Fundamentos de Programación

Grado en Ing. Informática

Guión práctico nº 3

Tema 4.- Tipos de datos estructurados

Dpto. de Tecnologías de la Información
Profesores Martín Ramos J. M., Polo Almohano M.P., Roche Beltrán F.

1. Implemente los métodos de la clase siguiente y diseñe un main que compruebe el funcionamiento de dichos métodos.

```
#define   M  10
class uno {
      int   tabla[M];
public:
      void  cargar();
      //Rellena la tabla con valores enteros leídos desde teclado

      int   maximo();
      //Devuelve el valor máximo almacenado en la tabla

      int   minimo();
      //Devuelve el valor mínimo almacenado en la tabla
};
```

2. Implemente los métodos de la clase siguiente y diseñe un main que compruebe el funcionamiento de dichos métodos:

```
#define   M   10
class matrices {
      int   tabla[M];
public:
      void   cargar();
      //Pondrá en cada elemento de la tabla el valor de su índice.

      bool   encontrar();
      //Pedirá un número entero por teclado y devolverá true si ese
      //número está en la tabla, en caso contrario devolverá false.
};
```

3. Implemente los métodos de la clase siguiente y diseñe un main que compruebe el funcionamiento de dichos métodos:

```
#define   M  10
#define   N   15
class  tres {
      int tabla[M][N];
public:
      void   cargar();
      //Pondrá en cada elemento de la tabla el valor del producto de
      //sus índices.

      int  encontrar();
      //Pedirá un número entero por teclado y devolverá  1 si ese
      //número está en la tabla, en caso contrario devolverá 0.
};
```

4. Implemente los métodos de la clase siguiente y diseñe un main que compruebe el funcionamiento de dichos métodos:

```
#define  M  3
#define  N  4
typedef   char cadena[30];

class cuatro {
     cadena  tabla[M][N];
public:
     void cargar();
     //Pondrá en cada elemento de la tabla una palabra leída desde
     //teclado.

     void encontrar();
     //Pedirá una palabra por teclado y mostrará por pantalla si esa
     //palabra está o no en la tabla y en qué fila y columna se
     //encuentra
};
```

5. Implemente los métodos de la clase siguiente y diseñe un main que compruebe el funcionamiento de dichos métodos:

```
#define   M  2
#define   N  4
typedef char cadena[30];
struct persona {
     int dni;        //DNI de la persona sin letra
     cadena nombre;  //Nombre de la persona
};

class matrices {
     persona tabla[M][N];
public:
     void cargar();
     //Pondrá en cada elemento de la tabla un dni (sin letra) y un
     //nombre leídos desde teclado.

     void encontrar();
     //Pedirá un dni por teclado y mostrará por pantalla si ese dni
     //está o no en la tabla y además en el caso de que esté
     //mostrará  el nombre de esa persona.
};
```

6. Determine si dos tablas tab1 y tab2 tienen el mismo contenido. Diseñe una clase que resuelva este problema.

```
class vector {
      int tab1[10], tab2[10];
public:
      void cargar ();
      //Llenara las dos tablas con valores leídos desde teclado.

      int comparar ();
      //Devolverá un 1 si son diferentes tab1 y tab2, 0 si son
      //idénticas
};
```

7. Implemente una clase de tal forma que fusione dos tablas (ordenadas previamente) en otra tercera, de tal forma que los elementos queden ordenados en ésta.

```
class  merges {
      int  uno[15], dos[15], fus[30];
      int numuno; //Número de elementos almacenados en la tabla uno
      int  numdos;//Número de elementos almacenados en la tabla dos
public:
      void cargar ();
      /*Preguntará al usuario cuantos elementos va a poner, colocando
este valor en  numuno, a continuación solicitará tantos elementos
como  numuno, almacenándolos en la tabla uno.

      Preguntará al usuario cuantos elementos va a poner, colocando
este valor en numdos, a continuación solicitará tantos  elementos
como  numdos, almacenándolos en la tabla dos.

      Los valores es preciso ponerlos desde teclado ordenados
crecientemente para cada tabla. */

      void   mezclar ();
      //Cargará la tabla fus con los valores de las tablas uno y dos
      //quedando la tabla fus ordenada crecientemente.

      void ver();
      //Visualizará por pantalla el contenido de las tablas uno y
      //dos.

      void verfusion();
      //Visualizará por pantalla el contenido de la tabla fus.
};
```

