Convenciones de Programación en C

1. Introducción

C es un lenguaje de programación creado en 1972 por Dennis M. Ritchie en los Laboratorios *Bell* como evolución del anterior lenguaje *B*, a su vez basado en BCPL. C es apreciado por la eficiencia del código que produce y es el lenguaje de programación más popular para crear software de sistemas, aunque también se utiliza para crear aplicaciones.

2. Estructura de un Programa en C

```
/*  Comentarios de un parrafo completo comprendidos entre
/*.....*/, sirven para aclarar el programa o una parte del
programa */

// Comentarios de 1 sola línea

// Zona de archivos de cabecera de las librerías

#include <....... . h>   // h de Head

#include <....... . h>


// Zona de prototipos de funciones

int Potencia (int x,y)

// Zona de variables globales

int valor; float media_total;


void main (void) // Prog. ppal. típico {

    // llave de inicio del programa

    // código del programa

    .......

    .......

    .......

    // fin del programa

}

//  Desarrollo del código de las funciones anteriores
```

3. Declaración Constantes, Variables, Tipos y Funciones

Las constantes, variables, tipos, funciones y acciones usados en un programa deben ser expresivos, claros, definitivamente relevantes a la solución del problema, y legibles. Sus nombres deben ser significativos o mnemónicos. Aunque es posible escribir sus nombres de cualquier forma, es conveniente utilizar alguna convención.

Las acciones en C se pueden simular como un tipo especial de funciones que no retornan nada (ver punto 5.6):

Ejemplo	**Identificador definido, y su uso**
const	Declaración de Variable. Deben comenzar siempre con una letra minúscula. Este estilo es el que usaremos.
variableLarga	Declaración de Variable. Este estilo es el que usaremos cuando usamos más de una palabra.
variable_larga	Declaración de Variable. "_" separa las palabras del identificador. No lo usaremos.
Blanco	Declaración de Constante. Deben comenzar siempre con una letra mayúscula.
Tpersona	Declaración de un tipo. La declaración de tipos comienza siempre con la letra T.
Ppersona	Declaración de un puntero. La declaración de punteros comienza siempre con la letra P.
Sumar(a,b)	Declaración de una acción. Una acción comienza en mayúscula.
PotenciaDos(x)	Declaración de una función. Una función comienza en mayúscula.

Hay que destacar que cuando se usan varias palabras para formar un identificador, éstas deben ser muy legibles: **estoSiSeVeBien**, **peroestocasiquenoseentiende**. ¿O si?

Los únicos identificadores que comienzan con una minúscula son las variables; todos los demás identificadores que no pueden cambiar comienzan con una mayúscula: constantes, tipos, acciones, funciones, etc.

No se debe restringir el tamaño de un identificador, sino más bien debe escogerlo para que sea significativo. Sin embargo, ante la posibilidad de escoger entre uno largo y otro corto, deberá escogerse el corto si tiene la misma expresividad del largo. Si queremos denotar la altura de una persona, es mejor usar el identificador **alt** que **alturaPersona**, siempre y cuando **alt** sea suficientemente claro en el contexto de programación. En general, cuando se trata de varias palabras, es mejor pegar las palabras es el estilo que se está imponiendo como estilo general cuando se trata de dos o tres palabras.

Es importante utilizar constantes para evitar el uso de números en las expresiones, por ejemplo:

```
pago = total * 1.1 * 0.25;
```

```
pago = total * ImpVentas * propina;
```

La segunda expresión es más clara y más simple de modificar.

4. PALABRAS RESERVADAS

El lenguaje C posee un número reducido de palabras reservadas, tan solo 32, que define el standard ANSI-C. Las **palabras reservadas** deben escribirse en **minúscula**.

`auto`	`double`	`int`	`struct`
`break`	`else`	`long`	`switch`
`case`	`enum`	`register`	`typedef`
`char`	`extern`	`return`	`union`
`const`	`float`	`short`	`unsigned`
`continue`	`for`	`signed`	`void`
`default`	`goto`	`sizeof`	`volatile`
`do`	`if`	`static`	`while`

5. Indentación

La indentación es la posición en que comienza una línea de código en el renglón. Es un anglicismo de uso común en informática. Por indentación se entiende mover un bloque de texto hacia la derecha insertando espacios o tabuladores para separarlo del texto adyacente. La indentación se utiliza para mejorar la legibilidad del código, teniendo en cuenta que los compiladores o intérpretes raramente los consideran. La indentación es uno de los factores más importantes para lograr mejores programas. Al indentar un programa adecuadamente se logra destacar su estructura de control. Son frecuentes las discusiones sobre cómo o dónde usar la indentación, si es mejor usar espacios en blanco o tabuladores. Para unificar la forma de escribir los algoritmos usaremos un estilo en particular, aunque no es el único.

