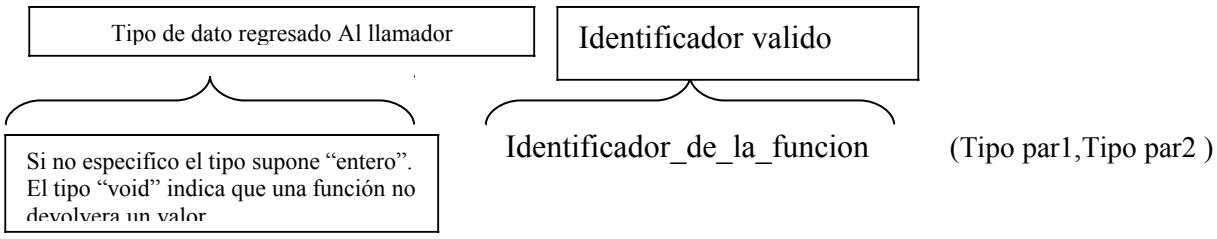
FUNCIONES

Las funciones son el medio básico de que se vale “C” para construir programas. Un Programa es, básicamente, una colección de funciones entre las que se incluye una especial llamada main(),”la función principal” que es donde comienza la ejecución del programa. Algunos programas sencillos como los ejemplos que hemos visto tienen al menos esta función principal diseñada a partir del pseudocódigo o diagrama de flujo, en donde se utilizaron además de las estructuras propias del lenguaje, algunas funciones de la librería estandar y de math.h. Pero cuando éstos programas crecen un poco necesitamos estructurarlos adecuadamente para mantenerlos legibles, facilitar su mantenimiento y para poder reutilizar ciertas porciones de código. El mecanismo en C que nos permite hacer esto son las funciones. Con los compiladores, los fabricantes nos proporcionan un conjunto importante de funciones de librería como ya vimos pero a veces no alcanza. Ahora vamos a aprender como se diseñan funciones creadas específicamente por nosotros y mas adelante con ellas formar nuestras propias librerías. Todas la funciones tienen una estructura similar a main() así es que tienen un nombre o identificador que no puede ser main ni ninguna otra palabra reservada , debe ser un identificador valido , debe estar seguido de paréntesis, entre los cuales se le pasa el dato o los datos si es necesario y el cuerpo de la función está compuesto por un conjunto de declaraciones y de sentencias (en donde se resuelve parte del problema planteado) comprendidas entre llaves. Veamos la estructura general:

Una función tiene generalmente la forma siguiente:




```
{
        Declaraciones;
        sentencias;
        return( valor de retorno);
}
```

Ejemplo: esta función determina el mayor de dos números distintos

```
int maximo ( int a, int b )       /*tipofunc. Nombrefunc. Y lista parámetros*/

{  int max;                 /*declaración local a la función*/
   if( a > b )
     max= a;
   else
      max= b;
   return(max); }       /*valor de retorno */
```

Ya hemos visto como diseñar las funciones básicamente es como diseñar un unico programa solo que debemos llamarlas por su nombre, pasarles el o los parámetros (datos) ,y asignar o mostrar los resultados pero veamos cómo debemos declararlas (como a las variables y los arreglos).

Los prototipos de funciones son una característica clave de la recomendación ANSI del C. Un prototipo es una declaración que toma la forma:

Tipo_resultado Nombre_función (tipo_parámetro nombre_parámetro ...);

Ejemplos de prototipos:

int maximo (int a, int b);

int cero (double a);

long raiz (long valor);

void final_countdown (void);

Observando el prototipo de una función podemos decir exactamente que tipo de parámetros necesita y que resultado devuelve. Si una función tiene como argumento void, quiere decir que no tiene argumentos, al igual que si el resultado es void, no devuelve ningún valor.

Otra parte importante es la llamada , cuando la funcion es llamada o invocada desde main o de cualquier otra funcion se debe hacer por el nombre y a continuación entre paréntesis se le deben pasar los datos reales que debes tener el mismo tipo que los parámetros formales creados en la funcion y en el prototipo .Si la funcion retorna un resultado ,lo debe volcar mediante una asignación en la funcion llamadora o mediante una salida .

