12. Manejo básico de pantalla en C.

Emulación de la librería Conio de Borland® para Dev-C++

Cuando se crean aplicaciones de consola usando compiladores para Windows, como Dev-C++, a menudo necesitamos compilar programas escritos originalmente para compiladores Borland®. O sencillamente, queremos dotar a nuestros programas de consola de una apariencia más amigable y agradable.

Es cierto que Dev-C++ incluye una emulación de **conio**, pero a nuestro juicio no es lo bastante fiel en cuanto a comportamiento a la original de Borland®. Sobre todo en funciones de ventanas, lecturas sin eco en pantalla, y en funciones como "`kbhit`", que o bien no se emulan, o se hace de forma incompleta.

Al contrario que sucede con los compiladores de Borland®, en el que la librería está incluida en el compilador, cuando se trabaja con otros compiladores es necesario incluir ***el archivo fuente conio.c como parte del proyecto***.

Instalación y configuración de la librería conio.h para Dev-Cpp de Windows

Si tienes instalado Dev-C++ sobre Windows debes seguir los siguientes pasos:

1. Descarga la librería **conio** de la siguiente dirección y grábala en cualquier directorio de tu PC, puedes descargarla de:
   http://c.conclase.net/devcpp/?cap=ejemplos#inicio

2. Descomprime este directorio y encontrarás los siguientes archivos:
   conio.h,
   conio.c
   libconio.a
   text.c
   conio-lib.dev
   conio-eje-a.dev
   conio-eje-c.dev
   Console_conio.template
   ConsoleConio_c.txt
   ConsoleConio_cpp.txt

3. Ahora, debes copiar los archivos "**conio.h**" y "**conio.c**" en el directorio "**C:\Dev-Cpp\include**". Si los archivos ya existen sobreescríbelos sin miedo!, suponiendo que tengas instalado Dev-C++ en el disco "C:".

4. Listo ahora ya puedes hacer programas que incluyan la librería `conio.h` con todas sus funciones para manejo de ventanas y colores. No olvides que

debes crear un proyecto y ***adicionar el archivo conio.c al proyecto*** para que se pueda compilar el programa.

EJEMPLO

A continuación se muestra un ejemplo muy sencillo del uso de la librería `conio`. Y utiliza algunas de las funciones conocidas como `window()`, `gotoxy()`, `textbackcolor()`, `textcolor()`, `clrscr()`, etc.

```
#include <cstdlib>
#include <iostream>
#include <conio.h>

void dibujarCuadro(int, int, int, int);

int main()
{
    int n;
    clrscr(); // Limpia TODA la pantalla

    // Crea una ventana COMPLETA de fondo azul claro
    window(1,1,80,25);
    textbackground(LIGHTBLUE); //fondo de la pantalla AZUL CLARO
    clrscr();

    // Ventana de fondo o sombra NEGRA
    window(4,4,79,24); //crea una ventana en estas coordenadas
    textbackground(BLACK);
    clrscr();

    // Ventana principal donde se solicitan datos
    window(2,2,78,23); //crea una ventana en estas coordenadas
    textbackground(GREEN); //fondo de la pantalla verde
    textcolor(YELLOW); //color de la letra amarillo
    clrscr();

    dibujarCuadro(1,1,76,22); //cuadro grande
    dibujarCuadro(3,3,74,5); //cuadro título

    gotoxy(16,4);
    textcolor(WHITE); //establece el color de texto en blanco
    cprintf("Programa que muestra los colores por numero");

    for (n = 0; n < 16; n++)
    {
        textcolor(n);
        gotoxy(5,6+n);
        cprintf("El %d corresponde al %c", n, 178);
    }
```

```
    cprintf("     Presione cualquier tecla para terminar...");
    getch();
    return 0;
}

void dibujarCuadro(int x1, int y1, int x2, int y2)
{
    int i;
    for (i=x1+1; i<x2; i++)
    {
        gotoxy(i,y1); cputs("-"); //línea horizontal superior
        gotoxy(i,y2); cprintf("-"); //línea horizontal inferior
    }

    for (i=y1+1;i<y2;i++)
    {
        gotoxy(x1,i); cprintf("|"); //línea vertical izquierda
        gotoxy(x2,i); cprintf("|"); //línea vertical derecha
    }
}
```

Algunas funciones útiles

Números aleatorios

En un programa de gestión o una utilidad que nos ayuda a administrar un sistema, no es habitual que podamos permitir que las cosas ocurran al azar. Pero los juegos se encuentran muchas veces entre los ejercicios de programación más completos, y para un juego sí suele ser conveniente que haya algo de azar, para que una partida no sea exactamente igual a la anterior.

