# UNIVERSIDAD AUTONOMA METROPOLITANA

## Programación Avanzada

## Práctica 1:

## "Entorno Linux, Compilador gcc, Funciones con el paso de parámetros, Variables Locales y Globales"

**Fecha de Elaboración:**
Martes, 24 de junio de 2008

**Fecha de Entrega:**
Martes, 1 de julio de 2008

# Contenido

## AVISOS:

- No se reciben reportes de las prácticas después de la fecha de entrega.
- En la página se encuentra el formato del reporte para las prácticas en donde se describen cada uno de los apartados que deberá considerarse. Se deberá seguir dicha plantilla para el desarrollo del reporte de las prácticas.
- Enviar a la cuenta de correo un paquete zipeado que contendrá:
  - Códigos fuentes (extensión **.c**).
  - Códigos Objetos (extensión **.o** )
  - Ejecutables (sin extensión )
  - En el reporte incluir por actividad (extensión **.doc)**:
    - **Desarrollo** de la actividad.
    - **Resultados** de la actividad.
- En el subject del correo escribir: ***Práctica 1- <nombre completo>***.

# 1 Objetivos

1) Estudiar las características del Sistema Operativo (S. O.) LINUX.
2) Familiarizarse con el S. O. LINUX practicando varios de sus comandos a través de una ventana de "shell".
3) Entender el concepto de compilador y las partes que lo constituyen.
4) Aprender a utilizar **gcc**, el compilador del lenguaje de programación "C" en LINUX.
5) Entender los pasos que se deben realizar para generar un programa ejecutable.
6) Utilizar el diseño de programación ascendente y descendente en el enfoque de **programación modular** (enfoque "divide y vencerás") para la resolución de problemas mediante el uso de **módulos** que serán traducidos a funciones (y/o procedimientos) en el lenguaje C.
7) Definir la **declaración e invocación de funciones** dentro de un programa fuente.
8) Utilizar los mecanismos de **paso de parámetros** en las funciones para la resolución de subproblemas dentro de un programa.
9) Utilizar el mecanismo de **valor de regreso** que se define en el uso de funciones para devolver el resultado de alguna operación que se desarrollo dentro de una función.
10) Determinar como declarar, utilizar y enviar como paso de parámetros, **arreglos** dentro de un programa.
11) Paso de parámetros por **valor** y por **referencia**.

# 2 Introducción

En el desarrollo de esta práctica se requiere que el alumno realice cada una de las actividades que se le indica.

En las actividades referentes al entorno Linux, el alumno ingresara en la línea de comandos algunas instrucciones y observará los resultados generados.

Posteriormente se efectuará una introducción al compilador **gcc**, GNU Compiler Collection (GNU: **G**NU is **N**ot **U**nix), considerando las fase para la creación de programas.

Se desarrollarán algunos programas básicos con la finalidad de que el alumno repase algunos conceptos.

## 2.1 Complementos

Para completar el reporte de la práctica el alumno deberá de efectuar una breve investigación de los siguientes puntos (máximo 3 párrafos por punto):

- El concepto de sistema operativo y las partes que lo constituyen.
- La historia de UNIX y sus características.
- El concepto de traductor: intérprete y compilador de lenguajes de programación de alto nivel.
- Fases de la compilación.
- Fases en la generación de un programa ejecutable.
- El lenguaje de programación “C”: su historia, sus características y sus diferencias con otros lenguajes.

## 2.2 Ayuda complementaria

- Referencia de la bibliografía dentro de las notas del curso de Programación Avanzada.
- Notas del Curso de Introducción a al Programación 08-I en el Capitulo 2, 3 y 4.
- Guía Básica para el uso de C.

# 3 Desarrollo

La sección del desarrollo de la práctica se divide en una serie de actividades que permiten englobar, desarrollar y alcanzar cada uno de los objetivos de la práctica. Es necesario leer con atención cada una de las actividades a desarrollar.

## 3.1 Actividad 1 "Comandos en el Entorno de Linux"

*Con el desarrollo de esta actividad se alcanzará los objetivos, 1 y 2 de la práctica.*

Para el desarrollo de esta actividad siga en orden los siguientes pasos, verifique que los resultados son los mismos que se mencionan a continuación:

1) Entrar al sistema escribiendo el nombre de usuario (login) y contraseña (password).
2) Abrir una ventana de "shell" (Terminal) para practicar varios de los comandos de línea permitidos por el sistema.
3) En la línea de comando es posible tener una ayuda rápida y referencia a los comandos. Practique los siguientes comandos y analice la salida:

   - La mayoría de los comandos tienen un archivo de ayuda que se obtienen tecleando "<comando> -- help", ejemplo:
     - [gustavob@tenampak]$ ls **--help**
       Modo de empleo: ls [OPCIÃN]... [FICHERO]...
     - [gustavob@tenampak]$ mkdir **--help**
       Modo de empleo: mkdir [OPCIÃN] DIRECTORIO..