Ejemplo de aplicación:

```
/*    Encuentra el máximo de tres enteros */

#include <stdio.h>

int Maxi( int x, int y, int z ); /* prototipo de la función */

/* la función main comienza la ejecución del programa */
int main()
{
   int num1; /* primer entero */
   int num2; /* segundo entero */
```

```
   int num3; /* tercer entero */

   printf( "Introduzca tres números enteros: " );
   scanf( "%d%d%d", &num1, &num2, &num3 );

   /* num1, num2 y num3 son los argumentos
      para la llamada a la función Maxi */
   printf( "El maximo es: %d\n", Maxi( num1, num2, num3 ) );

   return 0; /* indica que main no retorna nada */

} /* fin de main */



/* Definición de la función máximo */
/* x, y, y z son parámetros */
int Maxi (int x, int y, int z )
{
   int max = x;     /* asume que x es el mayor */

   if ( y > max ) { /* si y es mayor que max, asigna y a max */
      max = y;
   }

   if ( z > max ) { /* si z es mayor que max, asigna z a max */
      max = z;
   }

   return (max);      /* max es el valor más grande */

} /* fin de la función máximo */
```

Ámbito de funciones y variables clases de almacenamiento.

El ámbito, o visibilidad, de una variable nos indica en que lugares del programa está activa esa variable. Hasta ahora, en los ejemplos que hemos visto, se han utilizado variables definidas en el cuerpo de funciones. Estas variables se crean en la memoria del ordenador cuando se llama a la función y se destruyen cuando la función termina de ejecutarse son las que llamamos locales a la función por ejemplo las que son declaradas en main() . Es necesario a veces, que una variable tenga un valor que pueda ser accesible desde todas las funciones de un mismo fuente, e incluso desde otros fuentes las llamadas globales.

En C, el ámbito de las variables depende de dónde han sido declaradas y si se les ha aplicado algún especificador de clase almacenamiento: auto, register , extern y static.

Una variable definida en una función es, por defecto, una variable automática sin necesidad de declararla como tal por ej :

auto flota var1;

Esto es, que sólo existe y puede ser accedida dentro de la función o el bloque donde es declarada.

Otra clase de almacenamiento automática, de duración y acceso limitado es register. En el caso en que necesitara ahorrar tiempo de ejecución puedo declarar una variable que es utilizada de manera intensa como register para que la variable sea almacenada en uno de los registros del hardware

Ejemplo:

register int cont=0;

Para que una variable sea visible desde una función cualquiera del mismo fuente debe declararse fuera de cualquier función. Esta variable sólo será visible en las funciones definidas después de su declaración. Por esto, el lugar más común para definir las variables globales es antes de la definición de ninguna función.

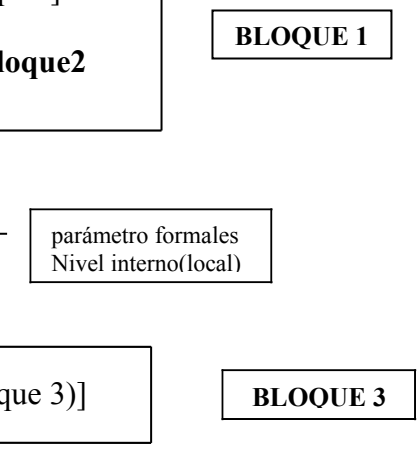
Por defecto, las variables globales y los nombres de función (prototipo o declaración de función) son extern o sea retienen el valor a lo largo de toda la ejecución del programa y desde cualquier lugar del archivo.

Las variables locales llevan implícito el modificador auto. Este indica que se crean al inicio de la ejecución de la función y se destruyen al final. En un programa sería muy ineficiente en términos de almacenamiento que se crearan todas las variables al inicio de la ejecución. Por contra, en algunos casos es deseable. Esto se consigue anteponiendo el modificador static a una variable local. Si una función necesita una variable que únicamente sea accedida por la misma función y que conserve su valor a través de sucesivas llamadas, es el caso adecuado para que sea declarada local a la función con el modificador static.

