ÍNDICE

6 Estructura de un programa

6.1 Diseño de Programación ascendente y descendente

Antes de comenzar a resolver un problema dado, primero se efectúa un Análisis del problema y posteriormente en la fase de Diseño se determina como se resolverá dicho problema mediante diagramas de flujos o pseudocódigos.

Muchos de estos problemas requieren de ser divididos en una serie de ***módulos***. Cada modulo contiene una serie de pasos que permite la resolución de una parte especifica del problema en su totalidad.

Ejemplo: ***Cambio de formato***

Cambiar la fecha de formato 'aaaa/mm/dd' a 'aa/ddd'.

Se requiere efectuar una **Carta de Estructura** para plantear la resolución del problema, esta se describe a continuación:

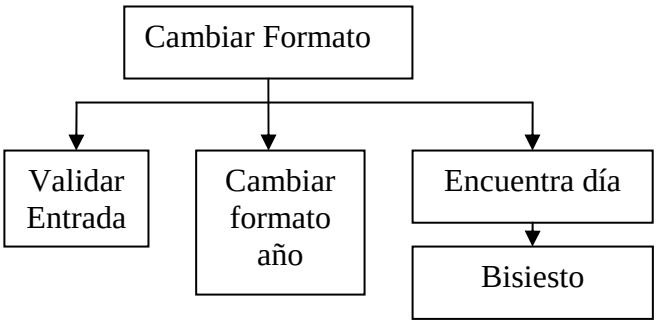


Figura 1.- Carta de Estructura de diseño modular.

6.2 Módulo principal y módulo subordinados.

Un **módulo** se define como una serie de pasos o instrucciones (conocidas como rutinas) que se denominan **funciones o procedimientos**.

Anteriormente se hacia uso del módulo principal en donde se describían todas las instrucciones necesarias para resolver un problema en especifico.

Ahora en adelante se utilizará la estrategia "***divide y vencerás***", en donde se hará uso de sub-módulos que permitan la resolución de los problemas de manera ordenada.

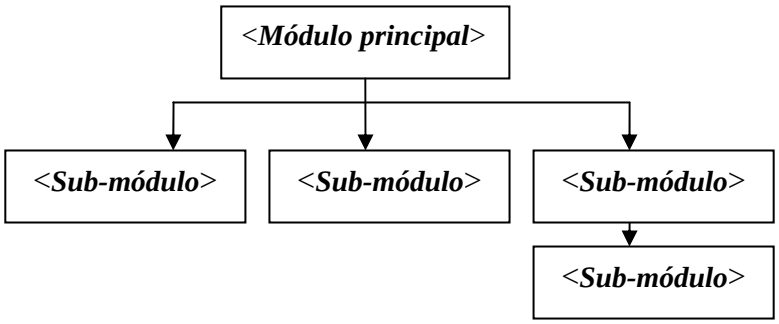


Figura 2.- Estructuración modular de un programa.

C es un **lenguaje de programación estructurado** (también llamado lenguaje de **programación modular**). El modulo principal es conocido como '**main**' (*principal*). Es posible escribir un programa en hace uso de distintos módulos (funciones o procedimientos) para resolver una parte del problema del programa principal.

Los beneficios de hacer uso de funciones o procedimientos, con la estrategia “divide y vencerás”, principalmente son:

-Aislar los problemas.
-Desarrollar funciones o procedimientos que resuelva un problema particular.
-Dar mantenimiento a cada función de manera rápida.
-Efectuar pruebas del funcionamiento correcto de cada función.
-Reutilización de funciones.

6.3 Tipo de Módulos: Funciones y procedimientos.

Las funciones y procedimientos presentan dos ‘secciones’ dentro de un programa (tanto para pseudocódigo y código):

-**Definición**: declaración las instrucciones de la función o procedimiento.
-**Invocación**: llamado de la función o procedimiento, para su utilización.

En los primeros apartados se analizara la definición de funciones y procedimientos, en otro apartado se explicara la invocación de ambas definiciones.

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, <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.

Ejemplo: "***Función suma de dos valores***"

```
int suma ( int valorx, int valory )
{
        int resultado;
        resultado = valorx + valory;
        return (resultado);
}
```

El nombre las funciones deben de ser un identificador valido, pueden comenzar con letra o guión bajo (_), seguido de letras, números o guiones bajos. De la misma forma que en la declaración de las variables, estos identificadores son sensibles a mayúsculas y no se permiten tener más de una declaración con el mismo nombre.

Ejemplos

```
int suma   (int valorx, int valory) ….
int Suma  (int valorx, int valory) ….
int SumA (int valorx, int valory) ….
```

El ***tipo de retorno*** puede ser cualquiera de los tipos básicos y aun estructurados. Los tipos básicos o simples son: int, float, char y double.

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 */
}
```

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

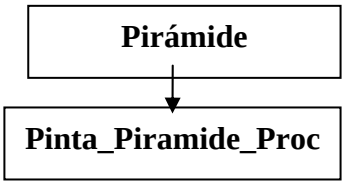
En Código C

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

Ejemplo: "Pirámide en Pantalla con Procedimiento"

Hacer uso de un **procedimiento** para mandar a pantalla una representación de una pirámide de símbolos '*'.

Entrada: --.
Salida: Una representación de una pirámide de '*'.
Carta de Estructura



Pseudocódigo "Pinta_Piramide"
Pinta_Piramide_Proc ( )
Comienza
Escribir (" * ")
Escribir (" *** ")
Escribir (" ***** ")
Escribir ("*******")
Termina

Principal ()
Comienza
Escribir ("Programa que imprime una pirámide:")
Pinta_Piramide_Proc ()

Termina

Código en C

```
/*
  Archivo: Pinta_Piramide.c
  Descripción: Este programa manda a pantalla una representación de
               una piramide de simbolos '*, haciendo uso de un Procedimiento.
              Objetivo:
               Utilización de Procedimiento para el envio de la información a pantalla
  Autor: Basurto Páez Gustavo
  Fecha: 31 de octubre de 2007
*/
/*Incluir librerías utilizadas en el programas*/
#include <stdio.h>

/*Documentación de Función o Procedimiento*/
void Pinta_Piramide_Proc( )
{
    /*Declaración de variables locales*/

    printf(" \n\n\t Inicio del Procedimiento Pinta_Piramide_Proc \n\n\n");
```

```
    printf("\t\t    *   \n");
    printf("\t\t   ***   \n");
    printf("\t\t  *****   \n");
    printf("\t\t *******  \n");
}

int main()
{
     /*Declaración (e inicialización) de variables*/

    printf("\t Programa Pinta Piramide  \n\n ");

    //Invocacion del Procedimiento
    Pinta_Piramide_Proc( );