8. Implementar la clase **PalabraOculta**, que nos permita jugar a adivinar una palabra. Dicha clase tendrá las siguientes características:

Atributos privados:

- **`palabraSecreta`**, que contendrá la palabra a adivinar.
- **`Puntos`**, que almacenará los puntos conseguidos por el jugador.

Métodos públicos:

- **`void Iniciar()`**
  Se establecerán los parámetros de inicio del juego:
  a) Se le pedirá al usuario la palabra a adivinar, teniendo en cuenta que dicha palabra no se podrá ver por pantalla y que en su lugar se mostrará una cadena de asteriscos. Además la cadena leída deberá almacenarse siempre en mayúsculas. Por todo anterior, al alumno le serán de utilidad las siguientes funciones:
     - **`getch()`** (librería **`conio.h`**), que lee un carácter desde teclado sin producir eco en pantalla.
     - **`strupr()`** (librería **`cstring`**), que convierte una cadena de texto a mayúsculas.
  b) Se establecerá a 9 la cantidad de puntos de partida del usuario.

- **`int Jugar()`**
  El juego le mostrará al usuario una palabra con tantos guiones '-' como letras tenga la palabra oculta y le pedirá una letra. Si la letra se encuentra en la palabra oculta se mostrará en su/s posición/es correspondiente/s junto con el resto de guiones y letras yas descubiertas. Si por el contrario la letra no está, se le quitará un punto.
  Si el jugador se queda con 0 puntos o bien adivina la palabra, el juego finaliza y el método devolverá la cantidad de puntos conseguidos.
  La letra solicitada al usuario deberá ser transformada a mayúsculas. Para ello el alumno podrá utilizar la función **`toupper()`** que se encuentra en la librería **`cctype.`**

- **`void MostrarSecreta()`**
  Muestra por pantalla cuál es la palabra secreta.

Codificar además un programa que nos permita jugar haciendo uso de esta clase.

9. Implemente la clase **Análisis** que se utiliza para analizar la frecuencia de entradas de números enteros entre 0 y 24 inclusive (25 elementos en total). Para ello implemente la siguiente clase cuyos métodos se explicarán a continuación.

```
#define TAMA 25

class Analisis
{
        int Datos[1000];
        int NDatos;
        int Valores[TAMA];
    public:
        void PedirDatos();
        void AnalizarDatos();
        bool EstanTodos();
        int ValorRepetido();
        int ValorMasRepetido();
        void MostrarDatos();
        void MostrarAnalisis();
};
```

Atributos privados:

- **`Datos`**. Vector de 1000 elementos donde se almacenarán números pertenecientes al intervalo [0,24]
- **`NDatos`**. Entero que almacena el número de valores almacenados en el vector **Datos**.
- **`Valores`**: Vector que contabilizará el número de veces que ha aparecido un valor en el vector **Datos**. Un ejemplo:

  Si el vector Valores contuviese los siguientes valores indicaría:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...	24
2	0	5	2	14	2	0	0	4	8	9	2	1	5	0	0	2	1	12

  El valor 0 ha aparecido dentro del vector Datos 2 veces.
  El valor 1 no ha parecido en el vector Datos.
  El valor 2 ha aparecido en el vector Datos 5 veces.
  El valor 3 ha aparecido en el vector Datos 2 veces.
  El valor 4 ha aparecido en el vector Datos 14 veces.
  etc.

Métodos públicos:

- **`void PedirDatos();`**

  Método utilizado para introducir valores en el vector **Datos**. Para ello, el método solicitará al usuario si desea introducirlos manualmente (**m**) o de generarlos aleatoriamente (**a**).

