Ejercicios semana del 11 al 15 de Mayo

DEPARTAMENTO DE TECNOLOGÍA

CURSO 4º ESO

Materia : Programación informática

GRUPO 4º A ESCUDERO MARTÍNEZ, Mariano

A partir del documento Word que se adjunta, utiliza Dev C++ para realizar y probar los programas. Se incluyen los ejemplos 1 y 2 correspondientes a la primera sesión y los ejemplos 3 y 4 de la 2ª. Se añaden las sesiones 3 y 4 y hay que realizar los ejemplos vacíos utilizando los ejemplos de al lado como muestra. Entregad por el correo electrónico de la asignatura (iesizprog4ace@gmail.com)

GRUPO 4º C ESCUDERO MARTÍNEZ, Mariano

La semana pasada no salieron las modificaciones realizadas como no sé si entendisteis que era igual para los 3 grupos os adjunto tambien lo de la semana pasada y el archivo completo con las 4 sesiones.

A partir del documento Word que se adjunta, instalar en el ordenador el programa que aparece en el texto Dev C++. Seguir las instrucciones para probar los ejemplos 1 y 2 correspondientes a la primera sesión. Después intentar realizar los programas que se piden en los ejemplos 3 y 4.

A partir del documento Word que se adjunta, utiliza Dev C++ para realizar y probar los programas. Se incluyen los ejemplos 1 y 2 correspondientes a la primera sesión y los ejemplos 3 y 4 de la 2ª. Se añaden las sesiones 3 y 4 y hay que realizar los ejemplos vacíos utilizando los ejemplos de al lado como muestra. Entregad por el correo electrónico de la asignatura (iesizprog4ace@gmail.com)

GRUPO 4º E ESCUDERO MARTÍNEZ, Mariano

La semana pasada no salieron las modificaciones realizadas como no sé si entendisteis que era igual para los 3 grupos os adjunto tambien lo de la semana pasada y el archivo completo con las 4 sesiones.

A partir del documento Word que se adjunta, instalar en el ordenador el programa que aparece en el texto Dev C++. Seguir las instrucciones para probar los ejemplos 1 y 2 correspondientes a la primera sesión. Después intentar realizar los programas que se piden en los ejemplos 3 y 4.

A partir del documento Word que se adjunta, utiliza Dev C++ para realizar y probar los programas. Se incluyen los ejemplos 1 y 2 correspondientes a la primera sesión y los ejemplos 3 y 4 de la 2ª. Se añaden las sesiones 3 y 4 y hay que realizar los ejemplos vacíos utilizando los ejemplos de al lado como muestra. Entregad por el correo electrónico de la asignatura (iesizprog4ace@gmail.com)

INTRODUCCIÓN A LA PROGRAMACIÓN EN LENGUAJE C++.

Sesión 1.

Se parte de una situación en la que el alumno está familiarizado con el ordenador personal, es decir sabe como entrar al sistema y llegar al escritorio.

Definición de lenguaje de programación e introducción al lenguaje C: Un lenguaje de programación es un lenguaje artificial que se utiliza para el control del comportamiento de una máquina, sistema u ordenador. Existen varios tipos de lenguajes como pueden ser Java, Pascal, C, etc.

Lenguaje C

El lenguaje C es un lenguaje de programación de propósito general que tiene una sintáctica sencilla, control de flujo y estructuras sencillas y un buen conjunto de operadores. No es un lenguaje de muy alto nivel, es sencillo y no está especializado en ningún tipo de aplicación concreta.

Definición de IDE: Un entorno de desarrollo integrado o **IDE** (acrónimo en inglés de integrated development environment), es un entorno que consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). En el mercado existe gran variedad de IDE y los hay para diferentes tipos de lenguaje.
El IDE que se utilizará en la asignatura se llama **Dev-C++** y puede ser descargado desde:

https://sourceforge.net/projects/orwelldevcpp/postdownload

Se trata de un IDE gratuito con una presentación en pantalla sencilla. La ventaja de este entorno frente a otros es que presenta el lenguaje español.

Primeros pasos (toma de contacto con el IDE):
Una vez instalado, desde el escritorio del ordenador se accede a Dev-C++ mediante el icono:

Aparecerá en pantalla la consola principal del IDE Dev-C++ (figura1). Para introducir el primer programa se debe pulsar en "Archivo" en la barra de opciones y seguidamente aparecerá una nueva pantalla con las pestañas "**nuevo"** y "**código Fuente"** (figura 2).