   - Muchas veces es recomendable utilizar un manual de los comandos donde contiene la información de referencia y el uso de cada uno de los parámetros de dicho comando. Para ver el manual se teclea "man <comando>", ejemplo:
     - [gustavob@tenampak]$ **man** ls
       Dando formato a ls(1); aguarde, por favor...
     - [gustavob@tenampak]$ **man** man
       Dando formato a man(1); aguarde, por favor...

     Puede desplazarse a lo largo del comando y para salir del documento de ayuda presione la tecla "q".

4) Practicar los siguientes comandos y observe los resultados:

   - **ls**

     Lista el contenido de un directorio.

     [gustavob@tenampak]$ **ls**
     Examples hola.txt

     Parámetros:

**-l** se muestra los archivos con la descripción de permisos, propietario, grupo, tamaño, fecha de modificación y nombre del archivo o directorio.

```
{~}[gustavob@tenampak]$ ls -l
total 4
lrwxrwxrwx 1 gustavob profesores 26 2008-06-20 13:11 Examples -> /usr/share/example-content
-rw-r--r-- 1 gustavob profesores 38 2008-06-24 07:54 hola.txt
```

**-a** Muestra todos los archivos.

**-la** muestra todos los archivos con descripción.

- **cal**

  Muestra el calendario, dependiendo del parámetro regresa para una fecha determinada.

  Pruebe los siguientes comandos:

  [gustavob@tenampak]$ **cal**
  [gustavob@tenampak]$ **cal 2008**
  [gustavob@tenampak]$ **cal 09 1980**

- **mkdir**

  Creación de un directorio o carpeta en un directorio con los permisos suficientes para la creación. Si no se tiene dichos permisos no es posible crear archivos o carpetas.

  Pruebe los siguientes comandos

  [gustavob@tenampak]$ **ls -l**
  [gustavob@tenampak]$ **mkdir Practica1**
  [gustavob@tenampak]$ **ls -l**

- **pwd**

  Despliega la ruta actual.

  {~}[gustavob@tenampak]$ **pwd**
  /lab/profesores/gustavob

- **"**Interrumpir proceso: **Crtl + c"**

  Cuando se haya mandado a ejecutar un comando, se puede presionar "Crlt + c" para interrumpirlo. También es útil cuando no se desea ejecutar dicho comando:

  Pruebe lo siguiente:

  {~}[gustavob@tenampak]$ ls -l
  En vez de presionar "enter" presione "Ctrl +c" para cancelar la ejecución del mismo.

- **cd**

  Cambia de directorio.

  - **cd** <directorio> o cd <ruta de directorios>

    Cambia a la ruta o directorio indicado.

    [gustavob@tenampak]$ **pwd**
    /lab/profesores/gustavob
    [gustavob@tenampak]$ **cd Practica1**
    **{~/Practica1}**[gustavob@tenampak]$ **pwd**
    /lab/profesores/gustavob/Practica1

- **cd ..**
  Cambia a la ruta "padre" o un nivel arriba.

```
{~/Practica1}[gustavob@tenampak]$ pwd
        /lab/profesores/gustavob/Practica1

{~/Practica1}[gustavob@tenampak]$ cd ..
{~}[gustavob@tenampak]$ pwd
        /lab/profesores/gustavob
```

- **cd**
  Cambia a la ruta definida para el usuario.

```
{~/Practica1}[gustavob@tenampak]$ pwd
        /lab/profesores/gustavob/Practica1
{~/Practica1}[gustavob@tenampak]$ cd
{~}[gustavob@tenampak]$ pwd
        /lab/profesores/gustavob
```

- **> y >>**
  Redirecciona la salida estandar (ventana de comandos o shell) del resultado de algunos comandos hacia un archivo.
  - <comandos> > <nombre del archivo >
    Guarda el resultado de la ejecución de un comando en el archivo, si ya existe lo sobrescribe.

```
{~}[gustavob@tenampak]$ ls -l > archivo.txt
{~}[gustavob@tenampak]$ ls
        archivo.txt  Examples  hola.txt  Practica1
```

  - <comandos> >> <nombre del archivo >
    Guarda el resultado de la ejecución de un comando al final del archivo, si ya existe coloca el resultado al final del archivo.

```
{~}[gustavob@tenampak]$ cal 2008 >> archivo.txt
```

- **cat**
  Muestra el contenido de un archivo de texto.