Generar números al azar (“números aleatorios”) usando C no es difícil. Si nos ceñimos al estándar ANSI C, tenemos una función llamada “rand()”, que nos devuelve un número entero entre 0 y el valor más alto que ésta puede retornar, que de acuerdo a nuestro compilador es ***32767.*** La palabra reservada RAND_MAX regresa este valor máximo admitido por dicha función.

Generalmente, nos interesarán números mucho más pequeños (por ejemplo, del 1 al 100), por lo que “recortaremos” usando la operación módulo (“%”, el resto de la división).

Vamos a verlo con algún ejemplo:

Para obtener un número del 0 al 9 haríamos `x = rand() % 10;`
Para obtener un número del 0 al 29 haríamos `x = rand() % 30;`
Para obtener un número del 10 al 29 haríamos `x = rand() % 20 + 10;`
Para obtener un número del 1 al 100 haríamos `x = rand() % 100 + 1;`
Para obtener un número del 50 al 60 haríamos `x = rand() % 11 + 50;`
Para obtener un número del 101 al 199 haríamos `x = rand() % 100 + 101;`

Pero todavía nos queda un detalle para que los números aleatorios que obtengamos sean “razonables”: los números que genera un ordenador no son realmente al azar, sino “pseudo - aleatorios”, cada uno calculado a partir del siguiente. Es decir, sigue un algoritmo, que muestra el resultado de un conjunto de operaciones (“numero aleatorio”). Para obtener estos números el algoritmo de la función **`rand()`** parte de un “número semilla”, un número inicial al cual somete al mismo conjunto de operaciones una y otra vez**...** sin importar las veces que ejecutemos el programa.

Podemos elegir cual queremos que sea el primer número de esa serie (la “semilla”), pero si usamos uno prefijado, los números que se generarán serán siempre los mismos. Por eso, será conveniente que el primer número se base en el reloj interno del ordenador: como es casi imposible que el programa se ponga en marcha dos días exactamente a la misma hora (incluyendo milésimas de segundos), la serie de números al azar que obtengamos será distinta cada vez.

La “semilla” la indicamos con `srand()`, y si queremos basarnos en el reloj interno de la computadora, lo que haremos será `srand(time(0))`, antes de hacer ninguna llamada a `rand()`.

Para usar `rand()` y `srand()`, deberíamos añadir otro fichero a nuestra lista de “includes”, el llamado “stdlib”:

```
#include <stdlib.h>
```

Si además queremos que la semilla se tome a partir del reloj interno del ordenador (que es lo más razonable), deberemos incluir también “time”:

```
#include <time.h>
```

Vamos a ver un ejemplo, que muestre en pantalla un número al azar entre 1 y 10:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

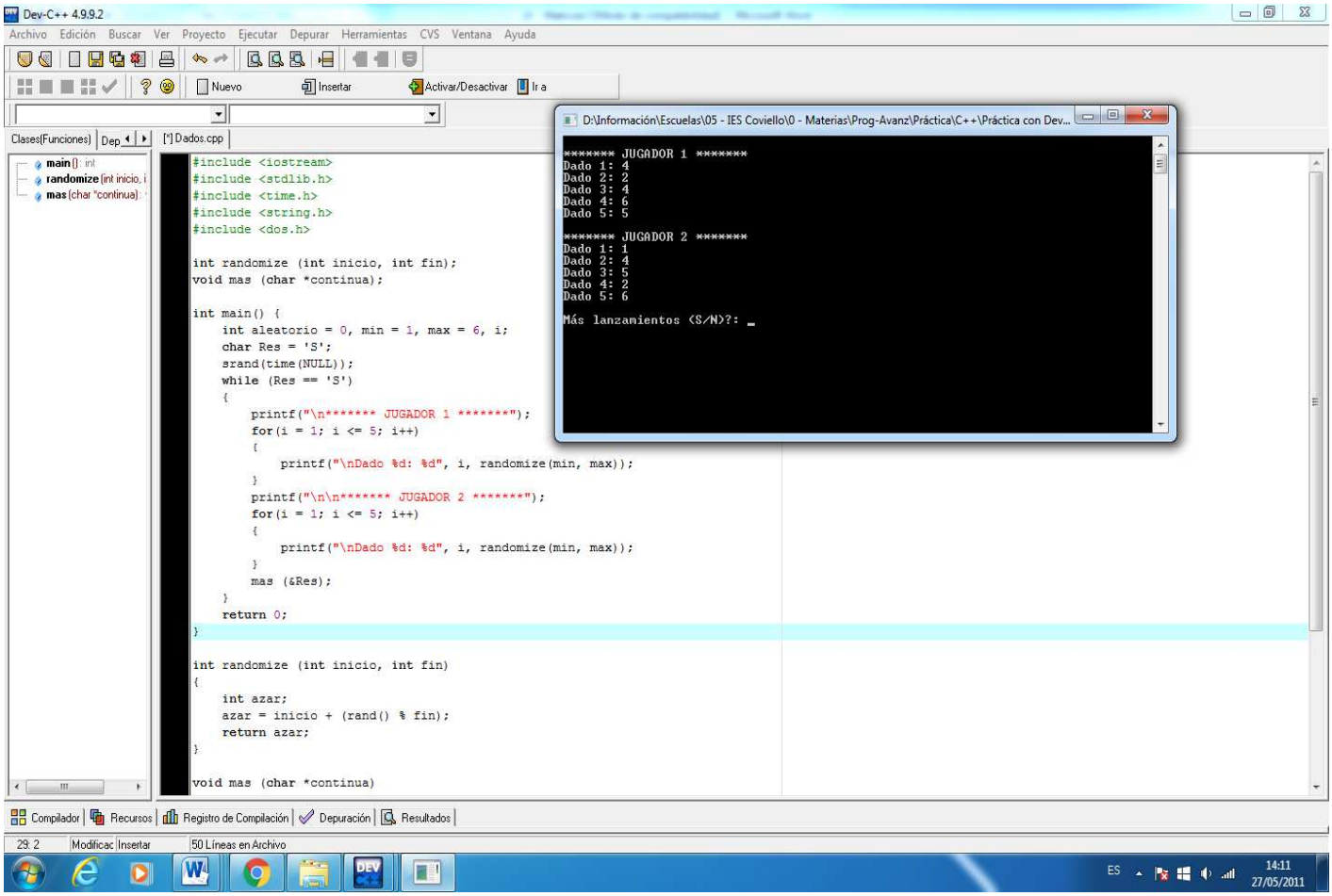
```
main()
{
     int n; srand(time(0));
     n = rand() % 10 + 1;
     printf("Un número entre 1 y 10: %d\n", n);
}
```