Conceptos previos al primer ejercicio.

Un programa en C es simplemente un fichero de caracteres que contiene un conjunto de instrucciones que un programa especial, el **compilador o traductor**, es capaz de entender.

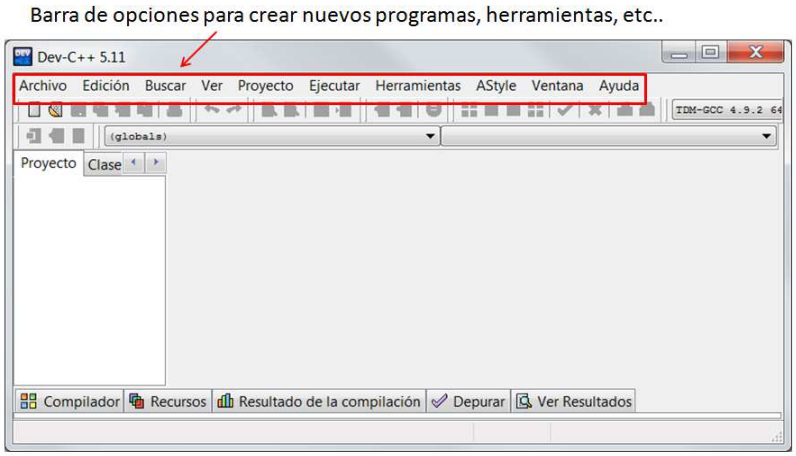

Figura 1. Pantalla principal del entorno IDE: Dev-C++.

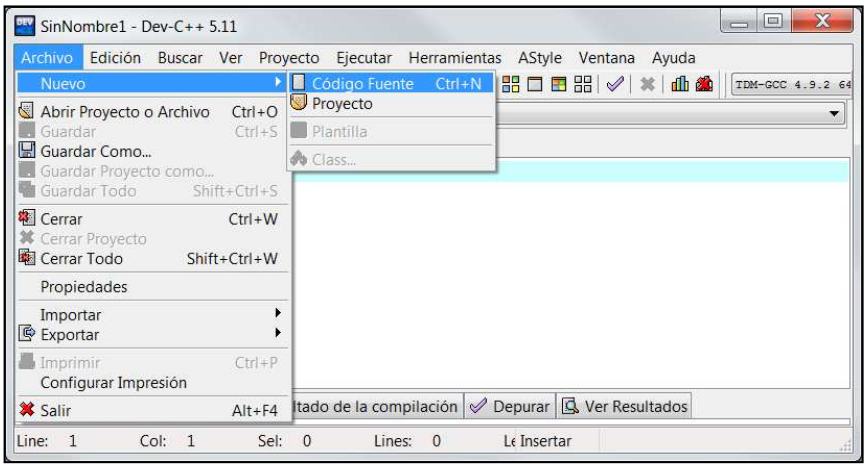

Figura 2. Creación de un nuevo programa C.

El compilador es un programa que transforma el programa escrito en C a otro lenguaje que puede ser interpretado por el ordenador. Revisa la sintaxis de las instrucciones y se encarga de transformar en un código que la computadora puede ejecutar. La estructura más básica de un programa en C es como se puede ver en la tabla 1.

Tabla 1. Estructura básica de un programa en lenguaje C.

1.- Inclusión de librerías como por ejemplo: #include<stdio.h>
2.- Se declara la función "**main",** esta función es la llamada por el Sistema Operativo. Toda función va en cerrada entre paréntesis **{ }**. Dentro del las llaves se declaran todas las instrucciones a ejecutar. Además las funciones pueden ser de diferentes tipos según el valor que devuelven. Se declara la función como **void** si no devuelve ningún valor o **int** si debe devolver algún valor entero.
3.- La segunda sentencia a definir es **return 0** y se declara dentro de las llaves de la función **main**. Esta sentencia significa que el programa devuelve un valor al sistema operativo, es decir, ejecución correcta del programa.

El siguiente paso es que el alumno escriba el programa de la figura 3 en el IDE, lo compile, ejecute y vea el resultado según aparece en la figura 4.

Una vez vista y entendida le ejecución el siguiente paso es ampliar el repertorio de instrucciones básicas para esta primera sesión.

Secuencias de escape.