  Pruebe los siguientes comandos:

```
{~}[gustavob@tenampak]$ cat archivo.txt

{~}[gustavob@tenampak]$ cal 06 2008 > archivo.txt

{~}[gustavob@tenampak]$ cat archivo.txt
```

- "**Teclas del curso y Tabulador**"
  Las teclas del cursor permiten:
  - Hacia arriba o abajo: pasar entre los comandos previamente escritos.
    Pruebe y presione Ente en el comando que desee ejecutar.
  - Izquierda o derecha: una vez seleccionado el comando permite modificar parte de los mismos.
    Pruebe seleccionado un comando previamente escrito y modificando algún parámetro.

  El tabulador permite completar algunos comandos, nombre de archivos y directorios.
  Pruebe lo siguiente:

{~}[gustavob@tenampak]$ **cd**
< Teclee ls, espacio y luego el tabulador>
{~}[gustavob@tenampak]$ **ls**
Examples/ hola.txt Practica1/
< Teclee ls, espacio, la letra 'P'y presione tabulador>
{~}[gustavob@tenampak]$ **ls P*ractica1/***

- **mv**

Permite mover archivos de un lugar a otro, o renombrarlos.

Pruebe los siguientes comandos:

{~}[gustavob@tenampak]$ **cd**
{~}[gustavob@tenampak]$ **ls**
archivo.txt Examples hola.txt Practica1
{~}[gustavob@tenampak]$ **pwd**
/lab/profesores/gustavob
{~}[gustavob@tenampak]$ **mv archivo.txt Practica1/**
{~}[gustavob@tenampak]$ **ls**
Examples hola.txt Practica1
{~}[gustavob@tenampak]$ **ls Practica1/**
archivo.txt

- Renombrar archivo

Pruebe los siguientes comandos:

{~}[gustavob@tenampak]$ **ls Practica1/**
archivo.txt
{~}[gustavob@tenampak]$ **cd Practica1/**
{~/Practica1}[gustavob@tenampak]$ **mv archivo.txt archivocopia.txt**
{~/Practica1}[gustavob@tenampak]$ **ls**
archivocopia.txt

- Mover al directorio padre

{~/Practica1}[gustavob@tenampak]$ **pwd**
/lab/profesores/gustavob/Practica1
{~/Practica1}[gustavob@tenampak]$**mv archivocopia.txt ..**
{~/Practica1}[gustavob@tenampak]$ **ls**
{~/Practica1}[gustavob@tenampak]$ **ls ..**
archivocopia.txt Examples hola.txt Practica1

- **cp**

Permite copiar o mover un archivo de un lugar a otro, también es posible renombrarlos.

**cp** <ruta/nombre-archivo origen> <ruta/nombre-archivo destino>

- Copiar archivos

Pruebe los siguientes comandos:

{~/Practica1}[gustavob@tenampak]$ **cd ..**
{~}[gustavob@tenampak]$ **ls**
archivocopia.txt Examples hola.txt Practica1
{~}[gustavob@tenampak]$ **cp archivocopia.txt archivo.txt**
{~}[gustavob@tenampak]$ **ls**
archivocopia.txt ***archivo.txt*** Examples hola.txt Practica1

- Copiar archivos en otro directorio

Pruebe los siguientes comandos:

{~}[gustavob@tenampak]$ **ls Practica1/**

{~}[gustavob@tenampak]$ **cp archivo.txt Practica1/**
{~}[gustavob@tenampak]$ **ls Practica1/**
***archivo.txt***
{~}[gustavob@tenampak]$ **ls**
archivocopia.txt archivo.txt Examples hola.txt Practica1

- Copiar archivos en otro directorio cambiando nombre

Pruebe los siguientes comandos:

{~}[gustavob@tenampak]$ **ls Practica1/**
archivo.txt
{~}[gustavob@tenampak]$ **cp archivo.txt Practica1/nuevo.txt**
{~}[gustavob@tenampak]$ **ls Practica1/**
archivo.txt ***nuevo.txt***

- **rm**

Permite borrar archivos o directorios.

- Borrado inmediato

{~}[gustavob@tenampak]$ **ls**
archivocopia.txt archivo.txt Examples hola.txt Practica1
{~}[gustavob@tenampak]$ **rm archivocopia.txt**
{~}[gustavob@tenampak]$ **ls**
archivo.txt Examples hola.txt Practica1

- Borrado con confirmacion

{~}[gustavob@tenampak]$ ls
***archivo.txt*** Examples hola.txt Practica1
{~}[gustavob@tenampak]$ **man rm**
{~}[gustavob@tenampak]$ **rm -i archivo.txt**
rm: Â¿borrar el archivo regular `archivo.txt'? (s/n) **s**