    //¿Que pasa si se manda a invocar nuevamente la función?
    //Pinta_Piramide_Proc( );

    getch();
    return (0);
}
```

6.4 Parámetros.

6.4.1 Definición.

Algunas funciones (o procedimiento) requieren de algunos parámetros para poder efectuar una serie de instrucciones. Un parámetro es un valor que es 'pasado' a una función para poder se utilizado en para la generación de resultados. Estos parámetros se clasifican en ***parámetros formales*** y ***parámetros actuales***.

6.4.1.1 Parámetros Formales y Parámetros Actuales.

Los parámetros de la **definición** de la función o procedimiento son llamados **parámetros formales**, o comúnmente ***parámetros***.

Los parámetros de la **invocación** de la función o procedimiento son llamados **parámetros actuales**, o comúnmente ***argumentos***.

Los parámetros formales y actuales pueden tener nombres distintos, en la definición de la función es más generalizada que la invocación. En la invocación es posible pasar como argumentos valores específicos.

Es necesario contemplar algunas características fundamentales del uso de los parámetros en funciones y procedimientos, estás se describen a continuación:

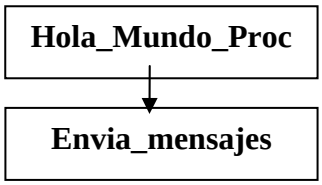
- **Número de parámetros**: Debe de ser el mismo número de parámetros como de argumentos.
- **Tipo de parámetro**: el tipo de valor (o variable) que se pasa como argumento en la invocación de una función, deberá ser compatible con el tipo de parámetro en la definición de la función.
- **Orden en los parámetros**: el orden los parámetros deberá ser respetado en la innovación de acuerdo a la definición de la función o procedimiento.

Ejemplos de procedimientos y funciones

Ejemplo: "Mensaje Hola Mundo con Procedimiento"

Leer el número de veces que se desee mandar a imprimir un resultado, y con el uso de un **procedimiento**, escribir ese número de veces el mensaje.

Entrada: Número entero (numEntero).
Salida: Serie de mensajes indicando: numEntero - "Hola Mundo.
Carta de estructura



Pseudocódigo "Hola Mundo Proc"

Envia_Mensaje (numEntero**)**
Comienza
    para (cont ← 0 **hasta** cont < numEntero **con** cont ← cont + 1) **hacer**
    comienza
        Escribir (cont " Hola Mundo")
    finPara
Termina

Principal ()
Comienza
    Escribir ("Proporcione un número entero:")
    Leer (numEntero)

    Envia_mensaje(numEntero)
Termina

Código en C

```
/*
  Archivo: Hola_Mundo_Proc.c
  Descripción: Este programa lee desde teclado un numero entero
               y manda a pantalla ese numero de veces el mensaje de
               "Hola Mundo" Haciendo uso de un Procedmiento.
              Objetivo:
               Utilización de Procedimiento para el envió de la
               la información a pantalla
  Autor: Basurto Páez Gustavo
  Fecha: 30 de octubre de 2007
*/
/*Incluir librerias utilizadas en el programas*/
#include <stdio.h>

/*Documentación de Funcion o Procedimiento*/
void Envia_Mensaje(int numEntero)
{
    /*Declaración de variables locales*/
    int cont = 0;

    printf(" \n\n\t Inicion del Procedimiento Envia_Mensaje \n");
    //Inicio del ciclo for, el contador empieza desde 0
    //se vaidad hasta que cont sea menor a numEntero
    //por lo tanto se imprimira  diez veces: desde el 0 hasta el 9
    for(cont =0 ; cont < numEntero; cont++)
    {
         printf( " %d.- Hola Mundo con el ciclo For....   \n", cont);
    } //fin for
}
```

```
int main()
{
     /*Declaracion (e inicializacion) de variables*/
    int numEntero = 0 ;

    printf("\t Programa Hola Mundo N-Veces \n\n ");
    printf(" Proporcione un numero entero: " );
    scanf("%d", &numEntero);

    printf("Se imprimiran %d veces el menseje:'Hola Mundo'\n\n ", numEntero );

    //Invocacion del Procedimiento
    Envia_Mensaje(numEntero);

    getch();
    return (0);
}
```

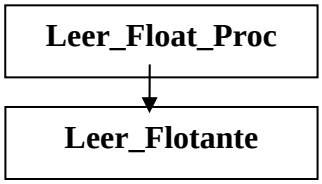
Evidencias

```
                Programa Hola Mundo N-Veces

 Proporcione un numero entero: 10
Se imprimiran 10 veces el menseje:'Hola Mundo'

        Inicio del Procedimiento Envia_Mensaje
0.- Hola Mundo con el ciclo For....
1.- Hola Mundo con el ciclo For....
2.- Hola Mundo con el ciclo For....
3.- Hola Mundo con el ciclo For....
4.- Hola Mundo con el ciclo For....
5.- Hola Mundo con el ciclo For....
6.- Hola Mundo con el ciclo For....
7.- Hola Mundo con el ciclo For....
8.- Hola Mundo con el ciclo For....
9.- Hola Mundo con el ciclo For....
```

Ejemplo "Obtención de un valor"

Efectuar un programa que haga uso de una función para leer un valor flotante (positivo e incluyendo al cero) proporcionado por el usuario. Dentro del programa principal invocar a la función y mandar a imprimir valor obtenido.

Entrada: Número flotante (numFloat), positivo e incluyendo al cero. Validar a la lectura desde e teclado.
Salida: Mensaje indicando el valor leído.



Pseudocódigo "Leer Float Proc"

Leer_Flotante ()
Comienza
    hacer
    Comienza
        Escribir ("Proporcione un número flotante positivo:")
        Leer (valor)
    Termina
    mientras(valor < 0)
    Regresa (valor**)**
Termina

Principal()
Comienza

numeroRes ← Leer_Flotante ( )
Escribir ("El primer valor leido es: " numeroRes)

numeroRes ← Leer_Flotante ( )
Escribir ("El segundo valor leido es: " numeroRes)

Termina

Código en C

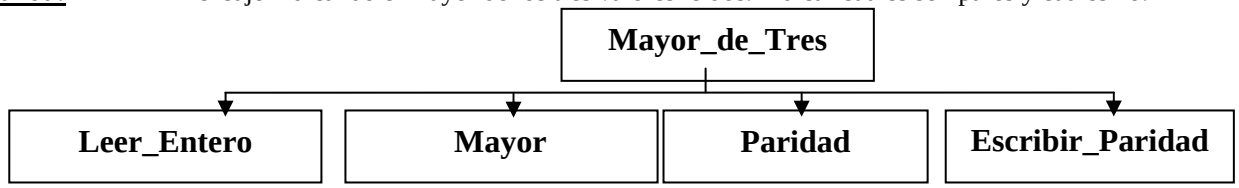
```
/*
  Archivo: Leer_Float_Proc.c
  Descripción: Este programa hace uso de funciones para leer un valor flotante (positivo
             e incluyendo al cero) proporcionado por el usuario. Dentro del programa
             invocar dicha función y mandar a imprimir el resultado.
             Objetivo:
              Utilización de Función para leer la información desde el teclado
             considerando la validación de la información. Especificar el uso de
             variables dentro de cada uno de las funciones.
  Autor: Basurto Páez Gustavo
  Fecha: 04 de octubre de 2007
*/
/*Incluir librerías utilizadas en el programas*/
#include <stdio.h>

/**
*Descripcion: Función que leer un valor flotante desde la consola, al momento que el el
             usuario teclee un valor. Se hace uso de la estructura de control do...while
             para validar que el valor sea positivo (mayor o igual a 0.0)
* Entrada : nada
* Salida : valor leido desde el teclado.
**/
float Leer_Flotante(){
      float valor = 0.0;

      do
      {
            printf("\n Escriba un numero flotante positivo: ");
            scanf("%f", &valor);
      }while(valor < 0.0);

      return (valor);
}

/**Funcion pricipal **/
int main (){
    float numeroRes = 0.0;
    printf("\t Programa Lectura de Valor Flotante \n\n ");

    numeroRes = Leer_Flotante ();
    printf("\n El primer valor leido es : %f \n\n", numeroRes);

