LENGUAJE C: PROGRAMACION ESTRUCTURADA

- Características deseables de un programa:
  - Funcionalmente **correcto**: produce los resultados requeridos.
  - **Legible**: fácilmente comprensible por cualquier programador.
  - **Modificable**: la incorporación de modificaciones debe ser sencilla.
  - **Depuración**: diseñado para facilitar la localización y corrección de errores.
  - **Documentado**: incluir comentarios y documentación suplementaria para facilitar su posterior modificación y/o actualización.
- Técnicas para cumplir los requesitos: programación a) **estructurada** y b) **modular**.
- **Programación estructurada**:
  - Todo programa tiene un único punto de inicio y un único punto de fin.
  - Uso de las estructuras de control secuenciales, selectivas y repetitivas.
  - Prohibidos los **saltos** de una instrucción a otra (palabra reservada **goto**).
  - La programación convencional genera programa largos y difíciles de mantener.

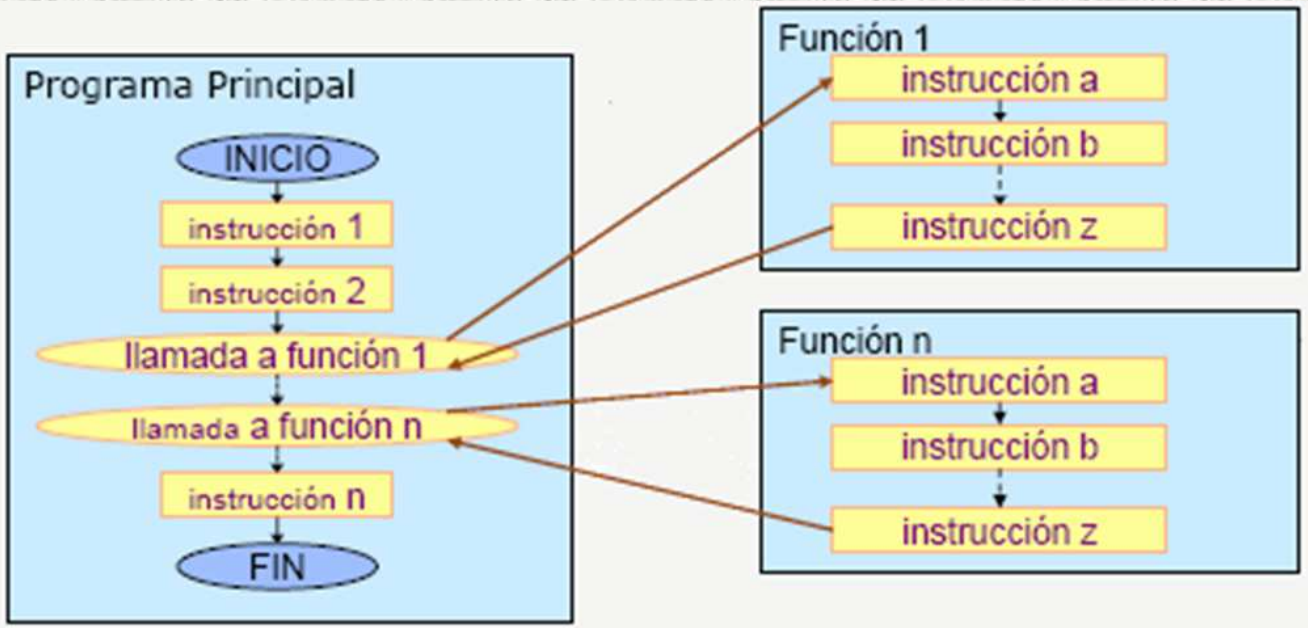
LENGUAJE C: PROGRAMACION MODULAR

- Se basa en la descomposición del problema en problemas más simples (**módulos**) que se pueden analizar, programar y depurar independientemente.
- **Módulo = función = subprograma**: conjunto de instrucciones (**sentencias**) que realizan una tarea concreta y/o proporcionan resultados y que pueden ser llamados (invocados) desde el programa principal o desde otros módulos.
- Ventajas:
  - Se producen programas estructurados y legibles (son más cortos y simples).
  - Las funciones se pueden escribir, compilar y depurar en forma independiente. En un programa de gran tamaño pueden trabajar distintos programadores.
  - Se puede modificar (actualizar) una función sin afectar el resto del programa.
  - Las funciones son reutilizables: pueden ser usadas por distintos programas que requieran la misma funcionalidad.

