Práctica 0 (Repaso)
Estándar de programación y Diagramas de flujo

.:::1 Sesión:::.

a. Estándar de normalización

En este apartado se sugieren una serie de normas que ayudarán a que el código sea más fácil de leer y modificar.

I) Extensión de los Ficheros:

Extensión	Descripción
.c	Fichero escrito en C
.cpp .cc	Fichero de C++
.h	Fichero de cabecera (header)
.exe	Ejecutable en windows
.o	Fichero objeto (intermedio)
.dev	Fichero de proyecto en Dev C++
.dll	Librería dinámica en windows
.lib	Librería estática en windows

Para mantener un orden en la creación de nuestros programas, separaremos los ficheros en una serie de directorios. Se recomienda al alumno la siguiente estructura de directorios del proyecto:

…/practica0/
- →/exe Fichero ejecutable de salida
- →/obj Ficheros objeto intermedios
- →/src Ficheros fuente (.c, .cpp)
- →/inc Ficheros de cabecera (inc o include)
- →/out Ficheros de salida del programa
  - →/in Ficheros de entrada del programa

practica0.dev

II) Proyectos:

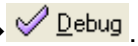
Un proyecto es una agrupación de ficheros (.c o .cpp) que forman un único programa. Normalmente, uno de los ficheros contendrá una función *main()*. Según se explicó antes, en Dev-C++ se debe crear un proyecto y después asociar los ficheros fuente necesarios. Cada fichero fuente (.c, .cpp) convendría que llevara asociado un fichero de cabecera (.h).

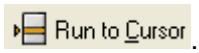
1. Pasos a realizar para crear un proyecto en Dev-C++ de acuerdo al estándar de normalización:

   - Crear un directorio para la creacion del proyecto: practica0

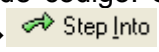
- Crear un proyecto: **New→ Project→ Console Application**: Nombre: ejnorma. Guardarlo en el directorio creado.
- Configurar el proyecto: **Project→Project Options→ Options**
  - Directorio de salida del ejecutable → exe
  - Directorio de salida del fichero objeto → obj
- Crear con el explorador de Windows los directorios **inc**, **src**, **exe** y **obj** dentro de la carpeta del proyecto.
- Descargar el fichero ejnorma.zip. Guardar los ficheros en los directorios **inc** y **src** adecuadamente.
- Añadirlos al proyecto: Pestaña de proyecto→ **Add to Project**.
- Ahora sería necesario teclear el código. Gran parte del código ya está completado, pero faltan algunos detalles importantes, como la inclusión del encabezado del proyecto y la definición de los prototipos de las funciones.
- Compilar, linkar y ejecutar.
- Comprobar con el explorador que los ficheros objeto y el ejecutable se encuentran en sus directorios correspondientes.

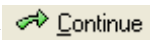
2. Explicación depurador utilizando el ejemplo anterior.

   - Activar la depuración de código. **Project → Project Options → Compiler → Linker → Generate debugging information**.
   - Recompilar todo → 
   - Depurar→ 

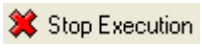
     . Inserción de Breakpoints (puntos de parada para la depuración).
   - Ejecutar el programa hasta el cursor: **Run to cursor** → 

     . En este caso deberemos elegir la opción del programa que nos lleve a ese punto del código.
   - Ejecutar una línea de código: **Next Step** (Step over)→ 

   - Ejecutar una línea de código. Si la instrucción es una función, saltar a su código: **Step into** → 

   - Continuar con la ejecución normal del programa: Continue→ 

   - Visualización de variables: Navegador del proyecto→Pestaña Depurar→**Add watch** → 

   - Parar el debugger. Stop Execution → 

3. Utilización de la ayuda. Dev-C++ no tiene integrada un sistema de ayuda. A la hora de programar es práctica común consultar el prototipo de funciones, tipos de datos… etc. Se aconseja al alumno que consulte alguna de las direcciones siguientes para obtener ayuda:

   http://www.cppreference.com/
   http://www.cplusplus.com/ref/

b. Diagramas de Flujo

Un programador eficiente maneja problemas de una manera sistemática mediante un proceso de programación metódico y minucioso con el fin de no dejar ningún cabo suelto a la hora de realizar un programa.