    //Reutilizacion de la variable numeroRes
    numeroRes = Leer_Flotante ();
    printf("\n\n\n El segundo valor leido es : %f \n\n", numeroRes);

    printf(" \n\n Presione una tecla para finalizar... ");
    getch();
    return(0);
}
```

Ejemplo "Mayor de Tres Números"

Desarrollar un programa que lea tres números enteros positivos (mayores o iguales a cero), y que se determine cual es el mayor de los tres. Posteriormente evalué cuales de los números leído son pares.

Entrada: Tres números enteros positivos (numEntero) Validar a la lectura desde e teclado.

Salida: Mensaje indicando el mayor de los tres valores leídos. Indicar cuales son pares y cuales no.



Pseudocódigo "Mayor de Tres"

```
Principal ()
Comienza

        valorUno← Leer_Entero ( )
        valorDos← Leer_Entero ( )
        valorTres ← Leer_Entero ( )

        auxiliar  ← Mayor ( valorUno, valorDos )
        mayorTodos ← Mayor ( auxiliar , valorTres)

        Escribir ("El mayor de todos es:" mayorTodos)
        auxiliar  ← Paridad ( valorUno )
        Escribir_Paridad (valorUno,auxiliar)

        auxiliar  ← Paridad ( valorDos )
        Escribir_Paridad (valorDos,auxiliar)

        auxiliar  ← Paridad ( valorTres )
        Escribir_Paridad (valorTres,auxiliar)
Termina

Leer_Entero ()
Comienza
        hacer
        Comienza
                Escribir ("Proporcione un número entero positivo:")
                Leer (valor)
        Termina
        mientras( valor < 0 )
        Regresa ( valor )

Termina

Mayor ( ValorA, ValorB )
Comienza

        Si (  ValorA >= ValorB) entonces
        Comienza
                Auxiliar ← ValorA
        Termina
        Otro
        Comienza
                Auxiliar ← ValorB
        Termina
        Regresa (Auxiliar)
```

Termina

Paridad (Valor **)**
Comienza

Auxiliar ← Valor **mod** 2
Si (Auxiliar = 0)
Comienza
Par ← 1
Termina
Otro
Comienza
Par ← 0
Termina
Regresa (Par **)**
Termina

Escribir_Paridad (valor, resultado**)**
Comienza

Si (resultado = 1 **) entonces**
Comienza
Escribir (" El valor "valor "es PAR")
Termina
Otro
Comienza
Escribir (" El valor "valor "es IMPAR")
Termina
Termina

Código en C

```
/*
  Archivo: Mayor_Tres.c
  Descripción: Este programa hace uso de funciones para:
                    Leer un valor entero (positivo e incluyendo al cero).
                    Determinar cual es el mayor de dos valores.
                    Determinar si es par o impar un valor entero.
                    Escribir un mensaje de acuerdo a su paridad.
                Dentro del programa invocar dichas funciones, de acuerdo a las
                operaciones y mandar a imprimir el resultado.
              Objetivo:
               Utilización de Funciones para leer la información desde el teclado
              considerando la validación de la información, también para hacer
              comparaciones, y determinar si es par.
               Utilización de un Procedimiento para mandar a escribir un mensaje a
              pantalla.
               Especificar el uso de variables dentro de cada uno de las funciones.
               El uso parámetros y argumentos de las funciones y procedimientos.
  Autor: Basurto Páez Gustavo
  Fecha: 04 de octubre de 2007
*/
/*Incluir librerías utilizadas en el programas*/
#include <stdio.h>

/**
*Descripcion: Funcion que leer un valor entero desde la consola, al momento que el
      usuario teclee un valor.
*         Se hace uso de la estructura de control do...while para validar que
*         el valor sea positivo (mayor o igual a 0)
* Entrada : nada
* Salida : valor leido desde el teclado.
**/
int Leer_Entero(){
      int valor = 0;
```

```
      do
      {
            printf("\n Escriba un numero entero positivo: ");
            scanf("%d", &valor);
      }while(valor < 0);

      return (valor);
}

/**
*Descripcion: Funcion que recibe como parametros dos valores enteros,
*         para ser comparados entre si y determinar el mayor de ambos.
* Entrada : ValorA y ValorB, valores enteros.
* Salida : El mayor de los dos valores recibidos.
**/
int Mayor (int ValorA, int ValorB){
      int Auxiliar = 0;
      if( ValorA >= ValorB)
      {
          Auxiliar = ValorA;
      }
      else
      {
          Auxiliar = ValorB;
      }
      return (Auxiliar);
}

/**
*Descripcion: Funcion que recibe como parametro un valor entero,
*         para determinar si es par o impar, es decir, si es divisible
*         entre 2. Esto se hace mediante el uso de la funcion modulo.
* Entrada : Valor de tipo entero que mantiene la cantidad a ser evaludad.
* Salida : El contenido de la variable Par:
*            1: si el Valor es par.
*            0: si el Valor es impar.
**/
int Paridad (int Valor){
      int Par = 0;
      int Auxiliar =0;

      Auxiliar = Valor % 2;

      //Verificacion de paridad
      if(Auxiliar == 0)
      {
          Par = 1;  //El valor de 1 indica que es Par
      }
      else
      {
          Par = 0;  //El valor de 0 indica que es Impar
      }
      return (Par);
}

/**
*Descripcion: Procedimiento que recibe como parametro un valor entero,
*         y un Resultado que indica si el valor es par(1) o impar (0).
*         De acuerdo al resultado se envia el tipo de mensaje.
* Entrada : -Valor: de tipo entero que mantiene uan COPIA de la informacion
*          del valor original.
*           -Resulatdo: DE tipo entero que especifica el resultado de la
*           evaluacion de paridad.
* Salida : nada.
**/
void Escribir_Paridad (int Valor, int Resultado){
      if(Resultado == 1 ) //Es par
```

```
      {
          printf("\nEl valor %d es PAR", Valor);
      }
      else //Es impar
      {
          printf("\nEl valor %d es IMPAR", Valor);
      }
}

/**Funcion pricipal **/
int main (){
    int ValorUno, ValorDos, ValorTres;
    int auxiliar =0;
    int mayorTodos=0;

    printf("\t Programa Mayor de Tres \n ");
    printf("\n Este programa permite la lectura de tres valores enteros positivos\n ");
    printf("para ser evaluados y detemrinar el mayor de los tres. Tambien se determina
      \n");
    printf("cuales valores son pares y cuales no. \n\n");

    //Lectura de los tres valores desde teclado
    ValorUno = Leer_Entero();
    ValorDos = Leer_Entero();
    ValorTres = Leer_Entero();

    //Determinar el mayor de ValorUno y ValorDos
    auxiliar = Mayor (ValorUno, ValorDos);
    //Determinar el mayor del resultado anterior con el ValorTres
    mayorTodos = Mayor (auxiliar, ValorTres);

    printf ("\n\n El mayor de Todos es: %d\n\n", mayorTodos);

    //Analisis de paridad para el ValorUno
    auxiliar = Paridad (ValorUno);
    Escribir_Paridad (ValorUno , auxiliar);

    //Analisis de paridad para el ValorDos
    auxiliar = Paridad (ValorDos);
    Escribir_Paridad (ValorDos , auxiliar);
    //Analisis de paridad para el ValorTres
    auxiliar = Paridad (ValorTres);
    Escribir_Paridad (ValorTres , auxiliar);

    printf(" \n\n Presione una tecla para finalizar... ");
    getch();
    return(0);
}
```

Evidencias

```
                Programa Mayor de Tres
 Este programa permite la lectura de tres valores enteros positivos
 para ser evaluados y detemrinar el mayor de los tres. Tambien se determina
cuales valores son pares y cuales no.

