Fundamentos de Programación

Relación de Problemas
Tema 5

1.- C++ no realiza comprobación de contorno cuando se introducen datos en una tabla, de manera que es posible (intencionada o erróneamente) introducir datos en una tabla indicando un índice de celda no existente (fuera de rango). Para evitar esto, implemente una clase vector seguro que admita un nº máximo de 100 enteros y que controle en todo momento que el índice indicado esté dentro de rango. Proporcione dos constructores a la clase: uno que admita un parámetro indicando el nº de elementos del vector creado y otro que admita dos: el primero indica el nº de elementos del vector y el segundo es una tabla con los valores dados. Proporcione además 3 métodos que devuelvan respectivamente el nº de elementos del vector, el valor almacenado en una determina celda (entre 1 y nº elementos) y una referencia a dicha celda (el índice de la primera celda es el 1). Realice un programa que compruebe su funcionamiento.

```
class vector {
 int celda[MAXIMO]; // array para almacenar los elementos
 int nelem; // nº de elementos del vector
public:
 vector(int n); // constructor
 vector(int n, int valores[]); // constructor
 int longitud(); // devuelve el nº de elementos del vector
 int get(int n); // devuelve el valor de celda[n]
 int &put(int n); // devuelve una referencia a celda[n]
}
```

2.- Añade un método público a la clase anterior que invierta el contenido del vector.

3.- Añade un método público a la clase vector que permita mezclar ordenadamente dicho vector con otro determinado.

4.- Añade un método público que permita visualizar el contenido del vector en pantalla.

5.- Sobrecargue el operador == de manera que devuelva si un vector es igual a otro.

6.- Implemente una clase matriz segura que permita gestionar matrices bidimensionales de 4 x 4 como máximo. Proporcione una serie de métodos para poder leer y modificar los valores almacenados en las distintas celdas, teniendo en cuenta que las coordenadas de las celdas deben estar comprendidas entre [1-nº filas] y [1-nº col]. Proporcione también una serie de métodos que permitan consultar o modificar el nº de filas y columnas que tiene dicha matriz, así como un método que visualice la matriz en pantalla.

```
class matriz {
 int celda[MAXIMO][MAXIMO];
 int fila,col;
public:
 matriz(int f, int c); // constructor
 ...
};
```

7.- Añade un método público que indique si la matriz es simétrica o no.

8.- Sobrecargue los operadores + y * que manera que permitan sumar y multiplicar matrices respectivamente.

9.- Sobrecargue el operador == de manera que devuelva si dos matrices son iguales.

10.- Cree una función genérica que permita cargar por teclado una matriz dada por referencia. Sobrecargue dicha función de manera que pregunte además el tamaño de la matriz desde teclado.

```
// lee por teclado los valores de la matriz
// de tamaño m.fila x m.col
void cargar(matriz &m);
// idem pero ademas pide por teclado el tamaño de la matriz
int cargar(matriz &m, int teclado);
```

11.- Dada la siguiente definición de clase almacen, implemente los métodos insertar y existe.
El método insertar añade el nuevo producto P al almacén. El método debe comprobar que dicho producto es nuevo (no existe ya en el almacén) así como que queda sitio para insertar el producto que se le indica. Devuelve 1 si lo añade, 0 si el producto ya existe (no lo añade) y 2 si no queda sitio en el almacén (no lo añade).
El método existe devuelve la posición en la que se encuentra el producto, cuyo nombre se pasa como parámetro, dentro del almacén. Si dicho producto no existe devolverá -1.

```
#define MAX 5

struct datosprod {
   char nombre[20];
   float precio;
   int stock;
};

class almacen {
  datosprod productos[MAX];
  int nprod;
public:
  almacen() { nprod = 0; }
  int insertar(datosprod P);
  int existe(char nombre[20]);
}
```

12.- Añade un método público a la clase anterior que permita unir 2 almacenes, sobrecargando el operador +.

almacen operator+(almacen A);
/* Dados los almacenes A1 y A2, el resultado de la operación A1+A2 será un almacén en el que se encuentren todos los productos del almacén A1, más todos los productos del almacén A2, de modo que si algún producto del almacén A2 ya existía en el almacén A1, sólo habrá que sumar los stock y recalcular el precio que será la media ponderada de los precios existentes. Este operador no comprueba que en el almacén resultado de la suma, tengan cabida todos los productos de ambos almacenes.*/