LENGUAJE C: PROGRAMACION MODULAR

- **La adecuada división de un programa en subprogramas constituye un aspecto fundamental en el desarrollo de cualquier programa.**

- Un programa consta de:
  - Programa principal: contiene operaciones fundamentales y las llamadas a las funciones. El programa principal en C es la función **main**.
  - Funciones: programas independientes que resuelven un problema particular. Se pueden invocar desde el programa principal o desde cualquier otra función.



LENGUAJE C: FUNCIONES

- Toda función está compuesta por:
  - Su **nombre**, por el cuál es invocada.
  - El tipo de **resultado** que retorna (devuelve).
  - Los **parámetros** o **argumentos** (datos) que utiliza para su tarea.
  - Las instrucciones (**sentencias**) que realizan la tarea necesaria.
- Debe ser fácilmente **modificable** (posibilidad de incorporar actualizaciones).
- **Declaración** = **prototipo**: antes de utilizar la función debe ser declarada mediante una forma predefinida. Se informa al compilador de la existencia de una función que será **implementada** (**definida**) más adelante. Debe finalizar con ';' y se incluye al comienzo del programa (después de las directivas #define y #include). Está formado por el tipo de dato que retorna el nombre y los argumentos:

  tipo nombre (lista de argumentos);

  Ejemplos: float suma (float n1, float n2); int potencia (int base, int exp);

  void mostrarDatos (int a, int b); char leerDato (void);

LENGUAJE C: FUNCIONES

- **Definición** = **implementación**: responde a la siguiente estructura.

```
tipo  nombre (lista de argumentos)          /* sin ';' */
{
   . . .;      /*  definición de variables y sentencias = cuerpo de la función */
   return (valor);
}
```

Debe ubicarse siempre después de su prototipo (puede ser a continuación del mismo). No puede estar dentro de otra función.

- **Argumentos**: para cada uno hay que indicar el tipo y el nombre (se puede omitir). Se separan con ',' y si no se utiliza ninguno se escribe **void**. La función utiliza una **copia de los valores** (**parámetro por valor**).

- Ejemplo:

```
float suma (float n1, float n2)
{
    float sum;
    sum = n1 + n2;
    return (sum);
}
```

```
float suma (float n1, float n2)
{
    return (n1+n2);
}
/* menos legible y más difícil
   para buscar errores */
```

LENGUAJE C: FUNCIONES – EJEMPLO 1

```
#include <stdio.h>                          /* librería estándar para entrada / salida */
int suma (int op1, int op2);                /* DECLARACION de la función suma() */
int main (void)                             /* función principal sin parámetros de entrada */
{
    int n1, n2, resultado;                  /* declaración de las variables de main() */
    printf ("Ingrese dos números \n");      /* imprime mensaje en la pantalla */
    scanf("%d",&n1);                        /* lectura del primer valor */
    scanf("%d",&n2);                        /* lectura del segundo valor */

    resultado = suma(n1, n2);               /* LLAMADA a la función suma() */

    printf ("La suma de %d y %d es %d", n1, n2, resultado);   /* imprime */
    return 0;                               /* se espera que main() retorne un entero */
}
int suma (int op1, int op2)                 /* DEFINICION de la función suma() */
{
    int  res;                               /* declaración de las variables de suma() */
    res = op1 + op2;
    return res;                             /* retorna res al programa que la invocó  */
}
```