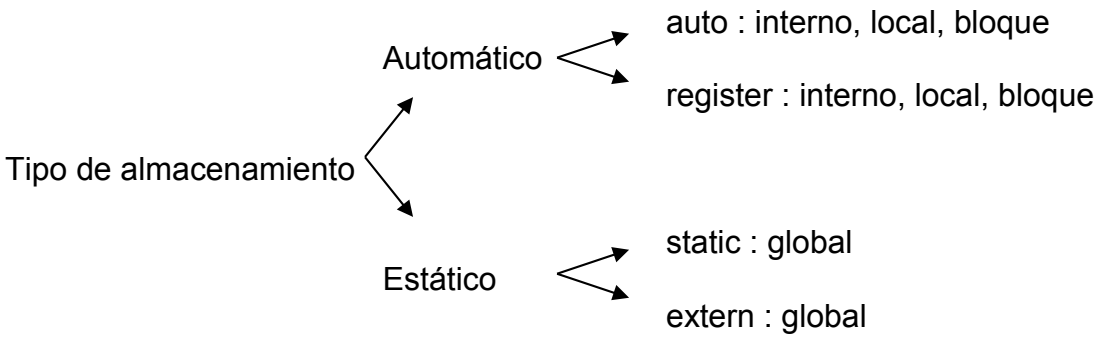
```
float Function (int arg1 , float arg2);    /*…PROTOTIPO */

        [Nivel externo (global)]
main()
{
        [Nivel intermedio local a main oBloque1]
        {
        [Nivel interior(local a Bloque2 )]  Bloque2
        }
}

[Nivel externo (global)]
float Function (ant arg1 , flota arg2)
{
        [Nivel interno(local a Function o bloque 3)]
}
```



Cuadro resumen



Ejemplo de alcance

```
#include <stdio.h>

void usoLoc( void );      /* prototipo de función */
void usoStatic( void ); /* prototipo de función */
void usoGlob( void );     /* prototipo de función */

int x = 1; /* variable global */

/* la función main comienza la ejecución del programa */
int main()
{
   int x = 5; /* variable local a main */

   printf("x local fuera del alcance de main es %d\n", x );

   { /* comienza el nuevo alcance */
      int x = 7; /* variable local con nuevo alcance */

      printf( "x local en el alcance interno de main es %d\n", x );
   } /* fin de nuevo alcance */

   printf( "x local en el alcance externo de main es %d\n", x );

   usoLoc();      /* usoLoc contiene a automatic local x */
   usoStatic(); /* usoStatic contiene a static local x */
   usoGlob();     /* usoGlob utiliza global x */
   usoLoc();      /* usoLoc reinicializa automatic local x */
   usoStatic(); /* static  x retiene su valor previo */
   usoGlob();     /* x global también retiene su valor */

   printf( "\nx local en main es %d\n", x );

   return 0; /* indica terminación exitosa */

} /* fin de main */
```

```
/* usoLocal reinicializa a la variable local x durante cada llamada */
void usoLoc( void )
{
   int x = 25;  /* se inicializa cada vez que se llama usoLoc */

   printf( "\nx local en usoLoc es %d después de entrar a usoLoc\n", x );
   x++;
   printf( "x local en usoLoc es %d antes de salir de usoLoc\n", x );
} /* fin de la función usoLoc */

/* usoStatic initializa la variable static local x sólo la primera vez
   que se invoca a la función; el valor de x se guarda entre las llamadas a esta
   función */
void usoStatic( void )
{
   /* se inicializa sólo la primera vez que se invoca usoStatic */
   static int x = 50;

   printf( "\nlocal static x es %d al entrar a usoStatic\n", x );
   x++;
   printf( "local static x es %d al salir de usoStatic\n", x );
} /* fin de la función usoStatic */

/* la función usoGlob modifica la variable global x durante cada llamada */
void usoGlob( void )
{
   printf( "\nx global es %d al entrar a usoGlob\n", x );
   x *= 10;
   printf( "x global es %d al salir de usoGlob\n", x );
} /* fin de la función usoGlobl */
```

Trabajo Practico numero 11

Actividad 1:

Diseñar un programa que simule un calculador de bolsillo y muestre un menú donde se puedan seleccionar estas distintas opciones :
1-sumar
2-multiplicar
3-restar
4-dividir
5-salir del programa
Y ejecutar las operaciones de acuerdo a la opción elegida entre dos numeros utilizando funciones del tipo void y que no retornan
Diagrama de flujo y pseudocodigo

Actividad 2:
Codificar en C el ejercicio anterior.
en el Laboratorio de informática

Actividad 3:
Diseñar un programa ídem actividad anterior pero utilizando funciones que retornen un resultado y con paso de parámetros.
Diagrama de flujo y pseudocódigo

Actividad 4:
Codificar en C el ejercicio anterior.
en el Laboratorio de informática

Actividad 5:
Diseñar un programa que llame a una función que pueda llevar la cuenta de cuantas veces es llamada (uso de variables static)