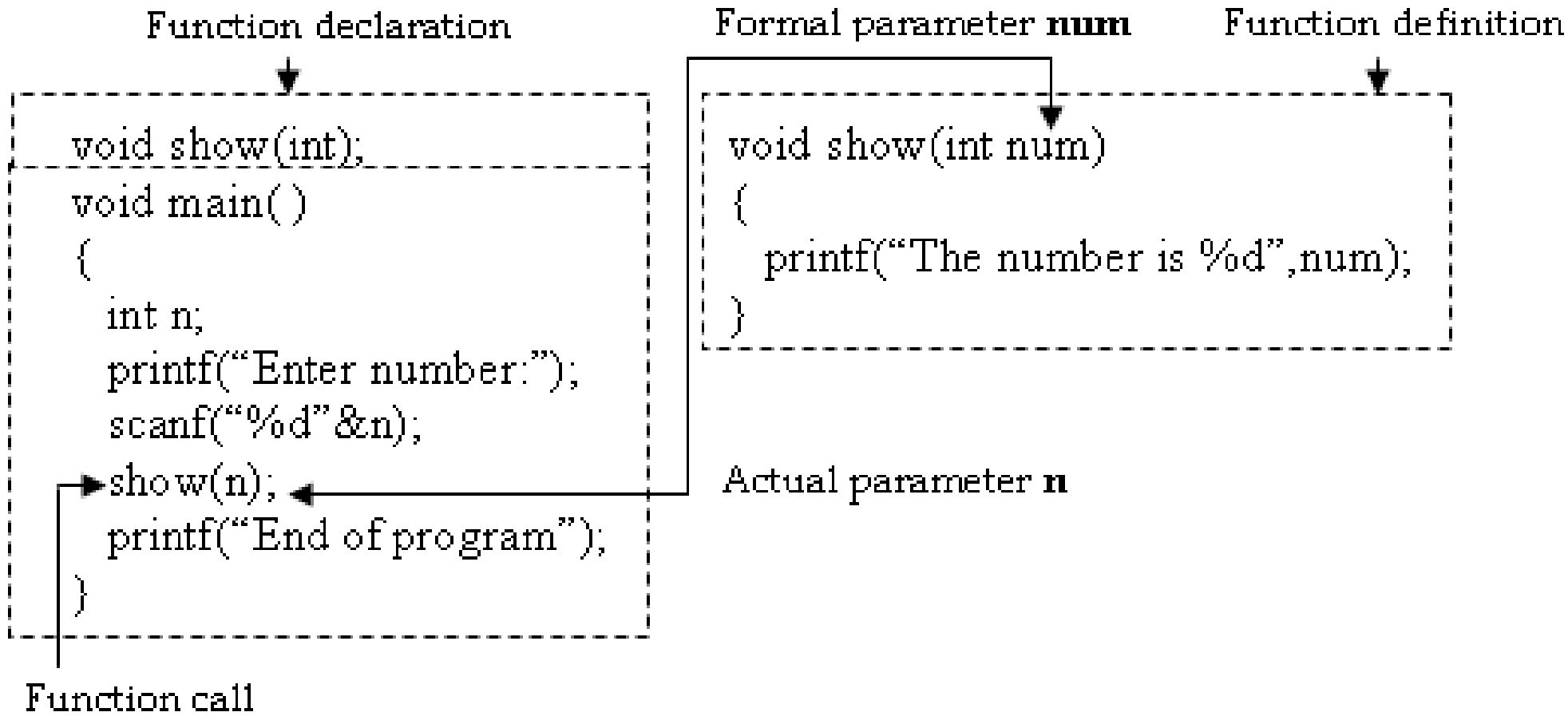
Write a program that display "Pakistan" ten times using function.

```
#include <stdio.h>
void show()
{
   printf("Pakistan \n");
}
void main()
{
   int i;
   for(i=1; i<=10; i++)
      show();
}
```

Passing Parameters to Functions

- Parameters are the values that are provided to a function when the function is called.
- Parameters are given in the parentheses
- If there are many parameters, these are separated by commas. If there is no parameter, empty parentheses are used
- The sequence and types of parameters in **function call** must be similar to the sequence and types of parameters in function declaration
    - Parameters in function call are called **actual parameters**
    - Parameters in function declaration are called **formal parameters**
- The number of actually parameter must be same as number of formal parameters.
- The values of actual parameters are copied to formal parameters when a function call is executed