LENGUAJE C: FUNCIONES – EJEMPLO 2

```
#include <stdio.h>                          /* librería estándar para entrada / salida */
void demo (int valor);                      /* DECLARACION de la función demo() */
void main (void)                            /* función principal sin parámetros de entrada */
{
    int n = 10;                             /* declaración de las variables de main() */
    printf ("Valor de n antes de llamar a demo = %d\n", n);
    demo(n);                                /* LLAMADA a la función demo() */
    printf ("Valor de n después de llamar a demo = %d\n", n);
    /* return ;  */                         /* no es necesario porque main() no tiene que retornar */
}
void demo (int valor)                       /* DEFINICION de la función demo() */
{
    printf ("Valor de n dentro de demo = %d\n", valor);
    valor = 999;
    printf ("Valor de n modificado dentro de demo = %d\n", valor);
}
```

Salidas:

```
Valor de n antes de llamar a demo = ?
Valor de n dentro de demo = ?
Valor de n dentro de demo = ?
Valor de n después de llamar a demo = ?
```

LENGUAJE C: FINALIZACION DEL PROGRAMA

- Se llega al final de la función main().
- La función main() realiza una llamada a **return**.
- Cualquier función realiza la llamada a **exit**(código). En código se puede pasar un valor que represente a un error o 0 si el programa termina sin errores.

Ejemplo:

```
void main (void)
{
    float num = 5.2e-16;
    exit (0);
    printf ("El valor de num es: %f",num);
}
```

Salida: no imprime nada porque no se alcanza a ejecutar la función printf().

LENGUAJE C: BIBLIOTECAS ESTÁNDAR

- En el estándar C se definen un conjunto de **librerías** de funciones de soporte para realizar las operaciones que se realizan con mayor frecuencia con sólo invocar la función, sin necesidad de escribir el código fuente de la misma.
- Las **declaraciones** de las funciones se realizan en **archivos cabeceras** (*header files*) cuyos nombres terminan en **.h**.
- Las funciones se agrupan por la similitud de tareas que realizan y cada grupo se declara en un único archivo de cabecera.
- Se pueden incluir (**#include**) tantos *.h como sean necesarios.
- Algunas de los servicios proporcionados por estas bibliotecas son:
  - **stdio.h** = entrada y salida de datos.
  - **string.h** = manejo de cadenas de caracteres.
  - **math.h** = principales funciones matemáticas.
  - **stdlib.h** = gestión de memoria dinámica y comunicación con el entorno.

LENGUAJE C: Librería math.h

- Define las principales constantes redondeadas a 21 dígitos decimales.

```
#define  M_E        2.71828182845904523536
#define  M_PI       3.14159265358979323846
#define  M_SQRT2    1.41421356237309504880
```

- Funciones trigonométricas: double **sin**(double x); double **asin** (double x);
- Funciones hiperbólicas: double **sinh**(double x); double **asinh**(double x);
- Funciones logarítmicas: double **log**(double x); double **log10**(double x);
- Valor absoluto: int **abs**(int x); long **labs**(long x); double **fabs**(double x);
- Potencia y raíz: double **pow**(double x, double y); double **sqrt**(double x);
- Redondeo por exceso y defecto: double **ceil**(double x); double **floor**(double x);
- Resto de la división: double **fmod**(double x, double y);
- Conversión a string: void **ftoa**(float numero, char *vector, int decimales);

LENGUAJE C: Librería stdio.h

- Define los tipos, macros y funciones necesarias para leer e imprimir valores.
- int **getchar**(void); (macro) devuelve el carácter introducido desde **stdin** (teclado).
- int **putchar**(int c); (macro) muestra en **stdout** (pantalla) el valor del parámetro.
- char **gets**(char *s); devuelve el string introducido por teclado (hasta '\n').
- int **puts**(**const** char *s); muestra en pantalla una cadena de caracteres + '\n'.
- int **printf**(**const** char *formato, ...); imprime en patalla de acuerdo a los formatos pasados como parámetros.
- int **scanf**(**const** char *formato, ...); lee desde el teclado los elementos del formato y los almacena en las direcciones determinadas por los siguientes parámetros.
- Funciones relacionadas con la E/S a ficheros.