Es importante conocer cómo hacer saltos de línea, tabulaciones, etc. Para esto se utilizan unos caracteres especiales, que son caracteres normales precedidos del carácter de diagonal invertida \(generalmente se obtiene pulsando AltGr + tecla de arriba a la izquierda). Estos caracteres especiales se llaman **secuencias de escape**, se pueden incluir en cualquier parte del texto, y son los que aparecen en la tabla 3.

```
#include<stdio.h>
void  main()
{
printf("Esta es una prueba de programación");
return 0;
}
```

Figura 3. Programa inicial en C.

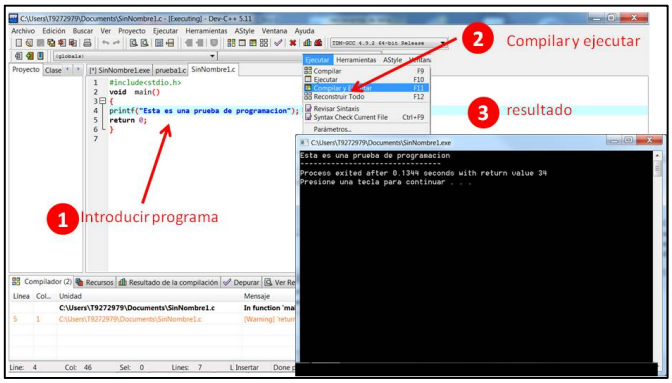

Figura 4. Introducción de programa, compilación y ejecución.

La función printf.

Mediante la función **printf** se puede escribir datos en el dispositivo de salida estándar (pantalla)**, printf** puede escribir cualquier combinación de valores numéricos, caracteres sueltos y cadenas de caracteres. Tiene la estructura siguiente:

printf (cadena de control, lista de argumentos);

Cadena de control hace referencia a una cadena de caracteres que contiene información sobre el formato de la salida y además se incluyen secuencias de escape.

Lista de argumentos son argumentos que representan los datos de salida. En la cadena de control se incluyen grupos individuales de caracteres cada uno de los cuales deben comenzar por **%.** Normalmente, un grupo de caracteres estará formado por el signo de porcentaje seguido de un ***carácter de conversión*** que indica el tipo del dato correspondiente (debe haber igual número de caracteres de conversión como de argumentos).

Detalle de cadena de control: %[- |+][número][.precisión][c|d|…].

La tabla 2 muestra la interpretación de los controles de la cadena de control, la tabla 3 muestra las secuencias de escape y la tabla 4 muestra los caracteres de conversión más habituales.

Tabla 2. Interpretación de los elementos de la cadena de control.

-	El valor se ajusta a la izquierda y el resto de rellena de blancos.
+	Un signo precede a cada valor numérico con signo
número	Indica la longitud mínima del campo
precisión	Número de dígitos decimales
c|d…	Carácter de conversión.

Tabla 3. Secuencias de escape

\a suena la alarma	\t tabulador horizontal	\n nueva línea	\' apóstrofe
\b retroceso	\\ diagonal invertida	\r regreso de carro	\" comillas

Tabla 4. Caracteres de conversión más habituales de la función printf.

Carácter	Significado
c	Escribe un carácter simple
d	Escribe un entero decimal con signo
i	Escribe un entero decimal, octal o hexadecimal con signo
e	Escribe un número en punto flotante con exponente
f	Escribe un número en punto flotante sin exponente

El ejemplo 1 en la siguiente página muestra por pantalla el número entero 888 mediante el carácter de conversión "d". El ejemplo 2 muestra por pantalla el número decimal 321.43 que contiene dos posiciones decimales mediante el carácter de control "f".

Comentarios.

Los comentarios en los programas son necesarios pues indican y aclaran los pasos que sigue el programador para escribir el programa. Son muy útiles para quien necesite modificar posteriormente el programa. Puede haber dos tipos de comentarios, de línea y de bloque.

Los comentarios de línea solo ocupan una línea en el programa y comienzan por el carácter "//". Los comentarios de bloque pueden ocupar varias líneas y comienzan por carácter"/*" y terminan por carácter"*/". En los ejemplos 1 y 2 que siguen a continuación se pueden observar ejemplos de comentarios.

Ejemplo 1	Ejemplo 2

#include<stdio.h>
int main()
{ // este es un comentario de línea
printf("%d",888);
return 0;
}
	#include<stdio.h>
	int main()
	{ /* Este es un comentario de bloque y se utiliza para describir un paso con más detalle*/
	printf("%3.2f",321.43);
	return 0;
	}

Sesión 2.