Example



Program 13.3

Write a program that inputs two numbers in main function, passes these numbers to a function. The function displays the maximum number.

```
#include <stdio.h>
#include <conio.h>
void max(int a, int b);
void main()
{
  int x, y;
  printf("Enter two numbers: ");
  scanf("%d %d",&x, &y);
  max(x,y);
  getch();
}
void max(int a, int b)
{
  if(a > b)
    printf("Maximum number is %d",a);
  else
    printf("Maximum number is %d",b);
}
```

Output:

```
Enter two numbers: 20 30
Maximum number is 30
```

Program 13.4

Write a program that inputs a number in main function and passes the number to a function. The function displays table of that number.

```
#include <stdio.h>
#include <conio.h>
void table(int n);
void main()
{
  int num;
  clrscr();
  printf("Enter a number: ");
  scanf("%d",&num);
  table(num);
  getch();
}

void table(int n)
{
  int c;
  for(c=1; c<=10; c++)
  {
    printf("%d * %d = %d \n",n,c,n*c);
  }
}
```

Output:

```
Enter a number: 3
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
```

Difference between Formal & Actual Parameters

Formal Parameters	Actual Parameters
1. They are used in function header.	1. They are used in the function call.
2. They are used to receive values that are passed to the function in function call.	2. They are the actual values that are passed to function definition in function call.
3. They similar to local variables of the function in which they are used.	3. They may be constant values or variable names.

Function Return Type

- A function can return a single value.
- The return type in function declaration indicates the type of value returned by a function

  e.g. int is used as return type if the function returns integer value
- If the function returns no value, the keyword **void** is used as return type
- The keyword **return** is used to return the value back to the calling function
- When the **return** statement is executed in a function, the controls moves back to the calling function along with the returned value

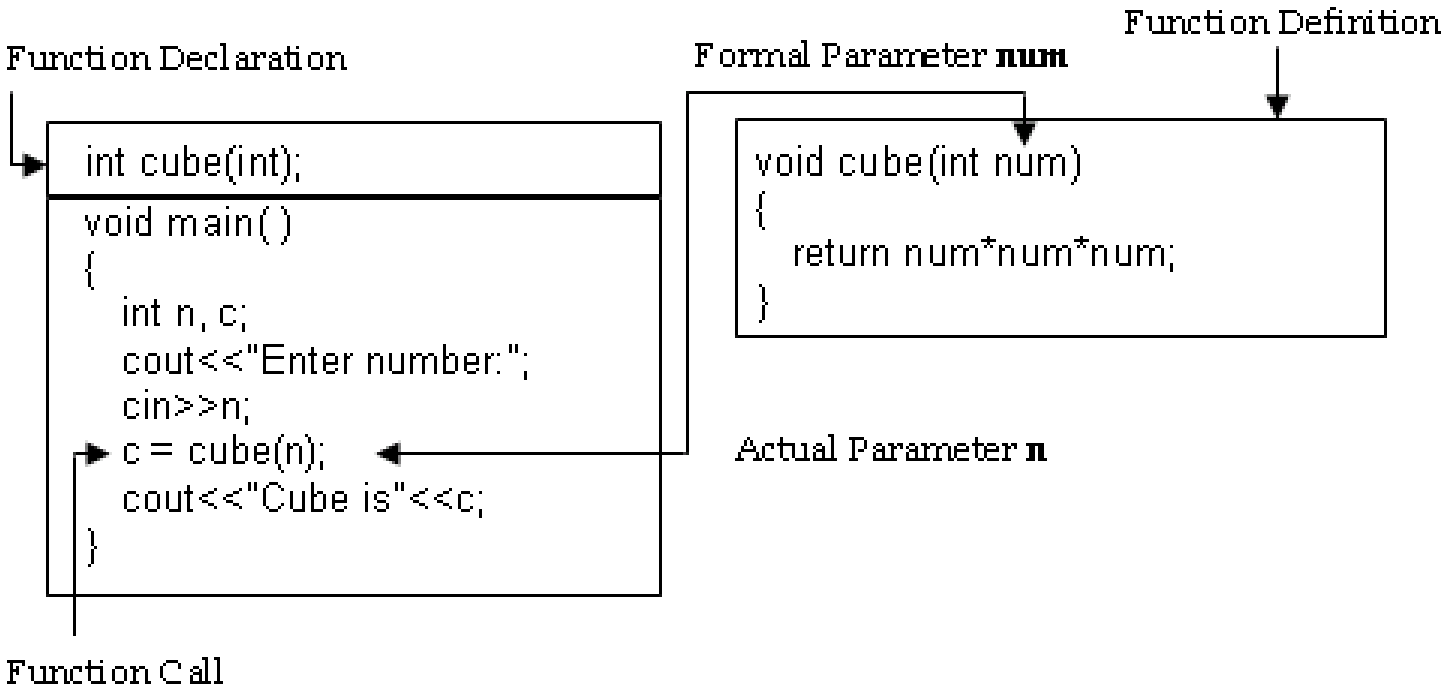
Syntax

The syntax for returning a value is as follows:

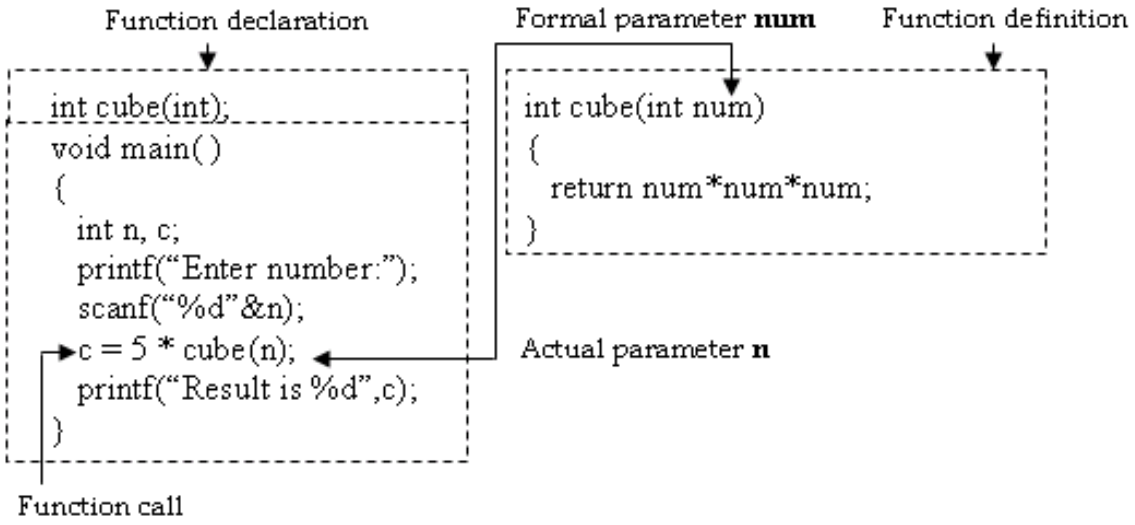
return expression;

- The calling function can use the returned value in the following ways:
  - Assignment statement
  - Arithmetic expression
  - Output statement

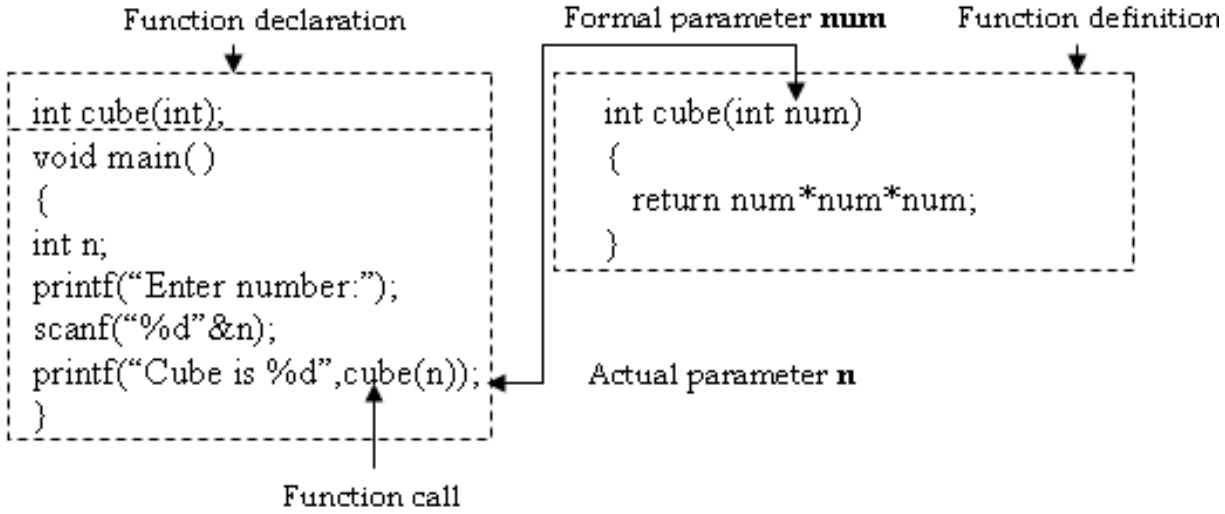
1. Assignment Statement



2. Arithmetic Expression



3. Output Statement



Program 13.3

Write a program that inputs two numbers in main function, passes these numbers to a function. The function displays the maximum number.

```
#include <stdio.h>
#include <conio.h>
void max(int a, int b);
void main()
{
  int x, y;
  printf("Enter two numbers: ");
  scanf("%d %d",&x, &y);
  max(x,y);
  getch();
}
void max(int a, int b)
{
  if(a > b)
    printf("Maximum number is %d",a);
  else
    printf("Maximum number is %d",b);
}
```

