- **Objetivos de la práctica:**.
- Introducción al entorno de trabajo Dev C++: edición y compilación de programas en C++:
- Uso básico del depurador

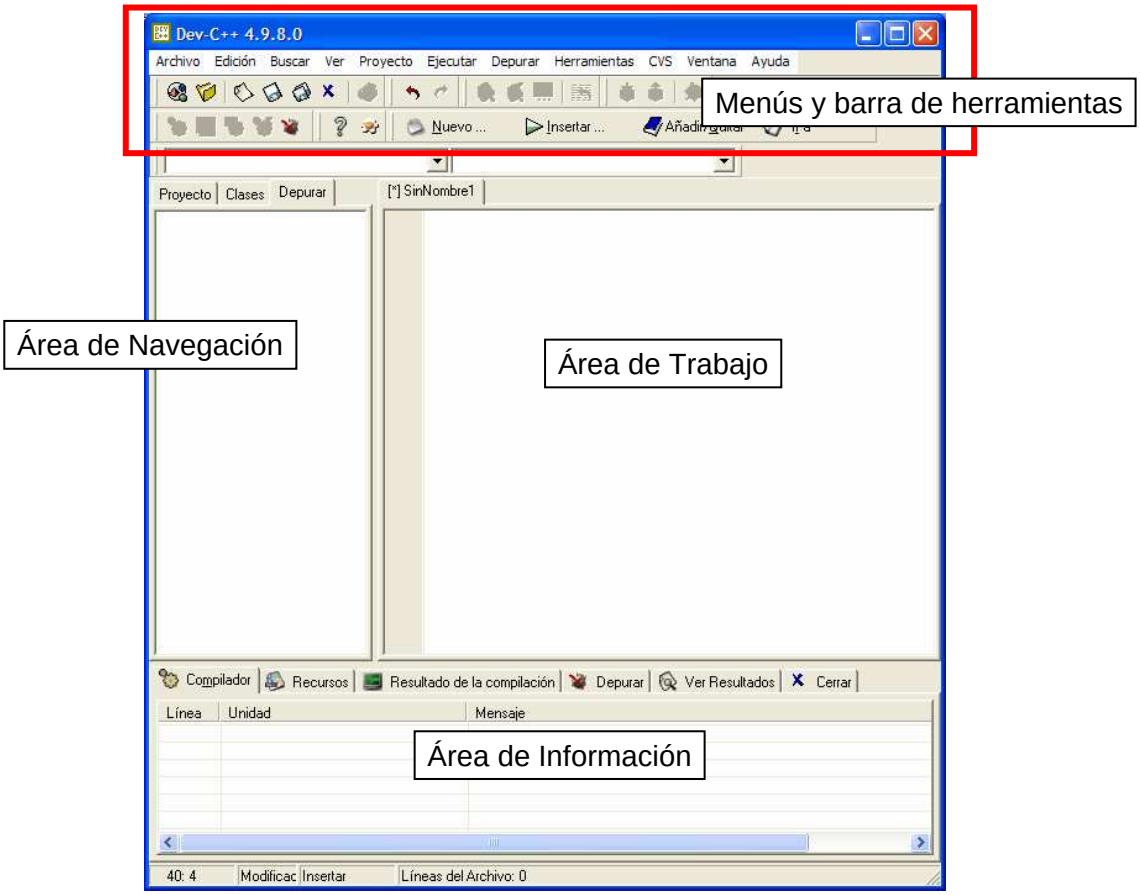
1.- Descripción de las 4 zonas en el entorno:

1.1.- Menú y barra de herramientas: todas las opciones están en el menú, la barra de herramientas contiene accesos rápidos (atajos) a las funciones.

1.2.- Área de navegación (`proyecto, clases, depurar`).

1.3.- Área de trabajo (edición de los archivos abiertos)

1.4.- Área de información (`compilador, recursos, resultado de la compilación, depurar, ver resultados`)



2.- Trabajar con un programa en C++ de ejemplo:

Descargar de la página web (`pizarra.uv.es`) todos los archivos de ejemplo y grabarlos en un directorio.