  - Si el usuario introduce **m**, el método deberá preguntar valores e introducirlos dentro del vector **Datos** hasta que se introduzca el valor **-1**. El valor de **NDatos** deberá ser actualizado consecuente con el número de datos introducidos en el vector **Datos**.
    Nota: Si el usuario introduce un valor fuera del rango [**0,24**] el método deberá solicitar nuevamente dicho valor, siempre y cuando no sea -1 ya que en dicho caso indica que la entrada de datos manual ha terminado.
  - Si el usuario introduce **a**, el método generará un valor aleatorio para **NDatos** en el intervalo [**1,1000**] y posteriormente generará y añadirá al vector **Datos** tantos valores aleatorios en el intervalo [**0,24**] como indique **NDatos**.

- **`void AnalizarDatos();`**

  Método que actualiza el vector **Valores** mediante el análisis de los datos del vector **Datos**. El análisis consiste en la contabilización del cuantos valores se repiten en el vector **Valores**. Por ejemplo, si el valor **5** se repite **20** veces en el vector **Valores**, la posición de índice **5** del vector **Datos**, tendrá el valor **20**.
  En resumen, cada **posición** del vector **Valores** indicará cuantas veces se repite el valor **posición** en el vector **Datos**.

- **`bool EstanTodos();`**

  Método que devolverá **true** si el análisis de los datos demuestra que todos los valores del intervalo [**0,24**] aparecen al menos un vez en el vector **Valores**, en caso contrario devolverá **false**. Recordad que el análisis está realizado en el vector **Valores**.

- **`int ValorRepetido();`**
  Método que solicita por teclado un valor del intervalo [**0,24**] y devolverá cuantas veces está repetido. Si el usuario introduce un valor fuera del rango deberá volver a preguntar. Recordad que esta información está en el vector **Valores** generado por el método **AnalizarDatos**.

- **`int ValorMasRepetido();`**
  Método que devolverá qué valor del intervalo [**0,24**] es el más repetido. Para ello se deberá consultar el vector **Valores** generado por el análisis.

- **`void MostrarDatos();`**
  Método que muestra por pantalla todos los valores del vector **Datos**.

- **`void MostrarAnalisis();`**
  Método que muestra por pantalla todos los valores del análisis (vector **Valores)**.

Realizar un programa con un menú que permita seleccionar y utilizar todos los métodos de la clase.

10. Implementar la clase **Tesoro**, que nos permita jugar a buscar un tesoro rodeado de minas a distinta profundidad. Para ello contará con:

Atributos privados:

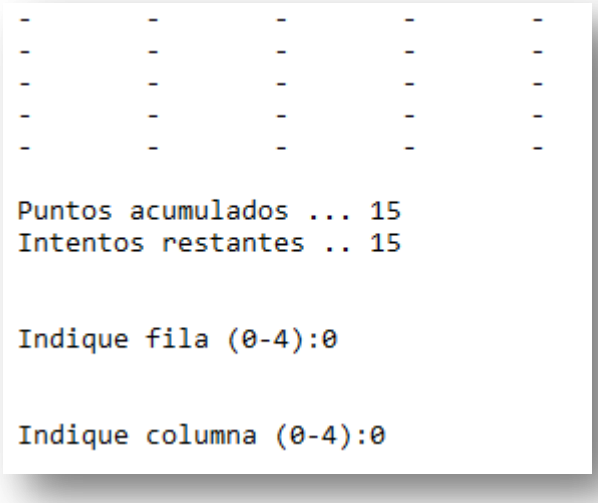
- **`Tablero`**. Será una matriz de **`5x5`** que almacenará en cada casilla:
  1. **Una letra**, indicando si el contenido es el **tesoro** (**`'T'`**), una **bomba** (**`'B'`**) o **arena** (**`'A'`**)
  2. **Un valor entero**, indicando la **profundidad** de la bomba. Habrá 3 niveles de profundidad (**`1`**, **`2`** ó **`3`**, de menos a más profundo). Las bombas más superficiales son las que más puntos quitan.
- **`Puntos`**. Será un valor entero con los puntos conseguidos por el jugador.