{~}[gustavob@tenampak]$ **ls**
Examples hola.txt Practica1

- Borrar un directorio

{~}[gustavob@tenampak]$ **mkdir Borrador**
{~}[gustavob@tenampak]$ **ls**
***Borrador*** Examples hola.txt Practica1
{~}[gustavob@tenampak]$ **cp Practica1/archivo.txt Borrador/**
{~}[gustavob@tenampak]$ **ls Borrador/**
***archivo.txt***
{~}[gustavob@tenampak]$ ls
Borrador Examples hola.txt Practica1

**<Mucho cuidado con el uso de este comando>**
{~}[gustavob@tenampak]$ **rm -r Borrador/**
{~}[gustavob@tenampak]$ **ls**
Examples hola.txt Practica1

**<Es mejor utilizarlo con confirmación>**
{~}[gustavob@tenampak]$ **rm -ir Practica1/**
rm: Â¿descender al directorio `Practica1/'? (s/n) **n**

{~}[gustavob@tenampak]$ **ls**
Examples hola.txt Practica1

- **who**

Muestra los usuarios conectados al sistema.

{~}[gustavob@tenampak]$ who
gustavob pts/0 2008-06-24 15:17 (148.206.46.26)

- **whoami**

Muestra el usuario actual.

{~}[gustavob@tenampak]$ whoami
gustavob

- **passwd**

Permite cambiar el password. Requiere el password actual.

{~}[gustavob@tenampak]$ passwd
Changing password for gustavob
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:

- **ssh**

Es utilizado para conectarse a otra maquina.

Dentro del laboratorio las computadoras tienen el seudónimo de quetzal#, es por ello que se puede conectarse a otra computadora con el siguiente comando:

{~}[gustavob@tenampak]$ **ssh quetzal17**
The authenticity of host 'quetzal17 (192.168.1.17)' can't be established.
RSA key fingerprint is 2f:82:4c:fe:c2:5a:8f:9f:1e:6e:7a:94:b5:c1:64:73.
Are you sure you want to continue connecting (yes/no)? **yes**
Warning: Permanently added 'quetzal17,192.168.1.17' (RSA) to the list of known hosts.
gustavob@quetzal17's **password**:
{~}[gustavob@quetzal17]$
{~}[gustavob@quetzal17]$ **w**
15:45:23 up 4:24, 3 users, load average: 0.02, 0.04, 0.03
USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
**aze** tty7 :0 15:17 3.00s 19.14s 0.16s x-session-manag
**aze** pts/0 :0.0 15:23 3.00s 0.18s 0.18s bash
**gustavob** pts/1 192.168.1.251 15:42 0.00s 0.16s 0.00s w

## 3.2 Actividad 2 “Entorno de gcc y uso de funciones”

*Con el desarrollo de esta actividad se alcanzará los objetivos, 3, 4, 5, 6, 7, 8 y 9 de la práctica.*

Las fases para la generación de un programa son:

- Edición (codificación)
- Compilación
- Ligado.
- Ejecución.

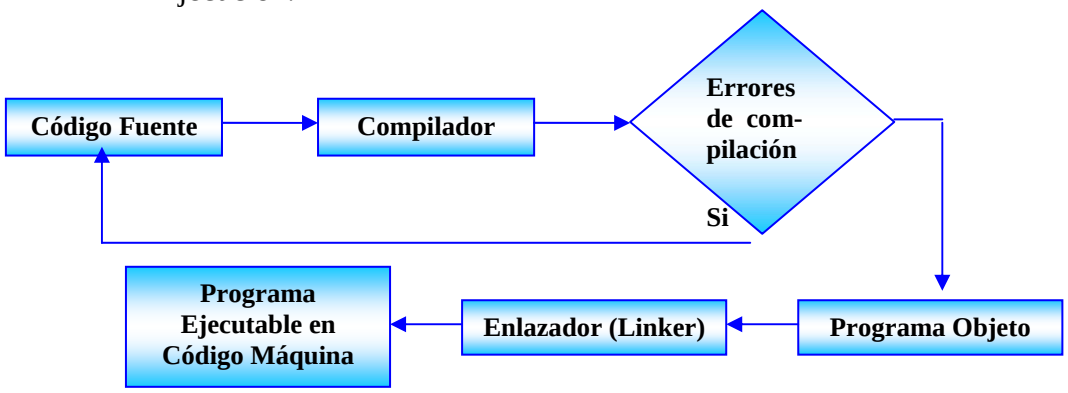



Figura 1.- Fase de un compilador.

Estas fases pueden ir de edición a compilación, repitiéndose varias veces hasta que el programa está libre de errores de sintaxis y pueda ya ligarse. Sin embargo, hay situaciones en donde de la fase de ligado hay que regresar a la de edición para corregir algún nombre de variable o función que estaba equivocado.