 Escriba un numero entero positivo: 1
 Escriba un numero entero positivo: 2
 Escriba un numero entero positivo: -3
 Escriba un numero entero positivo: 3
 El mayor de Todos es: 3

El valor 1 es IMPAR
El valor 2 es PAR
El valor 3 es IMPAR

 Presione una tecla para finalizar...
```

6.4.1.2 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.

Paso de Parámetros por Valor

En la mayota de los ejercicios que se han desarrollado en este capitulo las funciones y procedimientos 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.

Ejemplo: Paso de parámetro por valor, las variables dentro del procedimiento son modificadas pero dichas modificaciones no se ven reflejadas en la función main.

```
void Procedimiento_paso_valor(int ent, float real)
{
     printf(" \n\n\n\t Procedimiento de Paso de Parametros por Valor.");
     printf(" \n\t Valor ent: %d \t Valor real: %f.", ent, real);

     ent++;
     real = real + 1.5;
     printf(" \n\tValores modificados \n\t Valor ent: %d \t Valor real: %f.", ent,
             real);
     printf(" \n\t Fin Procedimiento de Paso de Parametros por Valor.");
}

int main ()
{
    int ent= 10 ;
    float real = 1.5;
    printf(" \n\n\t Funcion Principal.");
    printf(" \n\t Valor ent: %d \t Valor real: %f.", ent, real);

    /*Invocación del procedimiento, con el paso de parámetros por valor*/
    /*Observacion: los argumentos como los parámetros de la función tienen el
      mismo nombre, como se había comentado anteriormente que pueden tener nombre
      distintos*/

    Procedimiento_paso_valor(ent, real);

    printf(" \n\n\n\t Fin Principal-Valores: \n\t Valor ent: %d \t Valor real:
            %f.", ent, real);
    getche();
    return (0);
}
```

Evidencia:

```
         Funcion Principal.
         Valor ent: 10   Valor real: 1.500000.

         Procedimiento de Paso de Parametros por Valor.
         Valor ent: 10   Valor real: 1.500000.
        Valores modificados
         Valor ent: 11   Valor real: 3.000000.
         Fin Procedimiento de Paso de Parametros por Valor.

         Fin Principal-Valores:
         Valor ent: 10   Valor real: 1.500000.
```

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

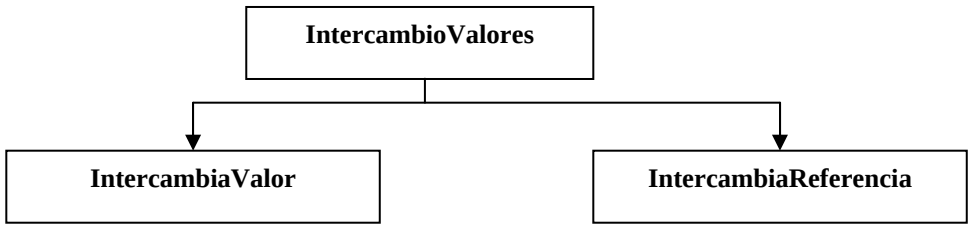
```
{
        /*A cada uno de los argumentos se le antepone el símbolo '&' */
        valor = Nombre_funcion (&parametro1,  &parametro2);
        Nombre_procedimiento (&parametro1,  &parametro2);
}
```

Ejemplo: "***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.

Entrada: --.

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



Pseudocódigo "Intercambio de Valores"

Comienza

```
ValA ← 10
ValB ← 1
Escribir ("El valor de A:" ValA "y el valor de B:"ValB);
IntercambiaValor (ValA, ValB );

Escribir ("El valor de A:" ValA "y el valor de B:"ValB);
IntercambiaReferencia (ValA, ValB );
```

Escribir (“El valor de A:” ValA “y el valor de B:”ValB);
Termina

IntercambiaValor (ValorA, ValorB)
Comienza
auxiliar ← ValorA
ValorA ← ValorB
ValorB ← auxiliar
Termina

IntercambiaReferencia (ValorA, ValorB)
Comienza
auxiliar ← ValorA
ValorA ← ValorB
ValorB ← auxiliar
Termina

Código en C

```
/*
  Archivo: Intercambio_Valores.c
  Descripcion: Este programa explica el paso de parámetros por POR VALOR (Copia) y POR
             REFERENCIA (Dirección).
               Se define dos procedimientos:
                IntercambiaValor: recibe dos parámetros por VALOR y se
                   intercambia el valor de ambos parámetros.
                IntercambiaReferencia: recibe dos parámetros por REFERENCIA
                   y se intercambia el valor de ambos parámetros.
              La función principal se encarga de utilizar dichos procedimientos y
             determinar su funcionamiento del paso de parámetros.
  Autor: Basurto Páez Gustavo
  Fecha: 11 de noviembre de 2007
*/

/*Incluir librerias*/
#include <stdio.h>

/**
*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.");
}

/**
*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.");
}

/**Funcion pricipal **/
int main ()
{
    int ValA = 10 ;
    int ValB = 1;
    printf(" \n\n\n Funcion 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);
    getche();
    return (0);
}
```

Evidencias

Funcion Principal.
ValA: 10 ValB: 1

Procedimiento **IntercambiaValor** (Paso de Parametros por Valor.)
ValorA : 10 ValorB: 1

Valores modificados
ValorA : 1 ValorB: 10
Fin Procedimiento IntercambiaValor.

Funcion Principal - Despues del IntercambiaValor
ValA: 10 ValB: 1

Procedimiento **IntercambiaReferencia** (Paso de Parametros por Referencia.)
ValorA : 10 ValorB: 1.

Valores modificados
ValorA : 1 ValorB: 10.
Fin Procedimiento IntercambiaValor.

Funcion Principal - Despues del IntercambiaReferencia(
ValA: 1 ValB: 10

Tanto el paso de parámetros por referencias como por valor, pueden estar contenidos dentro de una misma función. La sugerencia es tener cuidado en la utilización del paso de parámetros (por valor y por referencias).

El operador **&** permite determinar la **dirección** de la variable. Por esto mismo se indica que se pasa la **referencia**.

Invocación:

```
IntercambiaReferencia(&ValA, &ValB);
```

El operador ***** permite determinar el **contenido** de la variable de referencia. Por esto mismo dentro del método se hace uso de este tipo de operador (también se pude interpretar como '*lo que apunta*'). Ejemplos:

Declaración:

```
void IntercambiaReferencia(int *ValorA, int* ValorB)
```

Asignación:

```
auxiliar = *ValorA;//El contenido de la 'dirección' 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 variable auxiliar se asigna a lo
                   //contenido en ValorB
```

6.5 Anidamiento de Módulos.

6.5.1 Documentación de Funciones

La documentación de funciones es importante como la documentación del encabezado del programa fuente (código). El principal objetivo de tener documentados cada una de las funciones y procedimientos, es describir el funcionamiento de cada una de estas, la reutilización total o parcial, el correcto mantenimiento, etc.

Los atributos que debe de presentar la documentación de cada función y/o procedimiento definidos en un programa, se describen a continuación:

Descripción: Se detalla los objetivos, el proceso y las características importantes que se presentan en las funciones o procedimientos.

Entrada: Hace referencia a los parámetros de la función o procedimiento, en donde se describe el tipo de datos, la utilización, si es por referencia o por valor, etc.

Salida: Describe lo relacionado al tipo de retorno de las funciones.

Ejemplo: "***Documentación de una función o procedimiento***"

```
/**
Descripcion: Procedimiento que recibe como parámetro un valor entero,
        y un Resultado que indica si el valor es par(1) o impar (0).
        De acuerdo al resultado se envía el tipo de mensaje.
Entrada :
      -Valor: de tipo entero que mantiene una COPIA de la información del valor
      original.
      -Resultado: De tipo entero que especifica el resultado de la
          evaluación de paridad.
Salida : nada.
**/
```

6.5.2 Prototipos de la funciones.

Las funciones y procedimientos presentan dos 'secciones' dentro de un programa:

-**Definición**: declaración las instrucciones de la función o procedimiento.

-**Invocación**: llamado de la función o procedimiento, para su utilización.

A nivel de código existe un mecanismo que permite **declarar** las funciones y procedimientos que serán utilizados a lo largo del programa, a la **declaración** de una función o procedimiento es llamada **prototipo**.

Los programas fuentes desarrollados hasta el momento se definen primero las funciones o procedimientos y al final del archivo se describe la función principal (main), si se da el caso de que se defina una función debajo de la función principal, y está la utilice, el compilador mandará un ***mensaje de error*** indicando que se requiere declarar dicha función.

Los **prototipos** de las funciones encuentran en la ***cabecera*** del programa fuente, antes de la función principal, con el objetivo de que el compilador resuelva y verifique la definición de las funciones o procedimientos. Cada declaración de prototipo, debe de ***finalizar con punto y coma*** ';'.

Ejemplo: "***Prototipo de funciones del Intercambio de valores***")

```
/*Incluir librerias*/
#include <stdio.h>

/**Prototipo de funciones**
void IntercambiaValor(int ValorA, int ValorB);
void IntercambiaReferencia(int *ValorA, int* ValorB);

/**Funcion pricipal **/
int main ()
{ …
}
```

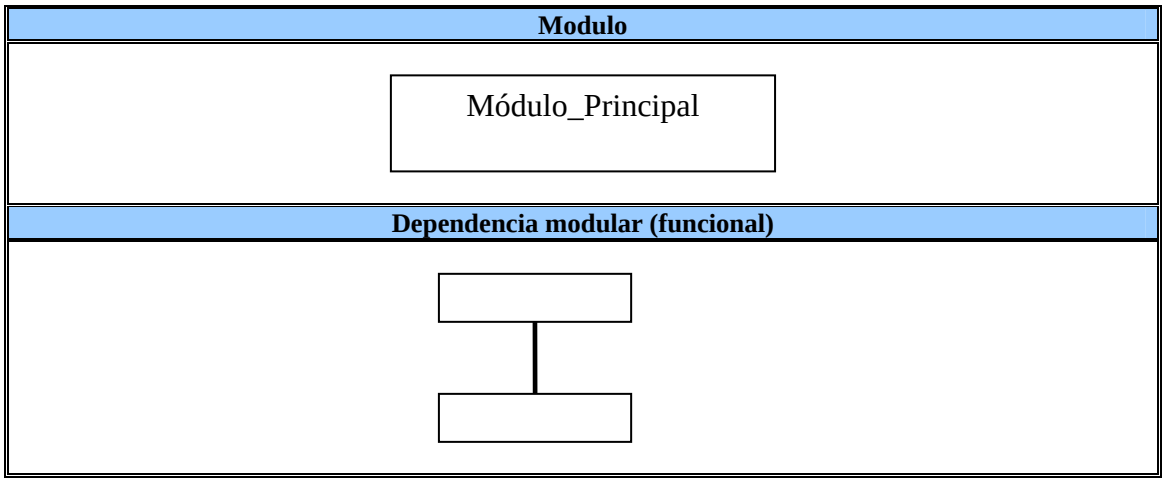
En el ejemplo anterior **las declaraciones de las funciones** dentro del programa fuente (código) no requiere de un orden específico. Sin embargo, ahora se recomienda comenzar con la función principal (***main***) y continuar con las siguientes declaraciones de acuerdo al orden de su utilización dentro del programa.

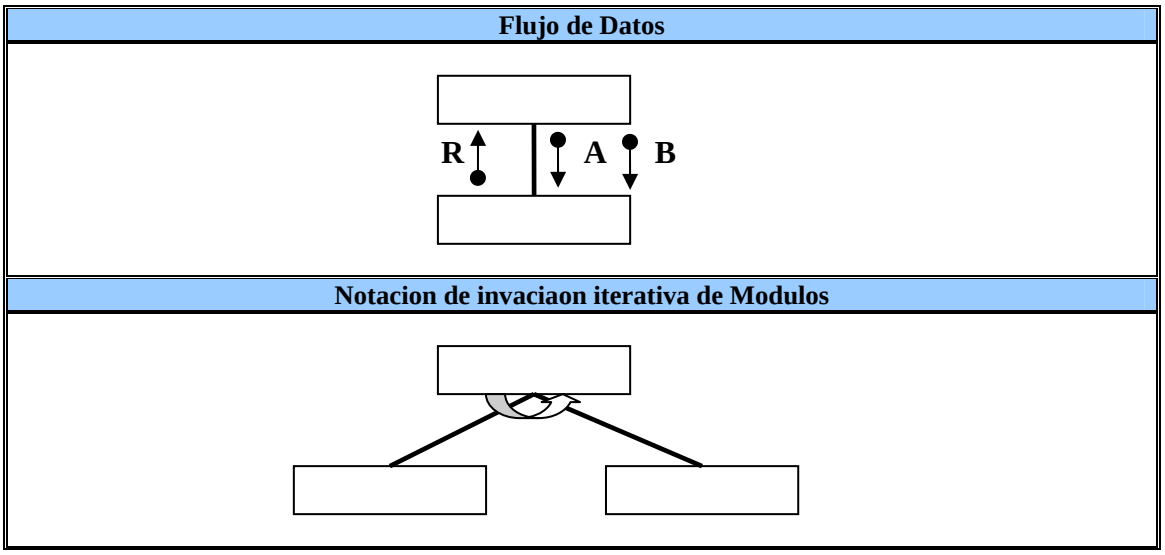
6.5.3 Cartas de Estructura

Una ***carta de estructura*** es una notación grafica que permite la representación estructural de un conjunto de módulos y la interrelación entre dichos módulos. Es importante mencionar que las cartas de estructuras no representan las estructuras de control o de secuenciación, sino por el contrario representa la ***relación entre los módulos***.

Para ser más explicita la comunicación entre los distintos módulos descritos mediante una carta estructural, es necesario especificar los parámetros en la invocación, así como el valor de regreso, mediante el uso de flechas, estas características es conocida como el **flujo de datos.**

Definición 16: Elementos de la **carta de estructura**

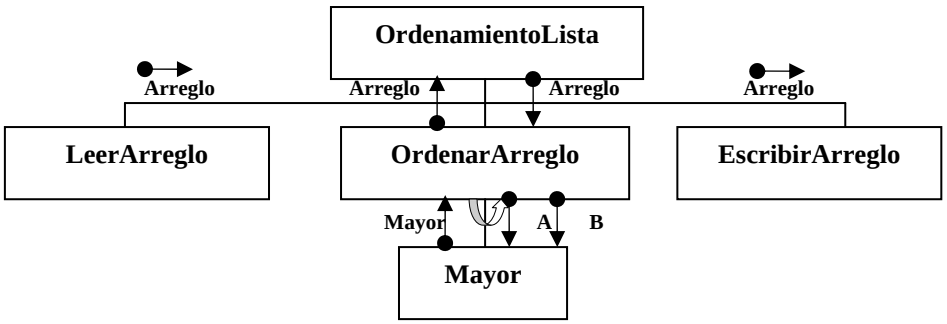




6.5.4 Anidamientos de Módulos

En la mayoría de los programas, el módulo principal hace uso de algunos módulos subordinados para delegar responsabilidades específicas a cada uno de estos módulos subordinados. Cuando un módulo utiliza a otro(s) módulo(s) subordinado(s) se tienen un **anidamiento de módulos**.

Ejemplo: "***Anidamiento de módulos***"



El ejemplo descrito en esta carta estructural se desarrollará más adelante.

6.6 Alcance de los Identificadores.

El ámbito o alcance de una variable determina cuales son las funciones que pueden hacer uso de dicha variable. El ***término de ámbito*** (o alcance) se refiere a una **sección** en específica del programa fuente en donde es posible **hacer uso** de una variable.

Existen distintos tipos de ámbitos, sin embargo en el curso se utilizaran los siguientes: **Programa** y **Función**, mejor conocidas como **Global** y **Local**, respectivamente. De acuerdo a la forma y lugar en el programa fuente, que es declarada la variable es definido el tipo de ámbito:

6.6.1 Ámbito de Programa (Variables Globales)

La **Variables Globales** presentan ámbito global del programa fuente, es decir puede ser utilizada por cualquiera de las funciones que estén definidas en el programa. Para definir una **Variable Global**, esta se debe **declarar** al principio del programa fuente, es decir, deben de estar fuera de las funciones (incluyendo la función principal).

Para tener un esquema adecuado del programa fuente, se recomienda **declarar** las Variables Globales después de la definición de prototipos de las funciones y antes de la función Principal.

Ejemplo: "***Declaración de variables Globales***"