LENGUAJE C: Función printf()

- Permite especificar el formato de la salida de datos (mensajes + valores). **Retorna** el número de bytes que se enviaron a stdout.
- Prototipo: int **printf**(const char ***format**, arg_1, arg_2, …., arg_n) ;
- **format**: es una cadena de caracteres que contiene la información sobre el formato de salida de arg_1 … arg_n. Cada grupo encabezado por '%' especifica un formato.

% [flags] [width] [.prec] [F|N|h|l|L] type

Ejemplo: char c = 'A'; int i = 59; float f = 378.123456789;

double d = 378.123456789; char s[20] = "Mi mensaje";

type	Formato de salida	Impresión en pantalla
c	char	("Salida = %c, %c",c,65) → Salida = A, A
d, i	signed int (decimal)	("%d, %i, %i",i,-i,c) → 59, -59, 65
u	unsigned int (decimal)	("%d, %u",-i,-i) → -59, 4294967237 (C2)
o	unsigned int (octal)	("%d, %o",i,i) → 59, 73 /* falta flag "%#o" */
x, X	unsigned int (hexadecimal)	("%d, %x, %X",i,i,i) → 59, 3b, 3B /* falta flag */

LENGUAJE C: función printf()

% [flags] [width] [.prec] [F|N|h|l|L] type

Ejemplo: char c = 'A'; int i = 59; float f = 378.123456789;

double d = 378.123456789; char s[20] = "Mi mensaje";

type	Formato de salida	Impresión en pantalla
f	floating point (decimal)	("%f, %f",f,d) → 378.123444, 378.123457
e, E	floating point (exp. e ó E)	("%e, %E",f,d) 3.781234e+002, 3.781235E+002
g, G	**e** o **f** depende del valor. Si *exp* es <(-4) o >[**.prec**] es **e**	("%g, %G, %g, %.2G", 1.2e-5, 1.23e-3, 1.23e3, 1.23e3) → 1.23e-005, 0.00123, 1230, 1.2E+003
s	char string ('\0' ó [**.prec**])	("%s, %.4s",s,s) → Mi mensaje, Mi m
%	el carácter %	("Descuento %d%%",10) → Descuento 10%
p	puntero	("i está en: %p",&i) → i está en: 0022:FEFC

- **width**: cantidad **mínima** de caracteres a imprimir. Si el valor es más corto, el resultado se completa con espacios en blanco (blank). Si el valor es más largo no se trunca. Si width = '*' → valor del argumento que precede al que se formatea.

LENGUAJE C: función printf()

% [flags] [width] [.prec] [F|N|h|l|L] type

Ejemplo: char c = 'A'; int i = 59; float f = 378.123456789;

double d = 378.123456789; char s[20] = "Mi mensaje";

Flag	Especificación	Impresión en pantalla
None	justifica derecha (completa con ' ' hasta [**width**])	("%d \n%10d",i,i) → |59 | 59 /*ancho 10*/
0	justifica derecha (completa con '0' hasta [**width**])	("%d \n%010d",i,i)→|59 |0000000059 /*ancho 10*/
-	justifica izquierda (agrega ' ' hasta [**width**])	("%dNuevo",i) → |59Nuevo ("%-10dNuevo",i) → |59 Nuevo
+	siempre el signo (+/-)	("%d, %d, %+d, %+d",3,-3,3,-3) 3, -3, +3, -3
blank	imprime '-' para números negativos, sino un ' '	("% d \n% d",-i,i) → |-59 | 59
#	type=**o** → octal (0ddd) type=**x**/**X** → hexa (0xddd)	("%d, %o %#o",i,i,i) → 59, 73, 073 ("%d, %x, %#X",i,i-3,i-3) → 59, 38, 0X38

LENGUAJE C: función printf()