Después de la fase de ligado, el programa ya puede ejecutarse y si al realizar pruebas nos damos cuenta que tiene errores semánticos (no hace al 100% lo que debería hacer), regresamos a la fase de edición para corregir y podríamos estar repitiendo las 3 fases y ejecutando hasta que el programa haga lo que especificamos que debe hacer.

### 3.2.1 Compilación

Generar el **código objeto** a partir del código fuente del ejemplo #1, invocando al compilador *gcc*:

**1.** Generar un archivo en código objeto

{~/Practica1}[gustavob@quetzal17]$ **gcc -c ConverTemperatura.c**

En caso de que se encuentren algunos errores compilación se indican por el compilador considerando la línea en donde se encuentra el error o la advertencia.

**2.** Verificar que se haya generado el archivo objeto ***.o**.

{~/Practica1}[gustavob@quetzal17]$ ls
ConverTemperatura.c ConverTemperatura.o

### 3.2.2 Ligado

Invocar al ligado del programa y generar el **programa ejecutable**:

**1.** Ligar y generar un archivo ejecutable con el comando:

**gcc -o <nombre_ejecutable1> <nombre1.o>**

{~/Practica1}[gustavob@quetzal17]$ **gcc -o ConvTemp ConverTemperatura.o**

**2.** Verificar que se haya generado el archivo ejecutable .

{~/Practica1}[gustavob@quetzal17]$ **ls**
ConverTemperatura.c ConverTemperatura.o **ConvTemp**

### 3.2.3 Ejecución

**1.** Ejecutar el programa y observar la salida con:

./<nombredelejecutable>

{~/Practica1}[gustavob@quetzal17]$ **./ConvTemp**
Programa de Conversion de Temperatura
Proporcione la temperatura: 3.651

La temperatura en Celcius es: 3.651000 C y en Fahrenheit 0.00 F

El resultado de la conversión es: 3.651 C = 38.5718
{~/Practica1}[gustavob@quetzal17]$

***Extraído de los apuntes del curso de Introducción a la Programación***

### 3.2.4 Funciones

Una función es un conjunto de sentencias (instrucciones) que cumplen un solo propósito bien definido. Pueden recibir una serie de ***parámetros*** que permitan efectuar una ***serie de instrucciones*** y el resultado generado es ***regresado*** por la función en el punto en donde fue ***invocada***. Estas funciones pueden ser invocadas (llamadas) por otras funciones, como por ejemplo la función principal (***main***).

**Definición 12:** La estructura de **definición** de una **función**:

**Pseudocódigo**

```
Nombre_funcion ( parametro1, parametro2 )
Comienza
        <inicialización (declaración) de variables locales>
        <instrucciones >
        <instrucciones >
        regresar ( <valor> )
Termina
```

**En Código C**

```
tipo_retorno Nombre_funcion ( <tipo> parametro1, <tipo> parametro2)
{
        /*Declaración de variables locales*/
        <tipo> nombre_var;

        /*Instrucciones */

        return ( <valor>);
}
```

Análisis de la estructura de **función** en ***Lenguaje C***

| **Concepto** | **Descripción** |
| --- | --- |
| tipo_retorno | Tipo de valor devuelto por la función. |
| Nombre_funcion | Identificador del nombre de función. |
| (<tipo> parametro1,<br><tipo> parametro2) | Declaración de la lista de parámetros, cada parámetro es separado por comas, contiene el tipo de parámetro (tipificados) y el identificador de la variable. |
| { --- } | Cuerpo de la función |
| Declaración de variables locales | La declaración de variables **sólo** son 'visibles' dentro de la función. |
| return (<valor>) | Valor de regreso de la función. |

### 3.2.5 Procedimientos

Los procedimientos son '***como***' funciones pero con la diferencia que ***no regresan*** un resultado específico. Un procedimiento tiene el objetivo principal de desarrollar una serie de instrucciones sin considerar un valor de regreso.

En el Lenguaje C, se hace uso de la **palabra reservada *void***, la cual indica que el procedimiento (función) no regresa valor alguno.

**Definición 13:** La estructura de **definición** de un **procedimiento**:

**Pseudocódigo**

```
Nombre_procedimiento( parametro1, parametro2 )
Comienza
        <inicialización (declaración) de variables locales>
        <instrucciones >
        <instrucciones >
Termina
```

**En Código C**

```
void Nombre_procedimiento ( <tipo> parametro1,  <tipo> parametro2)
{
        /*Declaración de variables locales*/
        <tipo> nombre_var;

        /*Instrucciones */
}
```

### 3.2.6 Invocación de Procedimiento y Funciones

Primero se describen en pseudocódigo la ***declaración*** de las funciones o procedimientos que se utilizaran y al final se define la estructura del programa principal. En el programa principal es en donde se efectúa la ***invocación*** de las funciones o procedimientos.