2.1.- Cargar el archivo, menú "`Archivo`", opción "`Abrir proyecto o archivo`": abrir el archivo "`ejemplo1_Practica1.cpp`".

2.2.- Estructura de programa en C++. Un programa consta de:

a) Inclusión de archivos de cabecera necesarios para la utilización de determinadas funciones propias del lenguaje. P.ej., la entrada y salida con `<iostream>`

b) Una función llamada `main()`, que especifica que el conjunto de órdenes del programa propiamente dicho. La estructura típica de esta función es la siguiente:

- Declaración de la información a utilizar en el programa (constantes y variables)
- Entrada de datos necesarios para resolver el problema
- Procesamiento de la información
- Salida de resultados

Estructura de un programa en C/C++

Inclusión de archivos

Programa

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

using namespace std;

int main ()
{
    //DECLARACIÓN DE VARIABLES
    const int anyo_actual = 2005;
    string nombre;
    int anyo;
    int edad;

    //ENTRADA DE DATOS
    cout << "Indica tu nombre: ";
    getline(cin, nombre);
    cout << "Indica tu anyo de nacimiento: ";
    cin >> anyo;

    //PROCESAMIENTO DE INFORMACION
    edad = anyo_actual - anyo;

    //SALIDA DE RESULTADOS
    cout << "Tienes " << edad << " anyos." << endl;
    if (edad >= 18)
        cout << nombre << ", deberias disimular esas arrugas !!!" << endl;
    else
        cout << nombre << ", eso no se lo cree nadie !!!" << endl;