% [flags] [width] [.prec] [F|N|h|l|L] type

Ejemplo: char c = 'A'; int i = 59; float f = 378.123456789;

double d = 378.123456789; char s[20] = "Mi mensaje";

- **.prec**: para enteros (d, i, o, u, x, X) especifica la cantidad **mínima** de dígitos que se imprimen. Si el valor es más corto, el resultado se completa con '0'. Si el valor es más largo no se trunca. Para flotantes (f, e, E) es el **máximo** número de dígitos significativos después del punto decimal. Para strings es el **máximo** de caracteres.

```
printf("%.4d",i); →                    |0059/
printf("%.1d",i); →                    |59/                  /* imprime 2 aunque prec = 1*/
printf("%4.2f /%4.2E/",f,f); →         |378.12 /3.78E+002/
printf("%3.1f /%3.1E/",f,f); →         |378.1 /3.8E+002/
printf("%10.3f /%20.10f/",f,f); →      |   378.123 /   3.7812344360E+002/
printf("%20.10E/",d); →                |   3.7812345679E+002/
```

LENGUAJE C: función printf()

% [flags] [width] [.prec] [F|N|h|l|L] type

Ejemplo: char c = 'A'; int i = 59; float f = 378.123456789;

double d = 378.123456789; char s[20] = "Mi mensaje";

printf("%2s/",i); → |Mi mensaje/

printf("%18s/",i); → | Mi mensaje/

printf("%18.5 s/",f); → | Mi me/ /* sólo 5 caracteres */

printf("%-18.5s /",f); → |Mi me /

- Modificadores:

F	arg es un far pointer
N	arg es un near pointer
h	d, i, o, u, x, X = arg es short int
l	d, i, o, u, x, X = arg es long int
l	e, E, f, g, G = arg es double (scanf)
L	e, E, f, g, G = arg es long double

Secuencias de escape:

\n	New Line = nueva línea
\r	Carry return = retorno de carro
\b	Backspace = retroceso
\' \"	Comillas simples y dobles
****	Backslash = barra invertida
\t	Tabulador

LENGUAJE C: función scanf()

- Permite leer valores numéricos, caracteres y strings desde stdin, asigna los valores a una **dirección**. **Retorna** el número de campos procesados correctamente.
- Prototipo: int **scanf**(const char ***format**, **&**arg_1, **&**arg_2, …., **&**arg_n) ;
- **format**: es una cadena de caracteres con el mismo significado que para printf. Como se pueden modificar luego los valores se deben utilizar sus **direcciones**.
- **%[ ]**: Permite especificar el conjunto de caracteres que se va a almacenar (en un string generalmente). La conversión finaliza cuando se encuentra un caracter que no corresponde al conjunto especificado. Por ejemplo:
  - %[0-9] = sólo los números de 0 a 9. Un carácter distinto frena la conversión.
  - %[AD-G34] = reconoce A, D~G, 3 ó 4.
  - %[^A-C] = No (^) A, B, C. Cualquiera de estos 3 caracteres frena la conversión.
  - Para considerar ‘]’, debe ser el primer caracter del conjunto, %[]A-C] ó %[^]A-C].
  - %[^]%^B-E-] = acepta todas las letras excepto "%", "^", "]", "B", "C", "D", "E” y "-”.
- ***%*type***: lee una entrada de tipo *type* pero no la almacena en ninguna variable.

LENGUAJE C: función scanf()

Ejemplo 1:

```
#include  <stdio.h>

int main()
{
   int valor;

   printf("Ingrese un valor entero: ");
   scanf("%d", &valor);        /* &valor es la dirección donde se almacena valor */
   printf("\nEntero = %d", valor);

   return 0;
}
```

```
Ingrese un valor entero: 3458<ENTER>
Entero = 3458
```

LENGUAJE C: función scanf()