El resultado de la invocación de una función, es asignado a una variable, permitiendo así hacer uso del resultado generado dentro del programa principal. En el siguiente tema se explica algunas características de los parámetros.

**Definición 14:** La estructura de **invocación** de **funciones** y **procedimientos**:

**Pseudocódigo**

```
Comienza
        valor ← Nombre_funcion ( parametro1, parametro2 )
        Nombre_procedimiento( parametro1, parametro2 )
Termina
```

**En Código C**

```
{
 valor = Nombre_funcion (parametro1,  parametro2);
  Nombre_procedimiento (parametro1,  parametro2);
```

```
}
```

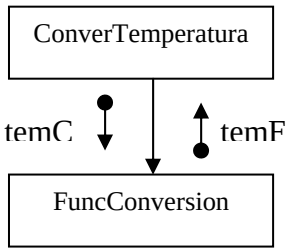
### 3.2.7 Problema "Conversión de Celsius a Fahrenheit"

Dada una temperatura en grados Celsius, efectuar la conversión de grados Celsius a Fahrenheit.

**Entrada:** Temperatura en grados Celsius (tempC).

**Salida:** Temperatura en grados Fahrenheit (tempF).

#### 3.2.7.1 Diseño Modular




#### 3.2.7.2 Código en C

```
/*
  Archivo: ConverTemperatura.c
  Descripción: Este programa efectúa la conversión de temperaturas de
Celsius a Fahrenheit. La lectura de la temperatura se efectúa desde
teclado.
      La salida del resultado se efectúa a consola.
      Objetivo:
        Utilización de funciones con el paso de parametros
  Autor: Basurto Páez Gustavo
  Fecha: 15 de abril de 2008
*/

/*Incluir librerías utilizadas en el programas*/
#include <stdio.h>

//Definicion de Constantes


/*Prototipo de Funciones*/
float FuncConversion (float parametro);


//Variable Global
float ResultadoGlobal=0;

int main(){
    /*Declaración (e inicialización) de variables*/
    float tempC = 0.0;
    float tempF = 0.0;

    printf(" Programa de Conversion de Temperatura \n");
    printf("Proporcione la temperatura: " );

    //Lectura de la temperatura, con formato de flotante : %d
    scanf("%f", &tempC);


    printf(" \n\n\tLa temperatura en Celcius es: %f C y en Fahrenheit
%0.2f F\n", tempC, tempF);
```

```
    tempF = FuncConversion (  tempC);

    printf(" \n\n\tEl resultado de la conversión es: %2.3f C = %2.4f
\n", tempC, tempF);

    return (0);
}


/*Descripcion:  Efectua la conversion de temperaturas
  Entrada: Valor de Temperatura de Celcius es un flotante
  Salida:  El resultado del conversión a Fahrenheit como flotante
  */
float FuncConversion (float parametro)
{
      //Declaracion de variables locales
      float tempF =0.0;

      //Calculo de la temperatura
      tempF = (9.0/5)*parametro + 32;
      //NOTA: si la división no es flotante genera otro resultado

      //modificacion de la variable Global
      ResultadoGlobal = tempF;

      return ( tempF);
}
```

### 3.2.8 Paso a seguir para desarrollar la actividad:

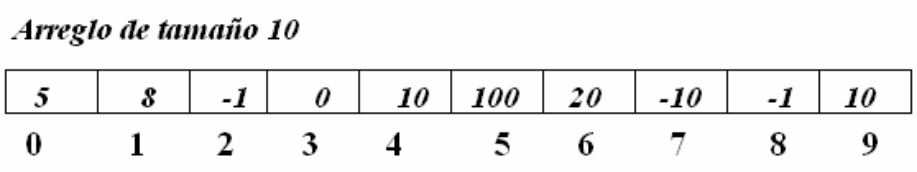
1) En la línea de comando de Linux, cree una carpeta con nombre de Practica1.
2) Utilice un editor de texto para codificar el código proporcionado.
3) Efectué la compilación, ligado y ejecución del mismo.

## 3.3 Actividad 3 “Utilización de Arreglos, Paso de Parámetros por Valor y Referencia”

*Con el desarrollo de esta actividad se alcanzará los objetivos, 10 y 11de la práctica.*

### 3.3.1 Arreglos

Un arreglo es una secuencia de datos del mismo tipo. Cada uno de los datos es considerado como un elemento del arreglo y estos están numerados consecutivamente, como por ejemplo:




Para identificar cada uno de los datos que se encuentran dentro del arreglo, se hace uso de un índice que indica la posición dentro del arreglo. Del ejemplo anterior, se determina que el valor 5 se encuentra en la posición 0, el valor 8 en la posición 1,…. el valor -1 en la posición 8 y el valor 10 en la posición 9.