    system ("PAUSE");
    return 0;
}
```

Declaración de información

Entrada de datos

Procesamiento de información

Salida de resultados

2.3.- Compilar el programa: Menú "`Ejecutar`", opción "`Compilar`". Lo que se está haciendo es comprobar la corrección del código escrito y después traducir a código ejecutable. En el cuadro de dialogo de la compilación se muestran los siguientes aspectos: "`Status`" (fase del proceso en que se encuentra), "`Errors`" (número de errores detectados), "`Warnings`" (número de avisos generados). Un aviso indica, generalmente, un futuro error de funcionamiento.
Como el programa está bien escrito el proceso de compilación finaliza sin errores ni avisos, cerrar el diálogo.
Sólo es preciso compilar el programa cuando se haya modificado alguna parte del código desde la última vez que se compiló. Comprobar mediante el explorador de archivos que la compilación crea un archivo ejecutable de manera "permanente".

2.4.- Ejecutar (hacer funcionar) el programa: Menú "`Ejecutar`", opción "`Ejecutar`". Se abre una ventana donde se pone en funcionamiento el programa. Cuando el programa finaliza (en el ejemplo hace falta un "`enter`" para acabar) la ventana se cierra.

2.5.- La opción "`Ejecutar`">> "`Compilar y Ejecutar`" realiza los dos procesos anteriores en un único paso. Verificar que esto es así.

2.6.- Verificación de lo que significa compilar el programa y que el archivo que se ejecuta no es el texto que vemos en pantalla sino el resultado de la compilación.

a) Proceder a realizar una modificación en el código fuente. Por ejemplo, comentar las líneas que solicitan y leen el nombre (líneas 22 y 23).
b) A continuación ir directamente a ejecutar el programa (SIN COMPILAR). Se comprueba que a pesar de haber eliminado esas líneas el programa sigue pidiendo el nombre, puesto que se está ejecutando el resultado de la última compilación.
c) Ahora, compilar y a continuación ejecutar para comprobar que los cambios se han hecho efectivos.
d) Eliminar los comentarios de las líneas 22 y 23 y volver a comprobar que el funcionamiento es el inicial.

2.7.- Comprensión de los mensajes de error que genera el compilador:
a) Cerrar el archivo actual ("`Archivo`" >> "`Cerrar`")

b) Cargar el programa "`ejemplo2_Practica1.cpp`".

c) Compilar el programa. Ahora el diálogo de compilación no da opción a continuar, salta directamente al editor marcando la primera línea de error. Se debe aprender a reconocer esa marca y a recurrir a la zona de información (zona inferior del entorno) para identificar los mensajes que el compilador ha generado (pestañas "`Compilador`" y "`Resultado de la compilación`"). La pestaña "`Compilador`" permite saltar en la zona de edición a las líneas de error sin más que pinchar 2 veces encima del mensaje. La pestaña "`Resultado de la compilación`" da información más completa sobre cada uno de los errores.
El compilador indica el punto donde se da cuenta de que hay un error, lo cual no significa que el origen del error esté siempre en la línea que se indica. La causa de un error puede estar en alguna de las líneas anteriores a donde ha sido detectado. Hay que saber leer código e interpretar los errores.

d) Modificar el programa para corregir los errores y verificar el funcionamiento.

3.- Depurar un programa:

a) Cerrar el archivo actual ("`Archivo`" >> "`Cerrar`")

b) Cargar el programa "`ejemplo3_Practica1.cpp`".

c) Compilar y ejecutar el programa para comprobar su funcionamiento.

d) Para poder utilizar el depurador es necesario haber compilado previamente con la opción de generación de código de depuración activado. Por lo tanto, comprobar que en el menú "`Herramientas`">>"`Opciones del compilador`">>"`Configuración`">>"`Linker`" está activada (`Yes`) la opción "`Generar información de Debug`".
e) Entrar en la pestaña inferior "`Depurar`". Antes de empezar a depurar vamos a fijar puntos de ruptura (puntos en los que se interrumpir la ejecución del programa).
Ejemplo: Puntos de ruptura en las líneas 32 y 54, que corresponden a la primera condición sobre los coeficientes y a la primera condición para la salida. Se pueden establecer pulsando sobre la barra a la izquierda del código.
Pulsar "`Depurar`" para empezar la ejecución del programa depurando. La ejecución es normal en cuanto a la entrada de datos, que hay que introducir normalmente y se debe parar en el primer punto de ruptura.
Al parar, establecer la visualización de todas las variables, crear "watch" para: `a, b, c, discr, x1, x2, tipo`. Comprobar que en este momento, los únicos valores con cierto sentido son los de `a`, `b` y `c`, el resto de las variables tienen valores arbitrarios hasta que empieza el cálculo y se van modificando.
También existe la posibilidad de que se visualice automáticamente el contenido de una variable sin más que situar el cursor durante unos segundos sobre una aparición de la variable en el código fuente.
"`Avanzar paso a paso`", va ejecutando instrucción a instrucción el programa (se puede observar la traza real del programa).
"`Saltar paso`" continua la ejecución hasta el siguiente punto de ruptura.
Depurar una vez haciendo "`Avanzar paso a paso`" y otra vez "`Saltar paso`". No olvidaos de poner los watch en cada depuración.
Mediante el depurador se puede analizar paso a paso lo que hace el programa, lo que permite sistematizar la búsqueda de errores.

El depurador relaciona el código ejecutable con el código fuente y esto puede tener pequeños fallos. Es de una gran ayuda pero no es infalible.

4.- Tareas de programación.

Tarea 1: Cargar el archivo "`ejemplo4_Practica1.cpp`", corregir los errores de compilación que contiene y comprobar que funcione correctamente.

Tarea 2: Cargar el archivo "`ejemplo5_Practica1.cpp`" y, con ayuda del depurador, corregir los errores de ejecución que contiene y comprobar que funcione correctamente.

Tarea 3: Escribir un programa conversor de moneda (euros → pesetas). El programa debe solicitar como entrada un valor que represente una cantidad en euros y calcular y mostrar por pantalla su valor equivalente en pesetas (1 euro = 166.386 pesetas).