Ejemplo 2:

```
#include  <stdio.h>
int main()
{
   int valor, cnt;
   char car;
   float fvalor;

   printf("Ingrese un entero <,> y un flotante ");
   cnt=scanf("%d%c %f", &valor, &car, &fvalor);          /* atención a los datos */
   printf("Entero = %d, flotante = %8.1f, retorno = %d\n",  valor,  fvalor,  cnt);
   scanf("%x", &valor);
   printf("Valor = %d, en hexadecimal = %x\n", valor,  valor);

   return 0;
}
```

```
Ingrese un entero <,> y un flotante: 1959, 256.387<ENTER>
Entero = 1959, flotante =    256.4 , retorno = 3   /* cuenta la ',' */
A3
Valor = 163, en hexadecimal = a3
```

LENGUAJE C: función scanf()

Ejemplo 3: strings /* toma como final un espacio, [**width**] o \n = <ENTER> */

```
    ⋮
  char car[60];
  printf("Ingrese un string: \n");
  scanf("%s", car);
  printf("Mensaje = %s\n", car);
  scanf("%s", car);
  printf("Mensaje = %s\n", car);
  scanf("%[^\n]", car);                          /* lee toda la línea */
  printf("Mensaje = %s\n", car);
  scanf("%[^A-C]",car);
  printf("Mensaje = %s\n",car);
    ⋮
```

```
Ingrese un string: modelo para mensaje con espacios<ENTER>
Mensaje = modelo
Mensaje = para                    /* no tiene en cuenta el ' ' antes de para */
Mensaje =  mensaje con espacios   /* tiene en cuenta el primer ' ' */
modelo_para_frenAr_con_A
Mensaje =  modelo_para_fren
```

LENGUAJE C: PASO de PARAMETROS

- Por **valor**: opción por defecto. No se pasa el valor del parámetro, sino una copia.

  Cualquier modificación que se realice en la función no afecta el valor original.

```
#include  <stdio.h>
void modificar(int vble);    /* declaración */
int main()
{
    int i = 1;
    printf("valor de i = %d antes de llamar a
            modificar", i);
    modificar(i);       /* invocación */
    printf("\nvalor de i = %d después de
            llamar  a modificar", i);
    return 0;
}
```

```
void modificar(int i)            /* definición */
{
    printf("\nvalor de i = %d en modificar",i);
    i = 9;
    printf("\nvalor de i modificado = %d en
            modificar",i);
}
```

Pantalla:
```
valor de i = 1 antes de llamar a modificar
valor de i = 1 en modificar
valor de i modificado = 9 en modificar
valor de i = 1 después de llamar a modificar
```

LENGUAJE C: PASO de PARAMETROS

- Por **referencia**: lo que se pasa no es el valor del parámetro sino la dirección en la que está almacenado, por lo que la función puede modificar su valor (ver **scanf**).

```
#include <stdio.h>
void modificar(int *vble);   /* declaración */
int main()
{
    int i = 1;
    printf("\nvalor de i = %d antes de llamar
            a modificar", i);
    modificar(&i);     /* invocación */
    printf("\nvalor de i = %d después de
            llamar a modificar", i);
    return 0;
}
```

```
void modificar(int *i)          /* definición */
{
    printf("\nvalor de i = %d en modificar",
            *i);
    *i=9;
    printf("\nvalor de i modificado = %d en
            modificar", *i);
}
```

Pantalla:
valor de i = 1 antes de llamar a modificar
valor de i = 1 en modificar
valor de i modificado = 9 en modificar
valor de i = 9 después de llamar a modificar

LENGUAJE C: ÁMBITO de las VARIABLES

- **Ámbito** (**alcance**): define desde dónde se puede acceder al valor de la variable.
- **Locales**: visibles sólo dentro del bloque (función) donde se declaran. Prioridad.
- **Globales**: se pueden acceder desde todas las funciones del programa. Reduce la legibilidad y dificulta su modificación y depuración. Se generan funciones no reutilizables. Solución: pasar información entre funciones mediante los parámetros.

```
#include  <stdio.h>
int val = 10;                          /* global , accesible por todas las funciones */
void imprime(void);
int main()
{
   int val = 17;                              /* local, sólo accesible dentro de main */
   printf("Valor en main = %d\n",  val);              Valor en main = 17
   imprime();
   return 0;
}
void imprime(void) {
   printf("Valor fuera de main = %d\n",  val);} Valor fuera de main = 10
```