**Definición 9: Arreglos unidimensionales**

| **Pseudocódigo** |
| --- |
| //**Declaración** de un arreglo especifico<br>arregloEntero: 5, 8, -1, 0, 10, 100, 20, -10, -1, 10 |

```
//Asignación de valor en la posición 2
arregloEntero[2] ← 25

//Lectura de un valor en la posición 8
Valor ← arregloEntero [8]
```

**En Código C**

```
//Declaración de un arreglo especifico
int arregloEntero [ 10 ] = {  5, 8, -1, 0, 10, 100, 20, -10, -1, 10 }

//Asignación de valor en la posición 2
arregloEntero[2] =  25

//Lectura de un valor en la posición 8
Valor = arregloEntero [8]
```

### 3.3.2 Tipo de Paso de Parámetros.

Las funciones en el Lenguaje C presentan la limitante que sólo es posible regresar un valor (de un tipo específico), sin embargo existe una alternativa que permite modificar la información de las variables que se pasan como argumentos de la función simulando que se regresa más de un valor al finalizar el cuerpo de la función.

En el lenguaje C existen dos formas de pasar parámetros a una función:

- Paso de parámetros por Valor
- Paso de Parámetros por Referencia.

#### 3.3.2.1 Paso de Parámetros por Valor

En la mayota de los ejercicios que se han desarrollado en Introducción a la Programación en el definen funciones y procedimientos que utilizan el paso de ***parámetros por valor*** (o paso por 'copia'), indicando que en la declaración de la función ***se recibe una copia del contenido de la variable*** que se pasa como argumento en la invocación de la función. Dentro de cuerpo la función, al modificar el contenido del parámetro, no afecta el contenido de la variable que es pasada como argumento en la invocación.

#### 3.3.2.2 Paso de Parámetros por Referencia

Algunas veces dentro de una función se requiere que se modifique el contenido de las variables que se pasan como argumentos de la invocación en una función, este paso de parámetros se le conoce como ***paso de parámetros por referencia*** (o por dirección).

Sí en la innovación se pasan los argumentos por referencia (o por dirección) y dentro del cuerpo de la función son modificados los valores, entonces al terminar de ejecutarse la función (después de la innovación) el contenido de dichas variables conservan esas modificaciones hechas dentro de la función invocada.

Para efectuar el paso de parámetros por referencia se tienen que indicar algunas características, tanto en el paso de argumentos en la invocación de la función como en la declaración de los parámetros de definición en la función.

El Paso de Parámetros por Valor, se ha explicado con detalle anteriormente en las notas del curso. Tanto el Paso de Parámetros por Valor como el Paso de Parámetros por Referencia no se especifican en Pseudocódigo, ya que es una característica del Lenguaje C.

**Definición 15:** Paso de parámetros por **referencia**:

**Declaración en Lenguaje C**

```
tipo_retorno Nombre_funcion ( <tipo> *parametro1,  <tipo> *parametro2)
{
        /*Declaración de variables locales*/
        <tipo> nombre_var;
        /*El uso de los parámetros se efectúa anteponiendo el símbolo de '*'.*/
        /*Instrucciones */

        return  ( <valor>);
}
```

**Invocación en Lenguaje C**

```
{
        /*A cada uno de los argumentos se le antepone el símbolo '&' */
        valor = Nombre_funcion (&parametro1,  &parametro2);
        Nombre_procedimiento (&parametro1,  &parametro2);
}
```

### 3.3.3 Problema: "Intercambio de valores"

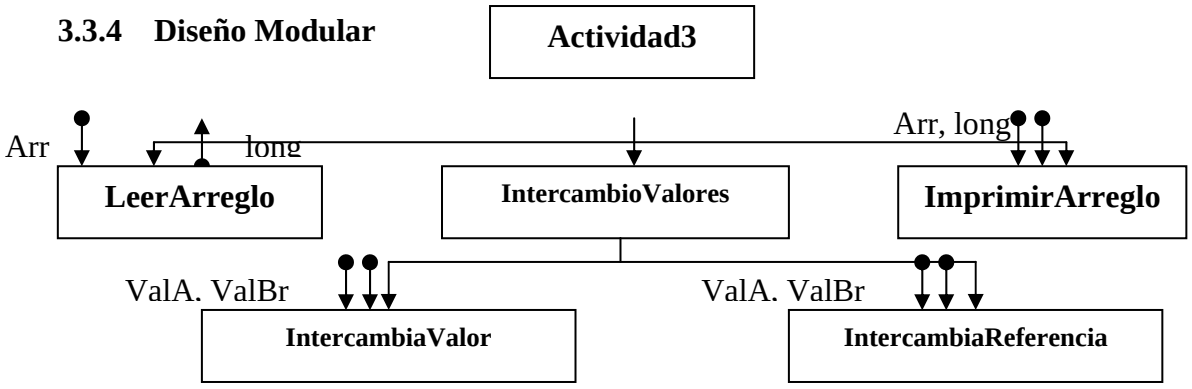
El objetivo es tener un ejemplo del paso de parámetros: ***por valor y por referencia*** mediante el uso de dos procedimientos que se encargaran de efectuar modificaciones en los parámetros que se utilizan dentro del cuerpo de los procedimientos.