En esta sesión los contenidos a impartir serán tipos de datos y operadores de asignación. Los ejercicios a realizar en esta sesión deben tener en cuenta todo lo visto en la sesión anterior y en la actual.

Variables, tipos de datos.

Los programas necesitan almacenar información y para ello necesita las variables. Una variable es un lugar de la memoria donde se almacena un valor y puede recuperarse. También en esta posición de memoria se puede modificar el valor y volver a almacenarlo. Estas variables o posiciones de memoria pueden ser de diferentes tipos. Para tener una variable en un programa, previamente hay que definirla con un tipo (ver tabla 5) y un identificador (es decir, un nombre para la variable). Debe estar definida dentro de la función **main**()

Por ejemplo**: int** número=24;

En este caso define una variable de tipo entero que se llama "número" y seguidamente hay que añadir un punto y coma a la expresión: int número;

Por lo tanto, la fórmula general es: **<tipo> <identificador> <;>**

C ofrece tres tipos de datos básicos:

- Números enteros definidos con la palabra clave **`int`**
- Letras o caracteres definidos con la palabra clave **`char`**
- Números reales o en coma flotante definidos con las palabras claves **`float`** o **`double`**.

Enteros: se definen con **"int"** y admiten de forma opcional dos prefijos modificadores:

- **"short"** y **"long"**: Modifica el tamaño en bits del entero. Existen por tanto tres tipos de enteros: "**int**", "**short int**" (que se puede abreviar como "**short**"), y "**long int"** (que se puede abreviar como "**long**").

  El lenguaje C no define tamaños fijos para sus tipos de datos básicos. Lo único que garantiza es que un short int tiene un tamaño menor o igual que un int y este a su vez un tamaño menor o igual a un long int. Esta característica del lenguaje ha complicado la creación de programas que sean compatibles entre varias plataformas.
- "**unsigned**": define un número natural (mayor o igual a cero).

Tabla 5. Tipos de datos enteros y doble precisión.

Tipo de dato	Palabra reservada	Ejemplo
Entero	Int	Int número=0;
Real	Float	float número=12.2;
Flotante doble	double	double númeroDoble=1,76E200

Operadores de asignación.

Si se dispone de dos variables llamadas v1 y v2. La tabla 6 muestra los operadores de asignación y el efecto que cada unos de ellos produce en el valor de las variables.

Tabla 6. Operadores de asignación más habituales.

Operador/nombre		Efecto producido		Operador/nombre		Efecto producido
=	Asignación	v1 recibe valor de v2		%=	Asigna resto (módulo)	v1 se asigna resto de división por v2
*=	Asigna producto	v1 se multiplica por v2		+=	Asigna suma	v1 se suma con v2
/=	Asigna división	v1 se divide por v2		-=	Asigna diferencia (resta)	v1 se resta con v2

Los ejemplos 3 y 4 muestran la implementación práctica de la materia de esta sesión.

Ejemplo 3. Se declaran dos variables enteras y a la variable x se le asigna el valor de v multiplicado por el valor de x. Se muestra el resultado con el formato correcto	**Ejemplo 4.** Se declaran dos variables x e y con decimales. A x se le resta el valor de v y se asigna ese valor a x. Se muestra el resultado por pantalla con formato correcto

Sesión 3

En esta sesión los contenidos a impartir serán los operadores aritméticos y las constantes. Los ejercicios a realizar en esta sesión deben tener en cuenta toda la materia estudiada anteriormente.

Operadores aritméticos

Se llaman operadores aritméticos a aquellos que permiten realizar cálculos con valores numéricos para obtener un resultado. Los operadores aritméticos más habituales son la suma, resta, multiplicación y división. En C y en otros lenguajes se dispone de un operador adicional al que se denomina operador módulo (%), que permite obtener el resto de una división entre enteros. La tabla 14 muestra los operadores más habituales. Toda operación debe ir entre paréntesis ().

Tabla 7. Operadores aritméticos.

Operador	Significado		Operador	Significado
=	Asignación		%	Resto de división entera (mod)
*	Multiplicación		+	Suma
/	División		-	Resta

Constantes

En lenguaje C, una constante puede ser de tipo entero, real, carácter, de cadena o enumerado. Las contantes de tipo enumerado se estudiarán en sesiones posteriores. Para expresar una constante con un nombre, la constante debe ser declarada previamente. Las constantes que se declaran en un programa escrito en lenguaje C reciben un tratamiento diferente al de la mayoría de los lenguajes de programación. En C, para representar a las constantes, se utilizan *constantes simbólicas*. Una *constante simbólica* representa (sustituye) a una secuencia de caracteres, en lugar de representar a un valor (dato almacenado en memoria).