LENGUAJE C: MODIFICADORES de VARIABLES

- Los modificadores determinan la forma de almacenamiento de la variable.
- **auto**: por defecto (son locales). Se crea en la pila del sistema (**stack**) cuando se invoca la función o cuando se ejecuta el código dentro del bloque, y se destruye cuando acaba la función o bloque. No son inicializadas por defecto.
- **extern**: (son globales). Variables con almacenamiento permanente. Una variable externa se define en un archivo como global y se la invoca desde otro archivo. Por defecto se inicializan con 0.

Archivo1 (*.c)

```
int x, z;     /* globales */
char c;
   ……….

int func1()
{
   x = 10;
   return 0;
}
```

Archivo2 (*.c)

```
void func2()
{
   extern int x, z;
   extern char c;
   ……….
   z = 2*x;
   c= 'P';
}
```

Las variables *x*, *z* y *c* se declaran en Archivo1 y se pueden acceder desde Archivo2 utilizando el modificador *extern*.

LENGUAJE C: MODIFICADORES de VARIABLES

- **static**: mantienen su valor dentro de su ámbito (bloque, función o archivo). Cuando se aplica a una variable local su valor se retiene entre sucesivas llamadas a la función (se inicializa sólo una vez, 0 por defecto). Si una variable global se declara estática, se considera local al archivo, no puede accederse desde otro archivo.

```
#include <stdio.h>
void cuenta();
int main()
{
    cuenta(); cuenta();
    cuenta(); cuenta();
    return 0;
}
```

```
void cuenta()
{
    static int cnt = 0;
    printf("Contador = %d; ", cnt++);
}
```

Pantalla: Contador = 0; Contador = 1; Contador = 2; Contador = 3

- **register**: el compilador trata de almacenar el valor en uno de los registros de la CPU, brindando mayor velocidad de acceso. Es útil cuando se usa su valor continuamente (contadores). Sólo se puede utilizar con los tipos int y char.

LENGUAJE C: MODIFICADORES de FUNCIONES

- **static**: la función sólo se puede acceder desde el archivo donde se declaró.
- **extern**: se define una función cuya declaración se encuentra en otro archivo que debe pertenecer al mismo proyecto (**project**). Las declaraciones de prototipos son externas por defecto.

main.c

```
#include  <stdio.h>

int val = 19;

void imprime(void);

int main()
{
    ……..
    imprime();
}
```

archivo2.c

```
#include  <stdio.h>

extern void imprime(void)   // extern:  opcional
{
    extern int val;
    …….
    printf("……",val);
}
```

- **inline**: la función es expandida íntegramente al ser invocada (se reemplaza la llamada por el código correspondiente), no hay un salto a la función. Incrementa la eficiencia, pero aumenta el tamaño del código. Puede tener variables locales.

LENGUAJE C: RECURSIVIDAD

- Las funciones en C son recursivas, pueden llamarse a sí mismas desde su propio cuerpo. Tienen que tener una condición de finalización, sino podría continuar infinitamente. Un ejemplo típico es el cálculo del factorial (n!) de un número entero, definido por n! = n * (n-1) * (n-2) * ... * 2 * 1.

```
#include  <stdio.h>

long factorial(int num);      /* declaración */

int main()
{
    int valor = 4;
    long resultado;
    resultado = factorial(valor);
    printf("El factorial de %d es %ld\n",
            valor, resultado);
    return 0;
}
```

```
long factorial(int num)       /* definición */
{
    long resultado;

    if(num == 1)          /* doble = */
        resultado = 1;    /* condición parada */
    else
        resultado = num*factorial(num-1);

    return resultado;
}
```

Pantalla: El factorial de 4 es 24

El factorial de 17 es -288588840 (< 0 !!???)