Además de considerar el uso de arreglos, para leer la información que el usuario proporciona y para imprimir su contenido.

**Entrada**: --.

**Salida**: Mensajes indicando el cambio del contenido de las variables.

### 3.3.4 Diseño Modular




### 3.3.5 Sugerencias:

- Defina una constante para el tamaño máximo del arreglo:
  #define TAMAX 30
- En la función ***LeerArreglo*** solicitar al usuario que teclee la longitud de arreglo a ingresar que debe ser mayor a 1 y menor que TAMAX, si es posible pueden validar la longitud que el usuario teclea. La longitud es el valor de regreso de la función.
- La función (procedimiento) ***ImprimirArreglo*** recibe como parámetros el arreglo y la longitud, utilice la estructura **for** para recórrelo e imprimir su contenido.

- La definición de la función ***IntercambiaValor*** es la siguiente.

```
/**
*Descripcion: Procedimiento que recibe dos valores de tipo entero, el paso de
*             parámetros es por VALOR. Se efectúan modificaciones en ambos
*             valores, concluyendo que no ven afectados los valores
*             de las variables de la función principal.
* Entrada : ValorA  y ValorB ambos de tipo entero, son pasados por Valor
            (COPIA).
* Salida : nada
**/
void IntercambiaValor(int ValorA, int ValorB)
{
     int auxiliar=0;
     printf(" \n\n\n\t Procedimiento IntercambiaValor (Paso de Parámetros por
            Valor.)");
     printf(" \n\t ValorA : %d \t ValorB: %d.", ValorA, ValorB);
     //Modificaciones
     auxiliar = ValorA;
     ValorA = ValorB;
     ValorB = auxiliar;
     printf(" \n\n\tValores modificados \n\t ValorA : %d \t ValorB: %d.",
            ValorA, ValorB);
     printf(" \n\t Fin Procedimiento IntercambiaValor.");
}
```

- La definición de la función ***IntercambiaReferencia*** es la siguiente.

```
/**
*Descripcion: Procedimiento que recibe dos valores de tipo entero, el paso de
*             parámetros es por REFERENCIA. Se efectúan modificaciones en
*             ambos  valores, concluyendo que la modificaciones dentro del
*             cuerpo de la función, se ven reflejados en valores de las
*             variables de la función principal.
* Entrada : ValorA  y ValorB ambos de tipo entero, son pasados por REFERENCIA
            (DIRECCION).
* Salida : nada
**/
void IntercambiaReferencia(int *ValorA, int* ValorB)
{
     int auxiliar=0;
     printf(" \n\n\n\t Procedimiento IntercambiaReferencia (Paso de Parametros
            por Referencia.)");

      //imprime las direcciones
     //printf(" \n\t ValorA : %d \t ValorB: %d.", ValorA, ValorB);

     printf(" \n\t ValorA : %d \t ValorB: %d.", *ValorA, *ValorB);
     /**El contenido de ValorA es *ValorA **/

     //Modificaciones
     auxiliar = *ValorA;  //El contenido de la 'direccion' de ValorA se asigna
                          a auxiliar
     *ValorA = *ValorB;  //Lo que apunta ValorB se asigna a lo que apunta
                          ValorA
     *ValorB = auxiliar; //El contenido de la varibale auuxiliar se asigna a
                          lo contenido en ValorB

     printf(" \n\n\tValores modificados \n\t ValorA : %d \t ValorB: %d.",
            *ValorA, *ValorB);
     printf(" \n\t Fin Procedimiento IntercambiaValor.");
}
```

- Para la invocación de las funciones (procedimientos) en **Principal:**

```
printf(" \n ValA: %d \t ValB: %d", ValA, ValB);
/*Invocacion del procedimiento, con el paso de parametros por valor*/
IntercambiaValor(ValA, ValB);

printf(" \n\n\n Funcion Principal - Despues del IntercambiaValor");
printf(" \n ValA: %d \t ValB: %d", ValA, ValB);
/*Invocacion del procedimiento, con el paso de parametros por referencia*/
// IntercambiaReferencia(ValA, ValB); //ERROR conversion int to int*
IntercambiaReferencia(&ValA, &ValB);
printf(" \n\n\n Funcion Principal - Despues del IntercambiaReferencia(");
printf(" \n ValA: %d \t ValB: %d", ValA, ValB);
```