Es por ello que, generalmente, el problema se resuelve mediante una serie de fases, que suelen ser las que siguen:

1. **Planteamiento del problema**: es imprescindible estar completamente seguros de haber comprendido en que consiste el problema y por tanto, ser capaces de plantearlo mediante un enunciado, especificando los datos que van a intervenir en él.

2. **Análisis**: no es más que obtener un algoritmo que resuelva el problema (nosotros nos basaremos en diagramas de flujo por su sencillez).

3. **Programar el código fuente**: se trata de transformar a lenguaje máquina, mediante un lenguaje de programación, el algoritmo que hemos obtenido.

4. **Compilar el programa**: obtención del código objeto, mediante el código fuente que hemos creado. Aquí interviene la fase de depuración de errores

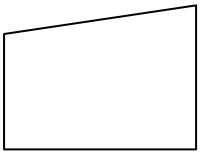
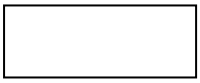
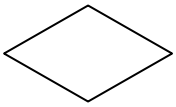
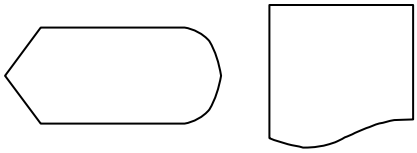
5. **Ejecución del programa**: si no hiciera falta linkado previamente, mediante la ejecución del programa, obtendríamos el resultado que pretendíamos y daríamos solución a nuestro problema.

Al tratarse de una revisión de lo aprendido en Fundamentos de Programación, nuestro objetivo en el presente tema, para la asignatura de Informática Aplicada, será efectuar un *Análisis* del problema mediante Diagramas de Flujo.

c.- Definición y simbología de los Diagramas de Flujo

Podemos definir un diagrama de flujo como una técnica gráfica para la representación de algoritmos.

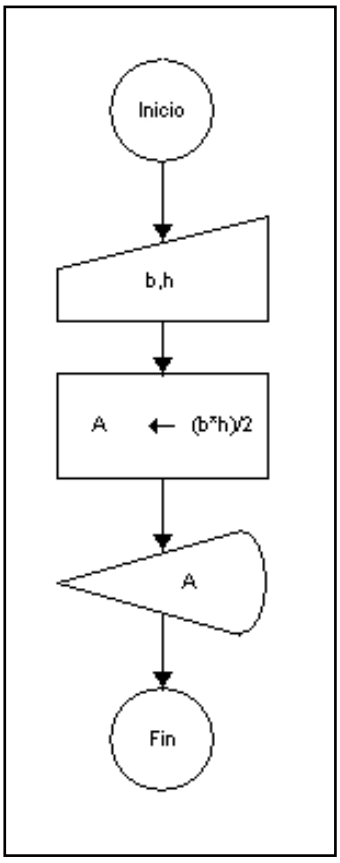
Aunque la nomenclatura puede variar según las fuentes que consultemos, en este caso, nos basaremos en la utilizada por el software DFD v1.0, que es el que utilizaremos en las prácticas para representar dichos diagramas:

Símbolo	Descripción
	**Inicio/Fin del diagrama**. Se suelen poner las palabras “Inicio” y “Fin” dentro del símbolo
	**Entrada de datos/Lectura** Se indican los valores iniciales para el proceso, es decir, definimos las variables y sus valores. Este símbolo debe tener como mínimo una conexión entrante (casi siempre del inicio) y una de salida
	**Proceso de datos/Asignación** Se usa para señalar operaciones matemáticas, aritméticas o procesos que se vayan a realizar con los datos Siempre debe tener como mínimo una conexión de entrada y una de salida
	**Decisión** Representa una disyuntiva lógica o decisión Es el único que puede contener dos salidas
	**Salida / Pantalla / Desplegado de información** Muestra un resultado. Debe tener al menos una conexión entrante y una saliente
	**Ciclo PARA o ciclo automático** Se corresponde con FOR
	Línea de flujo