```
/*Incluir librerias*/
#include <stdio.h>

/**Prototipo de funciones**/
void IntercambiaValor(int ValorA, int ValorB);
void IntercambiaReferencia(int *ValorA, int* ValorB);

/**Declaración de Variables Globales**/
int resultado = 0;
float promedio;

/**Funcion pricipal **/
int main ()
{
       printf(" Variables globales %d, %f", resultado, promedio);
…….
```

6.6.2 Ámbito de Función (Variables Locales)

Las **Variables Locales** presentan ámbito específico a una función, y este tipo de variables pueden ser utilizadas únicamente en la función en donde están declaradas. Para definir una **Variable Local**, dentro de una función se debe **declarar** al principio de la misma, es decir, deben de estar después de la definición de la función (justo después de abrir la llave). Las variables locales **NO PUEDEN** ser utilizadas fuera del cuerpo de la función en donde fue declarada.

Las observaciones importantes que son necesarias mencionar son:

- Los parámetros pueden ser considerados como parámetros locales, es decir, tienen un ámbito local dentro de la función.
- Se tiene es determinar el ámbito de las variables respecto al posible **conflicto de identificadores**.

Ejemplo: "***Declaración de variables Locales***"

-Variables Locales para la función principal.

```
int main ()
{
    //Declaración de variables locales
    int ValA = 10 ;
    int ValB = 1;
```

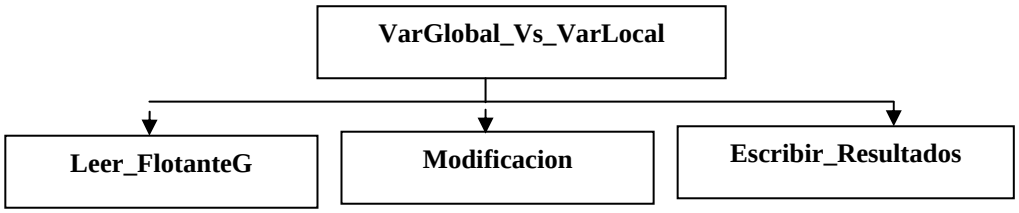
-Variable Local para la función IntercambiaValor.

```
void IntercambiaValor(int ValorA, int ValorB)
{
    //Declaración de variables locales
     int auxiliar=0;
```

Ejemplo: "*Uso de Variable Global y Variable Local*"

Se definen dos tipos de variables, una se inicializará localmente y la otra se leerá desde teclado. Se utilizara una procedimiento para modificar el valor de la variable global y se mandará a imprimir las modificaciones.

Entrada: Un valor de tipo flotante (No requiere de validación).
Salida: Mensaje indicando los cambios efectuados en el contenido de las variables.



Pseudocódigo "*VarGlobalVsVarLocal*"

Comienza
valorLocal ← -10
Leer_FlotanteG ()

Escribir ("Innovación de Procedmiento")
Modificacion ()
Escribir_Resultados ()
Escribir ("El contenido de la variable Local " valorLocal)
Termina

Leer_FlotanteG ()
Comienza
Escribir ("Proporcione un número flotante (real):")
Leer (valorGlobal)
Termina

Modificacion()
Comienza
valorLocal ← 5
Escribir ("Se modifica la variable Global:")
valorGlobal ← valorGlobal * 100
Termina

Escribir_Resultados ()
Comienza
//No es posible definir el valorLocal, esta fuera de su ámbito.
//Escribir ("El contenido de la variable Local " valorLocal)

Escribir ("El contenido de la variable Global " valorGlobal)
Termina

Código en C

```
/*
  Archivo: VarGlobalVsVarLocal.c
  Descripcion: Este programa hace uso de variables globales y locales para
               ejemplificar el uso de ambas variables. Se define un
               procedmiento que modifica la varible Global y se describe la
               modificacion de la variabel local.
  Autor: Basurto Páez Gustavo
  Fecha: 12 de noviembre de 2007
*/

/*Incluir librerias*/
#include <stdio.h>

/**Prototipo de funciones**/
void Leer_FlotanteG();
```

```
void Modificacion();
void Escribir_Resultados();

/**Declaracion e incializacion de Variables Globales**/
float varGlobal;

/**Funcion pricipal **/
int main ()
{
    /**Declaracion e incializacion de Variables Locales**/
    float varLocal = -10 ;

    printf(" \n\n\n Funcion Principal.");
    printf(" \n VarLocal: %f \t varGLobal: %f", varLocal, varGlobal);

    //Lectura de la variable Global
    Leer_FlotanteG();

    printf(" \n\n\n Invocacion del procedimiento. ");

    //Modificacion de las variables
    Modificacion();

