# *Guía muy básica sobre programación en C*

El lenguaje C es muy parecido a java, salvo por 2 cosas:

- No es orientado a objetos (no existen la instrucciones *class* ni *new*)
- Permite el uso de punteros

Las semejanzas y diferencias entre los lenguajes se tratan a continuación

## Declaración de variables

Las variables pueden ser de varios tipos: *int*, *long*, *double*, *float*, *char*, etc. En C no existe el tipo *String.*

Ejemplos:

```
int a;     //a es un número entero entre –32768 y 32767
long b;    //b es un número entero entre –8388608 y 8388607
float c; //c es un número real con pocos decimales
double d;  //d es un número real con muchos decimales
char e;    //e es un carácter

int arr[20]; //En C los arreglos se declaran así
```

Después de declarar las variables se les puede asignar valores y trabajar con ellas:

```
int m, n;   //m y n son números enteros
m=5;        //m vale 5
n=m+8;      //n vale 13
```

Además de los tipos de variables mencionados, C posee otro tipo de variable llamada puntero, que es una variable que almacena una dirección de memoria.

```
int *arr; //un puntero
```

Un puntero es lo mismo que un arreglo en java al que aún no se le ha asignado memoria con *new*.

Esto se puede resumir en una tabla:

| **Cosa a hacer** | **Java** | **C** |
|---|---|---|
| Crear un puntero | `int[] p;` | `int *p;` |
| Crear un arreglo | `int[] p=new int[20];` | `int p[20];` |

Se puede apreciar que la diferencia que existe entre un puntero y un arreglo es que el arreglo tiene una cantidad de memoria propia, mientras que el puntero no. A continuación se muestra un pequeño ejemplo:

| 25 | 352 | 1700 | 56 | 254 | 1945 | 2456 | 1214 | 7561 | 6457 |
|---|---|---|---|---|---|---|---|---|---|
| 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 |

Aquí se muestra una porción de la memoria. En diferentes posiciones de la memoria hay diferentes datos. El número que identifica a cada posición de memoria se denomina dirección de memoria (en este caso 56, 57, 58, ...). Mediante el operador * se puede modificar el contenido de una posición de memoria. Por ejemplo, si hacemos lo siguiente:

```
*58=200;
```

se logra lo siguiente:

| 25 | 352 | 200 | 56 | 254 | 1945 | 2456 | 1214 | 7561 | 6457 |
|---|---|---|---|---|---|---|---|---|---|
| 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 |

Esto mismo podríamos haberlo hecho usando un puntero, es decir, haciendo esto:

```
int *p;
p=58;
*p=200;
```

Se debe notar que al puntero se le asigna el valor 58, es decir, se le asigna una dirección de memoria, y luego se modifica el contenido almacenado en esa dirección. Es decir, los punteros almacenan direcciones de memoria. Además, los punteros pueden sumarse y restarse. Por ejemplo, si escribimos:

```
p=p+1;
```

se logrará que ahora p valga 59. El puntero se usa como un índice para acceder a la memoria, pero no posee una memoria reservada.

Un arreglo se diferencia por reservar memoria:

int a[5];

| 25 | 352 | 200 | 56 | 254 | 1945 | 2456 | 1214 | 7561 | 6457 |
|---|---|---|---|---|---|---|---|---|---|
| 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 |

Se creó el arreglo a[]: `a[0],a[1],a[2],a[3]` y `a[4]` y se reservó memoria para cada uno de los componentes del arreglo (la memoria sombreada).

Si escribimos:

```
p=a;
```

entonces p tomará el valor 59, que es la dirección de memoria en que comienza el arreglo a.

Por último, escribir `*a` y `a[0]` es equivalente, ya que ambos darán como resultado 56. Escribir `*(a+1)` y `a[1]` es equivalente, ya que ambos darán como resultado 254.

Además de trabajar con números, caracteres y punteros, C puede trabajar con strings, pero no del mismo modo que Java. Para C, los strings son arreglos de caracteres que terminan con un caracter ‘’ (el carácter cuyo código ASCII es 0). La siguiente línea definen un string:

```
char a[5]=”Hola”; // siempre se debe dejar 1 espacio más en el
                  // arreglo que el número de caracteres.
```

**NOTA:** En el lenguaje en que se programa la tarjeta DSP se define un tipo de datos llamado `__fixed__`. Este tipo de datos sirve para almacenar números decimales entre -1 y 1. El DSP puede trabajar muy rápidamente con este tipo de datos, por lo que se usarán para hacer la convolución.

```
double  a;    // a es un número real cualquiera
__fixed__ b;  // b es un número real entre –1 y 1
```

## Instrucciones de control

### for

La instrucción for se ocupa de la siguiente forma:

```
for (i=i0; i<=i1; i++)
{
      < instrucciones a repetir >
}
```

Las instrucciones dentro de los corchetes se repetirán varias veces. Inicialmente i tomará el valor i0. Cada vez que se repita el ciclo, i aumentará en 1. El último ciclo será cuando i valga i1.