Métodos públicos:

- **`void Iniciar()`**
  Se establecen en **15** los **puntos iniciales** con los que parte el jugador y se inicializa el tablero de juego, debiendo introducir **arena**, **1 tesoro**, **1 bomba** a **profundidad 1**, **2 bombas** a **profundidad 2** y **3 bombas** a **profundidad 3**.
  Las casillas seleccionadas para insertar el tesoro y las bombas deben ser completamente aleatorias, por lo que serán de utilidad las siguientes funciones:
  - **`rand()`**. Se encuentra en la librería **`cstdlib`** y obtiene un número aleatorio entre **0** y **`RAND_MAX`** (constante con un valor muy elevado).
  - **`srand()`**. Para conseguir que la semilla utilizada por **`rand()`** sea diferente cada vez que se juega, y por lo tanto las casillas se rellenen de forma distinta en cada ocasión. Esta función se utiliza conjuntamente con **`time(0)`**, que ofrece la hora actual y se encuentra en la librería **`ctime`**. Así pues **`srand()`** se usaría en nuestro código del siguiente modo **`srand(time(0))`** y sólo al comienzo del juego.

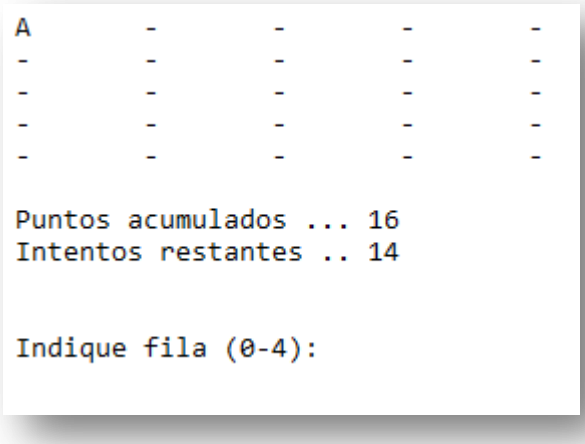
- **`bool Jugar()`**

  En primer lugar se establecerá como número máximo de tiradas 15. A continuación se mostrará el tablero ocultando lo que contiene cada casilla por el símbolo `'-'` y se le pedirá al usuario qué casilla quiere descubrir, por lo que tendrá que indicar una fila y una columna (ambas serán valores comprendidos entre 0 y 5). Si los valores indicados son erróneos se le mostrará un mensaje y se volverán a pedir.



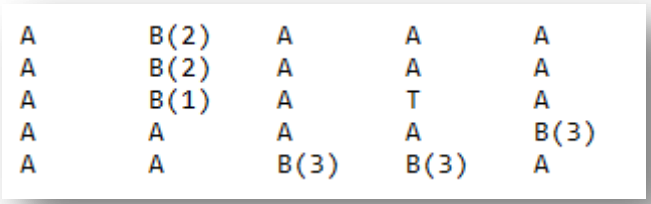
  En función de la casilla descubierta se actuará de un modo diferente:

  a) Si se ha encontrado el **tesoro**, el jugador suma 100 puntos y el juego finaliza.
  b) Si se ha encontrado **arena**, se le añade 1 punto a los ya acumulados.
  c) Si se ha encontrado una **bomba**, se le restará la siguiente cantidad de puntos a los ya acumulados **4-profundidad**.



  Tras la tirada, se volverá a mostrar el tablero con la casilla que se acaba de descubrir, se mostrarán los puntos acumulados junto con las oportunidades que restan, y si no se ha encontrado el tesoro o aún quedan oportunidades, se volverá a pedir al usuario que indique las coordenadas de otra casilla.
  El juego terminará cuando se encuentre el tesoro (el método devolverá **`true`**) o cuando se agoten las 15 oportunidades (el método devolverá **`false`**).

- **`void MostrarTablero()`**
  Mostrará por pantalla el contenido del tablero siguiendo la idea indicada en esta imagen:



Realizar un programa que nos permita jugar, que nos de la enhorabuena si encontrarmos el tesoro y que nos muestre cuál era el tablero de juego original en el caso de no haberlo conseguido.