Debe indentarse **dos (2) espacios** o **un (1) TAB** cada vez que se utilice una construcción anidada. Siempre debe indentarse una construcción anidada, y siempre debe indentarse en **dos espacios** o un TAB. La consistencia en la aplicación de esta regla permite a nuestro cerebro predecir la forma del programa, y hacer la lectura más sencilla.

La **declaración** de variables, constantes, tipos y funciones debe indentarse **dos (2) espacios** o **un (1) TAB**. El cuerpo principal del programa debe indentarse **dos (2) espacios** o **un (1) TAB**.

Ejemplo:

```
#include <stdio.h>

/* Variables */
int  num;     //variable para almacenar el numero a analizar

void main(){
  printf("Ingrese un numero entero: ");
  scanf("%d",&num);
  if (num<0) {
    //num es negativo
    printf("%d es negativo \n", num);
  }
  else {
    //num es positivo
    printf("%d es positivo \n", num);
  }
  printf("Muchas gracias! \n");
}
```

5.1. Indentación del IF

La indentación de IF – THEN que se utilizará en los programas será:

```
if (num<0) {
  //num es negativo
  printf("%d es negativo \n", num);
}
```

La indentación de IF – THEN – ELSE que se utilizará en los programas será:

```
if (num<0) {  //num es negativo  printf("%d es negativo \n",
num);}else {  //num es positivo  printf("%d es positivo \n",
num);}
```

5.2. Indentación del WHILE

La indentación de WHILE que se utilizará en los programas será:

```
int i=1;

suma=0;

while (i<=100) {

   suma = suma + i;

   i++;

}
```

5.3. Indentación del REPEAT

La indentación de REPEAT que se utilizará en los programas será:

```
int i,j;

do {

  scanf(“%d”,&i);

  scanf(“%d”,&j);

} while (i<j);
```

5.4. Indentación del FOR

La indentación de FOR que se utilizará en los programas será:

```
for(x=1;x<=100;x++){
  printf(“Estoy dentro del for”);
}
```

5.5. Indentación de las funciones

```
int potencia(int x) {
```

```
  int aux;

  aux= x * x;

  return aux;

}
```

5.6. Funciones como acciones:

C no posee acciones pero se pueden simular como un tipo especial de funciones que no retornan nada:

`void potencia(int a)  // donde *void* (significa “nada”) es un tipo especial que indica que la función no retorna nada.`

Ejemplo:

```
void numero(int x) {
  if (x>=0){
    printf(“numero positivo”);
  }
  else {
    printf(“numero negativo”);
  }
}
```

6. Comentarios

Es difícil indicar exactamente cuándo incluir comentarios en el programa. Sin embargo, todos los comentarios deben cumplir con lo siguiente:

c1) Deben ser completos
c2) Deben ser válidos
c3) Deben ser pertinentes
c4) Deben ser claros
c5) Deben ser coherentes
c6) Deben ser concisos

En cuanto a la forma de los comentarios, es conveniente que estén alineados unos con otros. Si están dentro de un bloque, deben estar adecuadamente indentados, y siempre que sea posible debe dejarse un espacio separando a las barras \\ del comentario. Las barras que enmarcan a un comentario deben estar alineadas. Por ejemplo:

```
/*  Comentarios  de  un  parrafo  completo  comprendidos  entre
/*.....*/,  sirven  para  aclarar  el  programa  o  una  parte  del
programa */

int x;              // Comentarios de 1 sola línea

int potencia(int a) // función que retorna ...
```

Los comentarios deben explicar en relación al programa los siguientes asuntos:

- el cometido del programa. Esto debe reflejarse al comienzo del mismo.
- el papel de un identificador cuando no se explique por si mismo.
- en los subprogramas, si el parámetro es de entrada o de salida.

El cometido de un fragmento de programa se puede anticipar con un comentario:

```
{
  // lectura de datos
  instrucciones para la lectura de los datos

  //cálculos
  instrucciones para los cálculos

  //presentación de los resultados
   instrucciones de salida de resultados
}
```