    Escribir_Resultados();

    printf(" \n\n\n Funcion Principal - Despues del de la Modificacion");
    printf(" \n VarLocal: %f \t varGLobal: %f", varLocal, varGlobal);

    getche();
    return (0);
}

/**
Descripcion: Procedimiento que efectua la lectura de un valor flotante y es
             asigando a una varibale Globla.
Entrada : nada
Salida : --Modifacion de la variable Global varGlobal
**/
void Leer_FlotanteG( )
{
     /**Declaracion e incializacion de Variables Locales**/

     printf(" \n\n\n\t **Leer_FlotanteG** ");
     printf ("\n\tProporcione un numero flotante (real):");
     scanf("%f", &varGlobal);
     printf(" \n\t VarLocal (no visible) \t VarGlobal : %f", varGlobal);
}

void Modificacion()
{
     /**Declaracion e incializacion de Variables Locales**/
     float varLocal = 5.0;

     printf(" \n\n\n\t **Modificacion** ");
     printf("\n\tSe modifica la variable Global:");
     varGlobal = varGlobal * 10;
     printf("\n\t VarLocal %f \t VarGlobal : %f", varLocal, varGlobal);
}

void Escribir_Resultados()
{
     /**Declaracion e incializacion de Variables Locales**/

     printf(" \n\n\n\t **Escribir_Resultados** ");
     //printf(" \n\t VarLocal %f \t VarGlobal : %f", varLocal, varGlobal);
     printf(" \n\t VarLocal (no visible) \t VarGlobal : %f", varGlobal);
}
```

Evidencias

Funcion Principal.

VarLocal: -10.000000 **varGLobal**: **0.000000**

*****Leer_FlotanteG*****

Proporcione un número flotante (real):3.5

VarLocal (no visible) **VarGlobal : 3.500000**

Invocación del procedimiento.

*****Modificacion*****
Se modifica la variable Global:
VarLocal 5.000000 **VarGlobal** : **35.000000**

*****Escribir_Resultados*****
VarLocal (no visible) **VarGlobal** : **35.000000**

Funcion Principal - Después del de la Modificación
VarLocal: -10.000000 **varGLobal**: **35.000000**

Ejercicio de práctica: ¿Cómo hacerle para modificar la variable local de la función principal, por medio de la función Modificación?

6.7 Ejercicios de Unidad 6

6.7.1 Ejercicios propuestos para completar

Los siguientes ejercicios permiten efectuar el manejo de funciones, se ha proporcionado información para que se complete las secciones que hagan falta.

I. Ejercicios que hacen uso funciones y arreglos.

1. "I***mprimir el contenido de un arreglo***"
Efectuar un programa que defina un procedimiento para leer la información que es proporciona el usuario y permita ser almacenada en el arreglo, posteriormente se desarrollará otro procedimiento que envié a pantalla el contenido de un arreglo.
Entrada:
Salida:
Carta de Estructura:
Utilizar los procedimientos: Leer_Lista e Imprimir_Lista

Pseudocódigo "Imprime Arreglo"
Principal( )
Comienza

Termina

Leer_Lista (arreglo[ ])
Comienza

Termina

ImprimirLista (arreglo[ ])
Comienza

Termina

Código

```
/*
  Archivo: ImprimeArreglo.c
  Descripcion: Este programa recibe numeros enteros de la entrada estandar,
               mediante la funcion Leer_Lista, que se encarga de leer,
               tantos valores enteros como lo indique la constante
               TAMANIO. Posteriormente se manda a imprimir en pantalla
               mediante el uso de un procedimiento. En ambos se hace
               uso de la iteracion condicional For.
  Autor: Basurto Páez Gustavo
  Fecha: 22 de noviembre de 2007
*/

/*Incluir librerias*/
#include <stdio.h>


/**Definicion de constantes*/
#define TAMANIO 10  //Tamanio del arreglo

/**Prototipo de funciones**/
void Leer_Lista(int lista[]);
void Imprimir_Lista(int lista[]);

int main ()
{
    int arreglo[TAMANIO];

    /*********************************
    TODOS LOS ARREGLOS SE PASAN POR REFERENCIA
    (Se pasa la direccion de a primera posicion del arreglo)
    *********************************/
    Leer_Lista(arreglo);

    Imprimir_Lista(arreglo);

    /**Que pasa si en la condicion acceso no es correcta y
    se accesa a otra localidad?
     cont <= TAMANIO  y despue mandar a imprimir
    */

    //Modificacion del arreglo
    arreglo[0]=-1;
    arreglo[1]=-2;
    arreglo[3]=-3;
    arreglo[4]=-4;

    Imprimir_Lista(arreglo);

    getch();
```

```
    return (0);
}

/*
* Descripcion:
* Entrada:
* Salida:
*/
void Leer_Lista(int lista[])
{
     //Declaracion de variables locales
     int cont;

     printf(" Teclee %d enteros.\n", TAMANIO);
     for(cont=0; cont < TAMANIO ; cont++)
     {
          printf("\nTeclee el %d valor: ", cont+1);

               //COMPLETAR CODIGO
     }
}

/*
* Descripcion:
* Entrada:
* Salida:
*/
void Imprimir_Lista(int lista[])
{
     //Declaracion de variables locales
     int cont;

       //     COMPLETAR CODIGO


}
```

Evidencias:

2. "***Buscar un elemento en un arreglo***"

Efectuar un programa que declare un arreglo de tamaño especificado y se defina una función (***Leer_Entero***) que permita leer un número entero mismo que será buscado dentro del arreglo mediante la función ***Buscador*** la cual regresará la posición en donde se encuentra la primer ocurrencia del numero o bien -1 en caso de que no se encuentre dicho valor dentro del arreglo. Reutilice la funcion Imprimir_Lista del ejercicio 1.

Entrada:
Salida:
Carta de Estructura:

Utilizar los procedimientos: Leer_Entero, Imprimir_Lista y Buscador

Pseudocódigo "Buscar Elemento"
Principal( )
Comienza

Termina

Leer_Entero ( )
Comienza

Termina

ImprimirLista (arreglo[ ])
Comienza

Termina

Buscador (valorBuscar, arreglo[ ])
Comienza
//Utilizar while

Termina

Código

```
/*
  Archivo: BuscarElemento.c
  Descripcion: Este programa recibe un numero entero de la entrada estandar,
               y el programa determina si dicho entero se encuentra en un
               arreglo de tamaño TAMANIO y que contiene valores inicializados.
               dentro del codigo fuente. Se hace uso de la iteracion
               condicional While.
  Autor: Basurto Páez Gustavo
  Fecha: 22 de noviembre de 2007
*/
/*Incluir librerias*/
#include <stdio.h>
/**Definicion de constantes*/
#define TAMANIO 10  //Tamanio del arreglo

/**Prototipo de funciones**/
void Imprimir_Lista(int lista[]);
int Leer_Entero();
int Buscador(int valorBuscar, int arreglo[]);

int main ()
{
    //Declaracion del arreglo de tipo entero, con valores inicializados
    int arreglo[TAMANIO]= {-1, 3, 32, -20, 22, 49, 7, 9, 11, 101 };
    int entBuscar=0;
    int exito = -1;
    //exito = -1 NO se encontro, 'otro valor' SI se encontro y esta en la
    //la posicion del valor contenido en la variable

    Imprimir_Lista(arreglo);

