**Functions& Parameters**

<http://www.tutorialspoint.com/cprogramming/c_pointers.htm>

{funcMes.c funcSumm.c}

**Definition.**

***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.

You can divide up your code into separate functions. How you divide up your code among different functions is up to you(not artificially), but logically the division is such that each function performs a specific task.

**Function definition in C programming consists of a *function header* and a *function body*.**

Here are all the parts of a 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 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.

**Call by value** .This method *copies the actual value of an argument into the formal parameter* of the function. In this case, changes made to the parameter inside the function have no effect on the argument.

**Call by reference**. This method *copies the address of an argument into the formal parameter*. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

{ epoint1.cpp funcVan.c }

* **Function Body** − The function body contains a collection of statements that define what the function does.

There are **three places where variables can be declared** in C programming language −

* **Inside a function** or a block which is called **local** variables.
* **Outside of all functions** which is called **global** variables.
* **In the definition** of function parameters which are called **formal** parameters.

{ point61.cpp without--🡪 Ltel.cpp }

Some C programming **tasks are performed more easily with pointers**, and other tasks, such as dynamic memory allocation, cannot be performed without using pointers. So it becomes necessary to learn pointers to become a perfect C programmer.

As you know, every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory.

#include <stdio.h> //standard input output

#include <stdlib.h> //standart library

void swap1( int a, int b )

{

 printf("a = %d b = %d\n",a,b);

 int abi = a; a = b; b = abi;

 printf("a = %d b = %d\n",a,b);

}

void swap2( int \* p, int \* q )

{

 printf("Addresses: p = %d q = %d\n", p, q);

 printf("Initial values: \*p = %d \*q = %d\n",\*p,\*q);

 int abi = \*p; \*p = \*q; \*q = abi;

 printf("Final values: \*p = %d \*q = %d\n",\*p,\*q);

}

void swap3( int & a, int & b ) // by reference

{

 int abi = a; a = b; b = abi;

}

int main()

{

 int m = 123, n = -17;

 swap1( m, n );

 printf("swap1: m = %d n = %d\n", m, n);

 swap2( &m, &n );

 printf("swap2: m = %d n = %d\n", m, n);

 swap3( m, n );

 printf("swap3: m = %d n = %d\n", m, n);

 system("PAUSE");

 return 0;

}

A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is –

 type \*varName; ---🡪 int \*point1; /\* pointer to an integer \*/

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

1. we define a pointer variable,

**(b)** assign the address of a variable to a pointer,

1. finally access the value at the address available in the pointer variable.

This is done by using unary **operator \*** that returns the value of the variable located at the address specified by its operand.

#include <stdio.h>

int main () {

 int var = 20; /\* actual variable declaration \*/

 int \*ip; /\* pointer variable declaration \*/

 ip = &var; /\* store address of var in pointer variable\*/

 printf("Address of var variable: %x\n", &var );

 /\* address stored in pointer variable \*/

 printf("Address stored in ip variable: %x\n", ip );

 /\* access the value using the pointer \*/

 printf("Value of \*ip variable: %d\n", \*ip );

 return 0;

}

Address of var variable: bffd8b3c

Address stored in ip variable: bffd8b3c

Value of \*ip variable: 20

It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.

