Curso Básico de C++

IDE Visual C++ Express Edition 2008

(I)

Toma de contacto

Como ya os expliqué, actualmente se puede generar "código", es decir, "aplicaciones" informáticas, tanto para entornos gráficos como Windows de Microsoft, Apple, Android, etc, como para ventanas (lo que se conoce como modo "consola") en modo carácter.

Existen gran cantidad de lenguajes de desarrollo, algunos especializados otros de tipo educativo, pero hay uno, con el que se desarrollan o son base de ello, la mayoría de aplicaciones de tipo científico, técnico, robótica, control de procesos, etc.

El lenguaje C/C++, ideado por Kernighan y Ritchie en 1978, se ha convertido en el mas extendido, prácticamente común a cualquier plataforma gráfica (Sistema Operativo) o de consola, tanto, que incluso para el desarrollo de páginas web dinámicas, el famoso lenguaje Java, está basado en C++, vamos, que sabiendo uno, podremos controlar en poco tiempo el otro.

PROYECTO modo Consola

Un proyecto de consola C/C++, se compone inicialmente de un único fichero de texto, llamado "Código fuente", cuya extensión es **.cpp** y de forma opcional una serie de archivos también de texto formado por librerías de diseño propio (extensión **.h**) y de recursos (normalmente **.rm**).

Cuando se "compila" un código fuente se genera otro fichero llamado "código objeto", de extensión **.obj**, el cual es enlazado con las librerías precisas para generar el definitivo fichero "ejecutable" de extensión conocida **.EXE**.

Nomenclatura:
Cuando se exponga la sintaxis de cada sentencia se adoptarán ciertas reglas, que por lo que sé son de uso general en todas las publicaciones y ficheros de ayuda.

- ⊕ Los valores entre corchetes "[ ]" son opcionales, con una excepción: cuando aparezcan en negrita "[ ]", en ese caso indicarán que se deben escribir los corchetes.
- ⊕ El separador "|" delimita las distintas opciones que pueden elegirse.
- ⊕ Los valores entre "<>" se refieren a nombres. Los textos sin delimitadores son de aparición obligatoria.
- ⊕ C/C++, es sensible a mayúsculas/minúsculas, ¡Ojo!, no es lo mismo Void que void.

ESTRUCTURA

En C/C++, hay sólo dos zonas perfectamente delimitadas, las voy a llamar Afuera y Dentro.

Dentro, será siempre el código que introduciremos en una función/procedimiento. Una función, es un bloque de código que se ejecuta de forma secuencial (de arriba hacia abajo), salvo que se indique lo contrario.

- Un procedimiento, es aquel bloque que recibe cero o varios valores o parámetros, realiza una operación pero finalmente, NO devuelve ningún dato.
- Una función, también puede recibir valores, pero siempre devolverá un valor.

En C/C++, no se hace una real distinción entre uno y otro, simplemente, de forma general, una función tendrá la siguiente forma:

<tipo_dato_devuelto> **nombre_función** ([tipo_dato1 nombre_dato1, tipo_dato2 nombre_dato2,…]) {

[Líneas de código, a partir de ahora, SENTENCIAS, terminan en un ;]

}

Cuando no queremos devolver ningún valor, es decir, un procedimiento, el <tipo_dato_devuelto>, será **void**.

En todo "programa", debe de existir al menos, una función llamada **main.** En general, se describe:

```
void main () {

}
```

Entre las sentencias que usaremos, lo más probable es que precisemos de usar ciertas funciones ya creadas y guardadas en "librerías", para conseguir un programa más útil, hay que saber, y para ello usaremos siempre la ayuda del compilador (IDE), a que librería pertenece cada una, si sabemos a priori el nombre y utilidad de esta, en caso contrario, habrá que preguntar al que sabe (yo) o rebuscar en la susodicha "ayuda" o tirar de un manual en papel.

Cuando queramos usar una función de una librería específica, tendremos que describir dicho uso, siempre "**fuera**", y por norma, usando las primeras líneas del código fuente, de la siguiente manera:

```
#include <nombre_librería.h>
```

También, y como regla para nosotros, cada vez que creemos un nuevo código fuente, incluiremos las siguientes librerías de forma automática, pues son las más usadas.
`#include <stdio.h>`, *funciones estándar de entrada y salida a pantalla.*
`#include <conio.h>`, *funciones especiales de teclado.*

Tipos de datos

Existen una cantidad ingente de tipos de datos que podemos usar, pero de momento, tiraremos sólo de cuatro:

`int` , tipo numérico entero con o sin signo (negativo).

`double`, tipo numérico decimal, con o sin signo (negativo)

`char` , tipo de texto, se refiere a un solo carácter o letra, si quiero definir un grupo de letras juntas (palabra/frase), lo haremos de momento como `char *.`

`bool` , tipo lógico, 1 si cierto y 0 si falso.

Un tipo de dato como tal, no sirve de nada, este tipo, se deberá asignar o bien a una función como hemos visto antes, o bien a una "variable".

Conceptualmente, desde el punto de vista de un programador, una variable es una entidad cuyo valor puede cambiar a lo largo de la ejecución de un programa.

En el nivel más bajo, una variable se almacena en la memoria del ordenador. Esa memoria puede ser un conjunto de semiconductores dentro de un circuito integrado, ciertos campos magnéticos sobre una superficie de un disco, ciertas polarizaciones en una memoria de ferrita, o cualquier cosa que aún no se haya inventado. Afortunadamente, no deberemos preocuparnos por esos detalles.

¿Dónde y como defino variables?

