PRÁCTICA Nº 2: 1 sesión

(el 22 de marzo y del 30 de marzo al 2 de abril de 2004)

Tipos Abstractos de Datos. Utilización e implementación de clases

0. OBJETIVO

El objetivo de esta práctica es aprender a implementar y utilizar un tipo abstracto de datos utilizando clases de C++. Para ello se propone un T.A.D. conjunto con unas operaciones definidas, y el alumno tendrá que implementar y probar la clase con un programa de ejemplo proporcionado por el profesor.

Una vez comprobada la clase, se propone resolver un problema simple del que se proporciona el algoritmo de resolución.

Es importante en esta práctica que a partir del enunciado de la práctica el alumno sea capaz de rellenar correctamente las hojas de documentación.

1. TIPO ABSTRACTO DE DATOS

Un tipo abstracto de datos (TAD) es, básicamente, una colección de valores junto con unas determinadas operaciones definidas sobre ellos. Para el usuario del tipo abstracto de datos sólo importan los valores que puedan guardar las variables asociadas al TAD y las operaciones de manipulación del mismo, dejando para el programador la tarea de la implementación de la estructura o tipo de dato que soporte el TAD y de las operaciones relacionadas con él.

En esta práctica vamos a realizar los dos papeles: Por un lado el papel de programador de un tipo abstracto de datos y por otro el papel de usuario de ese tipo de datos (tipo *Conjunto de enteros*).

2. TAREAS A REALIZAR

Implementación de la clase *Conjunto*

Crearemos una clase 'Conjunto' con la siguiente interfaz:

```
class Conjunto
{
    public:
        //Tipo de datos de los elementos del conjunto
        typedef int Elemento;

        // Inicia el conjunto a vacío
        void IniciarConjunto (void);
        // Añade un 'Elemento' al conjunto
        // Si la estructura para guardar la información es estática
        // y el elemento si que cabe, devolverá el exito de la función.
        bool AnadirElemento (Conjunto::Elemento);
        // Elimina el 'Elemento' del conjunto
        // Si el 'Elemento' no pertenece al conjunto simplemente no hace nada
        void BorrarElemento (Conjunto::Elemento);
        // Comprueba si el conjunto es o no vacío
        bool EsConjuntoVacio (void);
        // Comprueba si el 'Elemento' esta contenido en el conjunto
        bool Contiene (Conjunto::Elemento);
        // Muestra todos los elementos contenidos en el conjunto
        void MostrarConjunto (void);

    private:
        .??.
};
```

En este caso, como somos los programadores de la clase, tendremos que decidir cual es nuestra parte privada y a partir de esta decisión implementar los diferentes métodos de la clase.

Esta clase deberá funcionar con el programa '**`ProbarConjuntos.cpp`**' proporcionado por el profesor.

Utilización de la clase *Conjunto de enteros* para implementar la criba de Eratóstenes

En este apartado vamos a ser 'usuarios' de la clase *Conjunto*. Como ejemplo de utilización de la estructura de datos *Conjunto de enteros*, se propone escribir un programa que implemente el método de obtención de números primos propuesto por Eratóstenes y conocido como la criba de Eratóstenes aplicado a los 'N' primeros números naturales.

El método consiste, básicamente, en los siguientes pasos:

1.- Pedir el número 'N' al usuario.

2.- Guardar los números que queremos cribar en un *Conjunto* (desde 2 hasta N),

3.- Desde i con valor 2 hasta N

   a.- Si el elemento 'i' está en el conjunto, 'i' es primo (guardar 'i' en el conjunto de primos).

   b.- Eliminar del conjunto todos aquellos elementos que sean múltiplos de 'i' (incluyendo el propio 'i').

4.- Mostrar el conjunto de primos.

Recordad que como usuarios de la clase, sólo podemos acceder a la clase a través de los métodos y definiciones públicas de la clase, la parte **`private`** es cosa del programador, y no nos importa a la hora de utilizar la clase como usuarios.

3. REQUISITOS

Recordar que para poder acceder a la sesión de prácticas correspondiente es necesario entregar un pequeño esquema de lo que se va a realizar en la práctica.

En esta práctica, habrá que entregar una documentación de programa completa para la criba de Eratóstenes, explicando el programa principal y la clase '*Conjunto*'. Si no se presenta la documentación ANTES de acceder al laboratorio, no se permitirá la entrada a la sesión de prácticas.

4. ENTREGA

La entrega de esta segunda práctica debe incluir **el código** con el programa principal y las funciones del programa que implemente la criba de Eratóstenes (**`##criba.cpp`**), y la implementación de la clase 'Conjunto' (**`##conjunto.cpp`** y **`##conjunto.h`**)

La entrega puede realizarse por e-mail a la dirección del profesor encargado del grupo o en disquette al iniciar la siguiente sesión de prácticas, pero siempre ANTES de empezar la siguiente sesión de prácticas.

Cualquier práctica entregada fuera de plazo no será admitida para su corrección.

ENTREGA DE PROGRAMAS:
Al comenzar la sesión 3 de prácticas (29 de marzo, 6, 7, 22 y 23 de abril de 2004)

COMPILACIÓN SEPARADA EN DEV-C++

Para realizar un programa con diferentes módulos en Dev-C++ se utiliza el concepto de 'proyecto'.

En Dev-C++, un proyecto es un conjunto de ficheros que una vez compilados conformarán un solo programa ejecutable.

La manera de crear un proyecto es utilizando la opción 'New Project' que aparece en el menú 'File'.

Una vez seleccionada la opción nos pedirá qué tipo de proyecto queremos y el nombre del proyecto y el nombre del fichero en el que queremos guardar el proyecto. En este curso nos limitaremos a realizar proyectos de tipo 'consola' (*Console aplication*).

A partir de aquí, Dev-C++ nos abre un proyecto que contiene una sola unidad o módulo con el nombre 'Untitled1', y que contiene un esbozo de un programa principal.

Podemos utilizar esta primera unidad para empezar a realizar nuestro programa o podemos añadir o eliminar unidades utilizando el botón 'Project' del menú o pinchando con el botón de la derecha sobre el árbol de proyecto desplegado de la izquierda del programa.

Aunque no es necesario que los ficheros '.h' estén incluidos en el proyecto, es conveniente para facilitar su escritura y manipulación.

Una vez tengamos escritas las diferentes unidades de que conste nuestro programa podemos utilizar los botones habituales para 'compilar', 'ejecutar' o 'compilar y ejecutar' para probar el programa escrito.