Methods

- Method Creation with single parameter
- Method Calling
- Method Creation with Multiple Parameters
- Passing Arrays into Methods

1

Function = Method = Subroutine = Procedure = Module

2

Parameter = Argument

Function

- A function is a group of statements that together perform a task. Every C program has at least one function, which is **main()**, and all the most trivial programs can define additional functions.
- A function **declaration** tells the compiler about a function's name, return type, and parameters. A function **definition** provides the actual body of the function.

Method

- A method is a program module that contains a series of statements that carry out a task, you can invoke or call a method from another program or method.
- Any program can contain an unlimited number of methods and each can be called an unlimited number of times.
- The rules for naming modules are different in every programming language, but they often are similar to the language's rules for variable names. In this text, module names are followed by a set of parentheses.

Method (Cont.)

- A method must include a header (also called the declaration or definition), a body, and a return statement.
- Within a program, the simplest methods you can invoke don’t require any data items (called arguments) to be sent to them, nor do they send any data back to you (called retuning a value).
- Variables and constants are in scope within or local to only the method in which they are declared.

Steps

- Function declaration
- Function definition
- Function call

Syntax – Defining Function

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

Function definition = Function header = Function body

- **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 value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.
- **Function Body** − The function body contains a collection of statements that define what the function does.

Syntax – Function Declaration

- A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- return_type function_name(parameter list);

Calling a Function

- While creating a C function, you give a definition of what the function has to do. To use a function, you will have to call that function to perform the defined task.
- When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.
- To call a function, you simply need to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

There are different types of function calling. Depending on the number of parameters it can accept , function can be classified into following 4 types –

Function Type	**Parameter**	**Return Value**
Type 1	Accepting Parameter	Returning Value
Type 2	Accepting Parameter	Not Returning Value
Type 3	Not Accepting Parameter	Returning Value
Type 4	Not Accepting Parameter	Not Returning Value

Method Creation with **no** parameter

```
#include<stdio.h>
void area();
void main()
{  area(); }
void area()
{
float area_circle;
float rad;
printf("\nEnter the radius : ");
scanf("%f",&rad);
area_circle = 3.14 * rad * rad ;
printf("Area of Circle = %f",area_circle);
}
```

Method Creation with single parameter

```
#include<stdio.h>
void area(float rad);
void main()
{
float rad;
printf(“\nEnter the radius : ");
scanf("%f",&rad);
area(rad);
}
void area(float rad)
{
float ar;
ar = 3.14 * rad * rad ;
printf("Area of Circle = %f",ar);
}
```

Method Creation with single parameter

```
#include<stdio.h>
float calculate_area(int);
int main()
{
int radius;
float area;
printf("\nEnter the radius of the circle : ");
scanf("%d",&radius);
area = calculate_area(radius);
printf("\nArea of Circle : %f ",area);
return(0);
}
float calculate_area(int radius)
{
float areaOfCircle;
areaOfCircle = 3.14 * radius * radius;
return(areaOfCircle);
}
```

Method Creation with Multiple Parameters

```
#include<stdio.h>
void funct(int x,int y,int z)
{
printf("%d%d%d",x,y,z);
}
void main()
{
int var=15;
funct(var,var++,++var);
}
```

Passing Arrays into Methods

```
#include<stdio.h>
void fun(int arr[])
{
     int i;
     for(i=0;i< 5;i++)
     arr[i] = arr[i] + 10;
}
void main()
{
     int arr[5],i;
     printf("\nEnter the array elements : ");
     for(i=0;i< 5;i++)
               scanf("%d",&arr[i]);
               printf("\nPassing entire array .....");
               fun(arr); // Pass only name of array
     for(i=0;i< 5;i++)
               printf("\nAfter Function call a[%d] : %d",i,arr[i]);
}
```

Recursion Function

- Recursion is the process of repeating items in a self-similar way.
- If a program allows you to call a function inside the same function, then it is called a recursive call of the function.

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

Recursion Function (cont.)

- The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.
- Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

Fibonacci using Recursion Function

```
#include <stdio.h>
int fibonacci(int i)
{
    if(i == 0)
    { return 0; }
    if(i == 1)
    { return 1; }
    return fibonacci(i-1) + fibonacci(i-2);
}
int main()
{
    int i;
    for (i = 0; i < 10; i++)
    {
          printf("%d\t\n", fibonacci(i));
    }
return 0;
}
```

- **Pass by value**
- Function in C passes all arguments 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.

- **Pass by reference**
- When passing by reference technique is used, the address of the data item is passed to the called function.
- Using & operator we can determine the address of the data item.
- Note that function once receives a data item by reference, it acts on data item and the changes made to the data item also reflects on the calling function.
- Here you don't need to return anything to calling function.

More about Function on:

http://www.c4learn.com/c-programming/c-function-calling-types/