### while

```
while(condición)
{
      <instrucciones>
}
```

Las instrucciones dentro de los corchetes se repetirán mientras la condición sea cierta. Para generar las condiciones, se pueden usar los símbolos > (mayor que) , < (menor que) , == (igual a) , != (distinto a) , || (“o” lógico), && (“y” lógico) y ~ (“no” lógico). Ej:

```
while(a>5 && b==3)
{
      z=z*2;
}
```

Aquí la condición es que “a sea mayor que 5” y “b sea igual a 3”.

Ejemplo de for:

```
int a[5];   // a es un arreglo con 5 elementos
int n=0;    //n es un nº entero. Su valor inicial es 0
a[0]=12;
```

```
a[1]=9;
a[2]=4;
a[3]=16;
a[4]=1;

for (i=0; i<=4; i++)
{
      n=n+a[i];
}
```

Este programa hace que n sea igual a la suma de todos los elementos del arreglo a[].

**Funciones para interacción con el usuario (no sirven en particular para el DSP)**

Las funciones más usadas son printf( ) y scanf( ). printf( ) se usa del siguiente modo:

```
printf(formato,variable1,variable2,...)
```

formato: es un string que contiene el texto que va a salir en pantalla. Para indicar en qué parte del texto se van a escribir las variables, éstas se reemplazan por %d (para variable int), %f (para variable float), %lf (para variable double), %c (para variable char) o %s (para strings). Los saltos de línea (return) se deben indicar cono "\n". Por ejemplo, para escribir "Números 2 y 3" se pueden usar las siguientes instrucciones:

```
int m;
int n;

m=2;
n=m+1;

printf("Números %d y %d \n",m,n);
```

scanf(formato,&variable1, &variable2, ...)

formato: es un string que indica qué tipos de variable se va a preguntar. Puede contener %d, %f, %lf, %c o %s. Por ejemplo, el siguiente código pregunta un número real.

```
float n=0;
printf(“Ingrese un número real:”);
scanf(“%f”,&n);
```

Para poder usar printf o scanf en el programa, se debe incluir al principio de éste:

#include <stdio.h>

que es lo mismo que “import java.io.*” en Java.

**Declaración de funciones**

En C se pueden crear funciones para luego utilizarlas. Una función se crea así:

```
tipo0 nombre(tipo1 argumento1, tipo2 argumento2, ...)
{
        tipo0 valor;
        <instrucciones>
        return valor;
}
```

La función puede recibir varios argumentos que pueden ser de diversos tipos. También puede devolver un valor de salida.

La primera función que se ejecuta al comenzar el programa es la función main( ), que debe devolver un número entero.

Como ejemplo, se muestra a continuación un programa que calcula el producto punto entre dos vectores. Para ello se crea una función que recibe 2 arreglos y un número n (el número de elementos de los arreglos) y calcula el producto entre los dos, considerándolos vectores.

```
#include <stdio.h> //para que busque printf() y scanf()

double punto(double *a, double *b, int n) //función producto punto
{
      double ret=0;
      int i;

      for (i=0; i<=n-1; i++)
      {
            ret=ret+a[i]*b[i];
      }
      return ret;
}
```

```
int main() // Aquí comienza el programa
{
      double a[3]; //a es un arreglo sin valor inicial
      double b[3]; //b es un arreglo sin valor inicial
      double res;  //res es una variable real sin valor inicial

      printf("Ingrese 3 nºs para el vector a:");
      scanf("%lf %lf %lf",&(a[0]),&(a[1]),&(a[2]));

      printf("Ingrese 3 nºs para el vector b:");
      scanf("%lf %lf %lf",&(b[0]),&(b[1]),&(b[2]));

      res=punto(a,b,3);

      printf("a.b=%lf\n",res);

      return 0;         //La función main() generalmente devuelve cero
}
```

A continuación se muestra la ejecución del programa:

```
Ingrese 3 nºs para el vector a:1 2 3
Ingrese 3 nºs para el vector b:4 5 6
a.b=32.000000
```