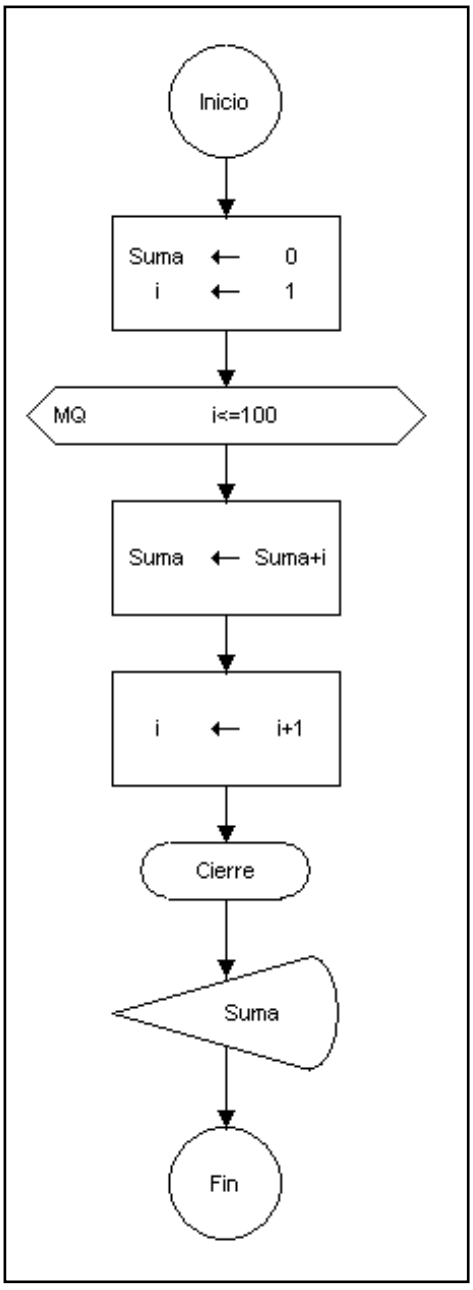
d.- Ejemplos con diagramas de flujo

A continuación se muestran varios ejemplos del uso de diagramas de flujo para resolver problemas:

1.- Diagrama de flujo para calcular y mostrar el área de un triángulo



2.- Imprimir la suma de los números del 1 al 100



e.- Ejercicios a resolver

1.- Realizar el diagrama de flujo para calcular el mayor de 3 números que introduciremos por el teclado

2.- Ley de Ohm

La ley de la física del área de electrónica o electricidad, llamada "*Ley de Ohm*", dice que:

"…una fuente eléctrica con una diferencia de potencia V, produce una corriente eléctrica I cuando pasa a través de la resistencia R…"

Siendo su fórmula:

$$V=I\cdot R$$

donde, empleando unidades del Sistema internacional:

- *V* = Diferencia de potencial en voltios (V)
- *I* = Intensidad en amperios (A)
- *R* = Resistencia en ohmios (Ω).

Se pide que realice los siguientes apartados:

a) Crear un diagrama de flujo que:

   a. Contenga una función llamada "*CalculaVoltaje*" que pasándole como parámetros la corriente en Amperios y la resistencia en Ohmios, calcule y devuelva mediante un return el valor de la diferencia de potencial calculado en Voltios.

   b. Contenga una función llamada "*CalculaResistencia*" que pasándole como parámetros la diferencia de potencial y la corriente, calcule y devuelva mediante un return el valor de la resistencia en Ohmios.

   c. Contenga una función "*main*", que:

      i. En primer lugar, pida que se introduzcan por teclado un valor para la corriente y otro para la resistencia, y se le pasen a la función "*CalculaVoltaje*", y la respuesta obtenida por la función se imprima por pantalla.

ii. En segundo lugar pedirá nuevos valores por teclado para la corriente y para la diferencia de potencial, utilizando el mismo nombre para las variables que en el caso anterior de este apartado. A continuación, se llamará a la función "*CalculaResistencia*" y se le pasarán dichos parámetros, devolviendo la función el valor de la resistencia e imprimirlo por pantalla.

b) Programar en C los apartados anteriores, basándose en los Diagramas de Flujo obtenidos anteriormente

3.- Realizar el programa en C a partir de los diagramas de flujo para los ejemplos mostrados en el apartado *c)* "*Ejemplos con diagramas de flujo*":

a) Calcular y mostrar el área de un triángulo
b) Imprimir la suma de los números del 1 al 100
c) Calcular el mayor de 3 números