    //Leer el entero  a buscar
    entBuscar = Leer_Entero();

    /*******************************
    TODOS LOS ARREGLOS SE PASAN POR REFERENCIA
    (Se pasa la direccion de a primera posicion del arreglo)
    *******************************/
```

```
    //Buscar el entero, La funcion Buscador regresa la posicion en donde
    //se encuentra el entero o -1 si no se encuentra dicho entero
    exito = Buscador (entBuscar, arreglo);

    if(exito == -1 )
    {
        printf("\n\n El Entero %d NO se encuentra en el arreglo...\n",entBuscar );
    }
    else
    {
        printf("\n\n  El  Entero  %d  se  encuentra  en  la  posicion  %d  del  arreglo.
...\n",entBuscar, exito );
    }
    getch();
    return (0);
}

/*
* Descripcion:   //COMPLETE
* Entrada:
* Salida:
*/
void Imprimir_Lista(int lista[])
{
     //Declaracion de variables locales
     int cont;

     printf(" Lista de Tamaño: %d .\n", TAMANIO);
     //COMPLETE CICLO FOR

     {
          printf("%d, ", lista[cont]);
     }
     printf(" \n\n");
}

/*
* Descripcion: //COMPLETE
* Entrada:
* Salida:
*/
int Leer_Entero()
{
     //Declaracion de variables locales
     int valor;
     printf("\n\n Teclee un numero entero: ");
     scanf("%d", &valor);
     return valor;
}

/*
* Descripcion: //COMPLETE
* Entrada:
* Salida:
*/
int Buscador(int valorBuscar, int arreglo[])
{
     //Declaracion de variables locales
     int cont, buscador;
     int bandera=0; //'0' indica que no se ha encontrado
     int posicion =-1;
     cont=0;
     //No es necesario recorrer todo
     while( cont < TAMANIO && bandera == 0)
     {
            if(valorBuscar == arreglo [cont])
            {
                 bandera = 1; //encontrado
                 posicion = cont; //en la posicion dada
            }
            //No es necesario el else
            cont++;
     }
     return (posicion);
}
```

Evidencias:

3. "***Buscar el elemento mayor en un arreglo - While***"

Efectuar un programa que declare un arreglo de tamaño especificado y se defina una función (***BuscarMayor***) que permita recorrer todo el arreglo buscando el elemento mayor, dicha función deberá de regresar la posición en donde se encuentra el elemento mayor del arreglo. Reutilice la función Imprimir_Lista del ejercicio 1.

Entrada:
Salida:
Carta de Estructura:

Utilizar los procedimientos: Imprimir_Lista y BuscaMayor

Pseudocódigo "Buscar Mayor"
Principal( )
Comienza

Termina

ImprimirLista (arreglo[ ])
Comienza

Termina

BuscaMayor (arreglo[ ])
Comienza

//Utilizar while

Termina

Código

```
/*
  Archivo: BuscarMayor.c
  Descripcion: Este programa determina el entero mayor en un arreglo
               arreglo de tamaño TAMANIO y que contiene valores inicializados.
               dentro del codigo fuente. Se hace uso de la iteracion
               condicional While.
  Autor: Basurto Páez Gustavo
  Fecha: 22 de noviembre de 2007
*/
/*Incluir librerias*/
#include <stdio.h>
/**Definicion de constantes*/
#define TAMANIO 10  //Tamanio del arreglo

/**Prototipo de funciones**/
void Imprimir_Lista(int lista[]);
```

```
int BuscaMayor(int arr[]);

int main ()
{
    //Declaracion del arreglo de tipo entero, con valores inicializados
    int arreglo[TAMANIO]= {-1, 3, 32, -20, 101, 49, 7, 9, 11, 22 };
    int posicion = 0;
    int Mayor;

    Imprimir_Lista(arreglo);

    /*********************************
    TODOS LOS ARREGLOS SE PASAN POR REFERENCIA
    (Se pasa la direccion de a primera posicion del arreglo)
    ********************************/
    //Buscar el entero mayor, La funcion BuscaMayor regresa la posicion en donde
    //se encuentra el entero mayor de todo el arreglo
    posicion = BuscaMayor (arreglo);
    Mayor = arreglo[posicion];
    printf("\n\n El Entero Mayor del arrelgo es: arreglo[%d]=%d  \n",posicion, Mayor );

    getch();
    return (0);
}

/*
* Descripcion:  //COMPLETAR
* Entrada:
* Salida:
*/
void Imprimir_Lista(int lista[])
{
     //Declaracion de variables locales
     int cont;

     printf(" Lista de Tamaño: %d .\n", TAMANIO);
     for(cont=0; cont < TAMANIO ; cont++)
     {
          //COMPLETAR
     }
     printf(" \n\n");
}
/*
* Descripcion: //COMPLETAR
* Entrada:
* Salida:
*/
int BuscaMayor(int arr[])
{
     //Declaracion de variables locales
     int cont;
     int mayor;
     int posicion =0;
     //Suponer qu el mayor es el primero
     mayor = arr[posicion];
     cont=1; //Revisar el siguiente
     //ES necesario recorrer TODO el Arreglo
     while( cont < TAMANIO)
     {
            if(mayor < arr [cont]) //se encuentra un valor mayor
            {
                 mayor = arr[cont];
                 posicion = cont; //en la posicion dada
            }
            //No es necesario el else
            cont++;
     }
     return (posicion);
}
```

4. "***Buscar el elemento mayor en un arreglo - For***"
Desarrolle el mismo programa del ejercicio 3, pero modifique el cuerpo de la función ***BuscarMayor***, y utilice la estructura de control iterativa **for**.

6.7.2 Ejercicios planteados

Los siguientes ejercicios permiten ejercitar aspectos prácticos de la unidad 6. Algunos ejercicios son propuestos el desarrollo del Algoritmo, Diagrama de Flujo y/o Pseudocódigo, antes de comenzar a efectuar el Código en C.

Para el desarrollo de los siguientes ejercicios se requiere efectuar previamente las fases de Análisis y Diseño, ya que se contempla desarrollar las siguientes fases: Codificación, Compilación, Ejecución y Pruebas, Documentación y Mantenimiento.

I. Ejercicios que hacen uso funciones.

1. "***Suma de dos valores***"

Efectuar la suma de dos números flotantes proporcionados por el usuario. Considerar el uso de Funciones para la resolución del problema. Desarrolle la función Suma (valor1, valor2) y utilícela en la función principal. Reutilice y modifique la función **Leer_Flotante()** definida en el ejercicio "**Obtención de un valor**", para que pueda leer valores negativos.

2. "***Juego de Apuestas Modularizado***"

Del ejemplo que se definió en las notas del curso en el capitulo 5, efectuar el desarrollo de manera modular haciendo uso de funciones y procedimientos.

3. "***Tipo de carácter***"

Diseñe un programa que mediante el uso de funciones y procedimientos, lea un carácter y que escriba un mensaje en pantalla, indicando si el carácter tecleado fue una vocal o una consonante, o si es un digito o ninguno de los casos anteriores. Sugerencias: Haga uso de la tabla ASCII para determinar los rangos para cada una de las características solicitadas. Las funciones, por ejemplo del tipo Es_Vocal, es necesario que regrese un valor para saber si se cumple que sea vocal o no. Puede hacer uso del siguiente diagrama modular.