La directiva **#define** indica al preprocesador que debe sustituir, en el código fuente del programa, todas las ocurrencias del <nombre_de_la_constante> por la <secuencia_de_caracteres>, antes de la compilación. La directiva **#define** se declara en el programa justo después de la directiva **#include.**

Ejemplo: Dos constantes muy habituales son:

#define PI 3.141592

#define NÚMERO_E 2.718281

También es posible definir las constantes de esta forma: #define PI, NÚMERO_E, es decir unas seguidas de otras. Por otra parte, en C, es posible declarar una variable indicando que su valor es inalterable. Para ello, se utiliza el *calificador* **const**.

Ejemplo: Uso de **const**: const int temperatura = -5;

En el ejemplo 5 que sigue a continuación se puede observar la implementación de lo aprendido en esta sesión.

Ejemplo 5. En este ejemplo se define dos constantes radio y PI. El objetivo es mediante esas constantes calcular la longitud de una circunferencia de radio RADIO y mostrar el resultado correctamente por pantalla.

Sesión 4.

En esta sesión los contenidos a impartir serán los operadores relacionales y las estructuras iterativas donde se mostrará la ejecución de bucles **for.**

Operadores relacionales.

Se llaman operadores relacionales o de comparación a aquellos que permiten comparar dos valores evaluando si se relacionan cumpliendo el ser menor uno que otro, mayor uno que otro, igual uno que otro, etc. Es importante tener en cuenta que para comparar si una variable A es igual a otra debemos usar A == B en lugar de A = B. El doble signo igual se usa en comparaciones mientras

que el signo igual sencillo se usa en asignaciones. Para determinar si una variable A tiene distinto contenido que una variable B debemos usar A != B. En C no se admite la sintaxis A <> B que se suele usar en pseudocódigo. La tabla 15 muestra los operadores relacionales.

Tabla 8. Operadores relacionales.

Operador	Significado		Operador	Significado
<	Menor que		>=	Mayor o igual que
<=	Menor o igual que		==	Igual a
>	Mayor que		!=	Distinto de ó no igual que

Ejemplo 6. Sean las variables A=4 y B=6, las siguientes sentencias darían como resultado 0 (Falso). B<A, B==A, B<=A

Ejemplo 7. Sean las variables A=4 y B=6, las siguientes sentencias darían como resultado 1 (Verdadero). A!=B, A<B, A<=B

Estructuras iterativas "for".

Los bucles permiten ejecutar instrucciones un número determinado de veces. Para ejecutar un bloque de instrucciones un número determinado de veces se utiliza la sentencia **for.** La estructura de esta sentencia es la que sigue:

```
for (<valor_inicial>; <condicion>; <incremento>)
        {
        //Instrucciones a ejecutar "x" número de veces
        }
```

En **<valor_inicial>** se le asigna un valor a una variable ya creada. Se debe asignar un número por el que se desea que comience a contar el bucle.

En **<condición>** es la regla que se debe cumplir para que el bucle termine su ejecución.

En **<incremento>** es la rapidez con la que se desea que se incremente la variable que controla el bucle para que se cumpla la regla.

Los ejemplos 8, 9, 10 y 11 muestran la materia de esta sesión. Mirar los comentarios.

Ejemplo 8. El objetivo del programa es mostrar en pantalla los cien primeros números enteros en una línea diferente.	**Ejemplo 9.** El objetivo del programa es mostrar en pantalla los cien primeros números pares en una línea diferente.

#include <stdio.h>
void main()
{
int x;// variable para control bucle
for (x=1;x<100;x=x+1) // el bucle se
{ //repite mientras x sea menor de 100	

printf("%d\n",x); //imprime valor
}
return 0;
}	

Ejemplo 10. El objetivo del programa es mostrar en pantalla la suma de los cien primeros números enteros..	**Ejemplo 11**. El objetivo del programa es mostrar en pantalla la suma de los cien primeros números enteros y que en cada línea diga cuantos números hay sumados.

#include <stdio.h>
void main()
{// define variables
int x, suma;
suma=0;
for (x=1;x<100;x=x+1)
{// cálculo operaciones
suma=suma+x;
printf("%d\n",suma);
}
return 0;
}	