En general, si defino una variable "**dentro**", es decir, dentro de una función, esta variable sólo es visible y utilizable dentro de dicha función; si quiero que una variable pueda ser usada por otras funciones, la definiremos "**fuera**", generalmente, justo después de las librerías y antes de la función **main**.

Para definir una variable, ponemos primero el tipo, y seguido de un espacio en blanco, el nombre que le queramos dar a esta, recordad que siempre hay que terminar una sentencia con "**;**".

Ej. `int` dividendo;

Si queremos definir varias variables del mismo tipo, lo podremos hacer usando la misma sentencia.

Ej: `int` dividendo, divisor;

Estas definiciones, las pondremos por "buena costumbre", justo después del nombre de la función y siempre antes de usarlas por primera vez.

Una variable, hay que "inicializarla" dándole un valor inicial, en caso contrario, el compilador dará un "aviso" de error, aunque finalmente nos deje continuar. Esto normalmente ocurre, para avisarnos

de esas variables que definimos y después NO usamos, y que ocupan espacio en el código y en la memoria del ordenador.

Un ejemplo de un fichero .cpp para un proyecto sería:

```
#include <stdio.h>
#include <conio.h>

void main(){
int i,tecla,result;
int numero,n;
double con_decimales;
char *texto;

i = 3;
tecla = 12;
result = 23;

texto = "Hola ¿Cómo estás?;

con_decimales = numero/n;
result = (tecla + i) * result;

}
```

Como podéis ver, los decimales se marcan con el punto "**.**", NO con la coma.

También nos daremos cuenta, que por defecto, Visual C++, no reconoce del español ni la ñ, ni acentos, ni siquiera los símbolos del teclado tales como interrogantes, etc.

Ya veremos más adelante como solucionar esto.

COMENTARIOS Es de buena costumbre también, comentar nuestro código, y cuanto más, mejor, esto hará que sea legible tanto por mí en el futuro como por otro programador que tenga que modificarlo.

Hay varias formas de comentar, en bloque o línea a línea:

```
// Línea comentada, no tiene que terminar en ";".

/* Bloque comentado, de aquí hasta que se encuentre el aste-
risco y la barra al revés. */
```

Si probamos el código anterior, Visual C++ no nos generará ningún error (creo), pero obviamente nos daremos cuenta de la futilidad de nuestro trabajo, ya que no solamente este código no hace ná de ná, si no que además, al "Depurarlo", es decir, "ejecutarlo", aparecerá una ventana (la consola de MS-DOS) que rápidamente desaparece y nuestro programa concluye sin pena ni gloria.

Esto es por que, tal y como hemos diseñado el código, se ejecutan las sentencias en su orden secuencial, no vemos nada en la consola por que en ningún momento e pedido que se haga tal cosa y al final, el Visual C++ decide cerrar la consola por que nadie le dijo tampoco que esperara.

¿Cómo evito que se cierre la consola?

La única forma, es realizando alguna "operación" que requiera que dicha consola se quede "esperando" una interactividad con el usuario, es decir, con nosotros, aunque esa interactividad no sirva para nada, al menos conseguiremos que la consola no finalice de golpe y me permita ver los resultados de mi código.

De la librería <conio.h>, disponemos de una función, que al ejecutarla, se queda esperando a que el usuario presione "cualquier tecla del teclado", para poder seguir con la ejecución de las sentencias que hayan después, si las hay. Esta función, devuelve como un valor de tipo `int`, el código ASCII de la tecla pulsada (ya veremos esto), como de momento me da igual la tecla pulsada, no hace falta recoger dicho valor en ninguna variable, simplemente ejecutamos la función dentro de una sentencia.

getch();

¿Qué cálculos matemáticos puedo realizar?

Muchos, básicamente podremos usar los "operadores matemáticos":

Existen dos operadores aritméticos unitarios, '+' y '-' que tienen la siguiente sintaxis:

```
+ <expresión>
- <expresión>
```

Asignan valores positivos o negativos a la expresión a la que se aplican.

En cuanto a los operadores binarios existen varios. '+', '-', '*' y '/', tienen un comportamiento análogo en cuanto a los operandos, ya que admiten tanto expresiones enteras, como con decimales. Sintaxis:

```
<expresión> + <expresión>
<expresión> - <expresión>
<expresión> * <expresión>
<expresión> / <expresión>
```

Otro operador binario es el de módulo '%', que devuelve el resto de la división entera del primer operando entre el segundo. Por esta razón no puede ser aplicado a operandos con decimales:

```
<expresión> % <expresión>
```

Nota: Esto quizás requiera una explicación:

Veamos, por ejemplo, la expresión 17 / 7, es decir 17 dividido entre 7. Cuando aprendimos a dividir, antes de que supiéramos sacar decimales, nos enseñaron que el resultado de esta operación era 2, y el resto 3, es decir 2*7+3 = 17.

En C++, cuando las expresiones que intervienen en una de estas operaciones sean enteras, el resultado también será entero, es decir, si 17 y 7 se almacenan en variables enteras, el resultado será entero, en este caso 2.

Por otro lado si las expresiones son en punto flotante, con decimales, el resultado será en decimal, es decir, 2.428571. En este caso: 7*2.428571=16.999997, o sea, que no hay resto, o es muy pequeño.

Por eso mismo, calcular el resto, usando el operador %, sólo tiene sentido si las expresiones implicadas son enteras, ya que en caso contrario se calcularán tantos decimales como permita la precisión de tipo utilizado.

Siguiendo con el ejemplo, la expresión 17 % 7 dará como resultado 3, que es el resto de la división entera de 17 dividido entre 7.