Output:

```
Enter two numbers: 20 30
Maximum number is 30
```

Program 13.4

Write a program that inputs a number in main function and passes the number to a function. The function displays table of that number.

```
#include <stdio.h>
#include <conio.h>
void table(int n);
void main()
{
  int num;
  clrscr();
  printf("Enter a number: ");
  scanf("%d",&num);
  table(num);
  getch();
}

void table(int n)
{
  int c;
  for(c=1; c<=10; c++)
  {
    printf("%d * %d = %d \n",n,c,n*c);
  }
}
```

Output:

```
Enter a number: 3
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
3 * 10 = 30
```

Local Variable

- A variable declared inside a function
- Local variables are also called **automatic variables**

Syntax

auto data_type identifier;

auto	It is a keyword that indicates that the variable is automatic. The use of **auto** is optional.
data_type	It indicates the data type of variable
identifier	It indicates the name of variable

Scope of Local Variable

- The area where a variable can be accessed is known as **scope of variable**
- Local variable can be used only in the function in which it is declared
- If a statement accesses a local variable that is not in scope, the compiler generates a syntax error

Local Variable (cont.)

Lifetime of Local Variable

- The time period for which a variable exists in the memory
- The lifetime of local variable starts when the control enters the function in which it is declared
- Local variable is automatically destroyed when the control exits from the function and its lifetime ends
- When the lifetime of a local variable ends, the value stored in this variable also becomes inaccessible

Global Variable

- A variable declared outside any function
- Global variables can be used by all functions in the program
- The values of these variables are shared among different functions
- If one function changes the value of a global variable, this change is also available to other functions

Scope of Global Variable

- Global variable can be used by all functions in the program
- It means that these variables are globally accessed from any part of the program
- Normally, global variables are declared before main function

Lifetime of Global Variable

- Global variables exist in the memory as long as the program is running
- These variables are destroyed from the memory when the program terminates
- These variables occupy memory longer than local variables

Program 13.12

Write a program that inputs a number in global variable. It calls a function that multiplies the global variable by 2. The main function then displays value of global variable.

```
#include <stdio.h>
int g;
void fun();
void main()
{
   printf("Enter a number: ");
   scanf("%d",&g);
   printf("Value of g before function call: %d\n",g);
   fun();
   printf("Value of g after function call: %d",g);
}
void fun()
{
   g = g * 2;
}
```

Output:

```
Enter a number: 5
Value of g before function call: 5
Value of g after function call: 10
```

Difference between Local and Global Variable

Local Variable	Global Variable
1. Local variables is declared within a function.	1. Global variable is declared outside any function.
2. It can be used only in the function in which they are declared.	2. It can be used in all functions.
3. It is created when the control enters the function in which it is declared.	3. It is created when the program starts execution.
4. It is destroyed when the control leaves the function.	4. It is destroyed when the program terminates.
5. It is used when the values are to be used within a function.	5. It is used when values are to be shared among different functions.

Functions without Arguments

- The simplest type of function is a function that accepts no argument and returns no value
- The return type of such function is **void**
- The parameter list may be empty or the keyword void can be written in the parenthesis
- The keyword void instead of parameter list indicates that the function accepts no argument when it is executed

Example

```
#include <stdio.h>
#include <conio.h>
void fun();
void main()
{
  fun();
  getch();
}

void fun()
{
  printf("I accept no argument and return no value.");
}
```

Output:
I accept no argument and return no value.

Functions that Return Value and Accepts Arguments

- A function may require certain values to calculate the result
- The required values are passed to the function as arguments
- The arguments are written in parenthesis
- Each argument is separated by semicolon
- The function can process the arguments and return the result back to main function.
- A function can only return single value.
- The keyword **return** is used to return a value.

Example

The following example inputs two numbers, passes the numbers to a function Add(). The function adds the numbers and returns to main() function. The result is displayed on the screen in main function.

```
#include <stdio.h>
#include <conio.h>
int Add(int x, int y);
void main()
{
  int a, b, s;
  printf("Enter two numbers: ");
  scanf("%d %d", &a, &b);
  s = Add(a, b);
  printf("%d + %d = %d", a, b, s);
  getch();
}

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