El siguiente ejemplo simula una tirada de cinco dados para cada uno de dos jugadores.

Ejemplo 2

```
#include <iostream>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <dos.h>

int randomize (int inicio, int fin);

void mas (char *continua);


int main()
{
     int aleatorio = 0, min = 1, max = 6, i;
     char Res = 'S';
     srand(time(NULL));
     while (Res == 'S')
     {
          printf("\n******* JUGADOR 1 *******");
          for(i = 1; i <= 5; i++)
          {
               printf("\nDado %d: %d", i, randomize(min, max));
          }

          printf("\n\n******* JUGADOR 2 *******");
          for(i = 1; i <= 5; i++)
          {
               printf("\nDado %d: %d", i, randomize(min, max));
          }
          mas (&Res);
     }
     return 0;
}
```

```
int randomize (int inicio, int fin)
{
     int azar;
     azar = inicio + (rand() % fin);
     return azar;
}

void mas (char *continua)
{
     char more;
     do
     {
          printf("\n\nM\xA0s lanzamientos (S/N)?: ");
          scanf(" %c", &more);
          more = toupper(more);
     }
     while (more != 'S' && more != 'N');
     *continua = more;
}
```



La función Sleep()

Esta función detiene el programa durante una cierta cantidad de tiempo en milisegundos. Vamos a tener que agregar la librería *windows.h* para poder usar la función **`Sleep()`** (*así, con mayúsculas*). Veamos un pequeño ejemplo:

```
#include <cstdlib>
#include <iostream>
#include <windows.h>

using namespace std;

int main()
{
    cout << "Hola" << endl;
    Sleep(5000);
    cout << "Hace cinco segundos te salude!" << endl;
    system("PAUSE");
}
```

Se muestra ‘Hola’, se pausa cinco segundos, y se muestra ‘Hace cinco segundos te saludé!’

APENDICE

Librería conio.h

clrscr

Función: Borra la pantalla.

Sintaxis: clrscr( );

clreol

Función: Borra desde la posición del cursor hasta el final de la línea.

Sintaxis: clreol( );

gotoxy

Función: Cambia la posición del cursor a las coordenadas indicadas.

Sintaxis: gotoxy(columna , fila);

textcolor

Función: Selecciona el color de texto (0 - 15).

Sintaxis: textcolor(color);

textbackground

Función: Selecciona el color de fondo (0 - 7).

Sintaxis: textbackground(color);

wherex

Función: Retorna la columna en la que se encuentra el cursor.

Sintaxis: col=wherex( );

wherey

Función: Retorna la fila en la que se encuentra el cursor.

Sintaxis: fila=wherey( );

getch

Función: Lee y retorna un único caracter introducido mediante el teclado por el usuario. No muestra el caracter por la pantalla.

Sintaxis: letra=getch( );

getche

Función: Lee y retorna un único caracter introducido mediante el teclado por el usuario. Muestra el caracter por la pantalla.

Sintaxis: letra=getche( );