13.- Añade un método público a la clase anterior que permita eliminar de un almacén todos los productos que ya existan en otro almacén, sobrecargando el operador -.

almacen operator-(almacen A);
/* Dados los almacenes A1 y A2, el resultado de la operación A1-A2 será un almacén en el que se encuentren todos los productos del almacén A1 que no estén en el almacén A2. */

14.- Dada la siguiente declaración de la clase agenda y la clase contacto, implementa los métodos eliminar_contacto y nuevo_telf, utilizando los métodos públicos de la clase contacto y de la clase agenda. Cualquier método u operador sobrecargado, de los definidos en las clases, que sea usado en la implementación de los dos métodos anteriores deberá implementarse también.

```
#define N 100

typedef char cadena[20];

class contacto {
  cadena nombre;
  double tlfns[4];
  int ntlf; // cantidad de teléfonos distintos del contacto
public:
  contacto() { ntlf = 0; }
  contacto(cadena nom, double tfs[], int nt);
  int getntlf() { return ntlf; }
  int buscartlf(double t);  // 1 si t existe, 0 si no
  void getnombre(cadena nom);
  void cambiar_nombre(cadena nom);
  int operator==(contacto c); // 1 si son iguales y 0 si no
  int operator!=(contacto c); // 1 si son distintos,0 si no
  void operator=(contacto c); // permite asignar objetos
                              // de la clase contacto
  contacto operator+(double t); // añade un teléfono
                                // a un contacto
};
```

```
class agenda {
  contacto listin[N];
  int nc;
public:
  agenda(){ nc = 0; }
  int buscar_contacto(contacto c);
/*Devuelve la posición del listin donde se encuentra el
contacto. Si no se encuentra devuelve -1*/
  void eliminar_contacto(contacto c);
/*Elimina el contacto si existe, si no no hace nada*/
  void modificar_nombre(contacto c, cadena nom);
/*Modifica el nombre del contacto c con el nuevo nombre
nom, si no existe el contacto no hace nada*/
  void insertar_contacto(contacto c);
/*Inserta el nuevo contacto c en la agenda siempre que no
exista y que no esté llena*/
  void nuevo_telf(contacto c, double tlfn);
/*Añade un nuevo teléfono a un contacto existente siempre
que no exista y no tenga ya 4 telefonos asignados*/
};
```

15.- Dada la siguiente declaración de la clase tabla y suponiendo los métodos en negrita ya implementados, implementa los métodos buscar, insertar y eliminar.

```
#define MAXIMO 8
typedef char cadena[20];

struct registro {
    cadena nombre;  // nombre de la persona
    int sig;        // indice del siguiente
};

class tabla {
  registro t[MAXIMO];
public:
  tabla();              // constructor
  int numelementos(); // numero de elementos de la tabla
  int estallena();    // 1 si la tabla esta llena, 0 si no
  int buscar(cadena c);  // 0 si no esta, 1 si esta
  int insertar (cadena c); // 1 si lo añade, 0 si no
  int eliminar (cadena c); // 1 si lo elimina y 0 si no
};
```

tabla();

/* El constructor tabla (ya implementado) pone la tabla t de la siguiente forma:

	0	1	2	3	4	.....
nombre	?	?	?	?	?	
sig	-2	-1	-1	-1	-1	

Indica que los elementos 1, 2, 3, 4 ... de la tabla están vacíos, (campo sig a –1).
El elemento 0 tendrá en su campo sig el valor –2, indicando que la tabla está vacía completamente. */

int insertar(cadena c);
/* Devuelve 0 si está llena y no se puede añadir c, 2 si la cadena ya existe en la tabla (no se añade) y 1 si se añade (la cadena c se añade en el último lugar).
Si se llama a este método con a.insertar("Mario") y suponiendo que la tabla t está con la siguiente información:

	0	1	2	3	4	....
Nombre	?	Carlos	?	Juan	Luis	
Sig	3	-2	-1	4	1	

La tabla tras ejecutar el método queda de la siguiente forma:

	0	1	2	3	4	....
Nombre	?	Carlos	Mario	Juan	Luis	
Sig	3	2	-2	4	1	

Podemos observar que "Mario" se ha colocado el último, detrás de Carlos que pasa a ser el penúltimo. */

int eliminar(cadena c);
/* Devuelve 1 si elimina c de la tabla. Si la cadena c no esta devuelve 0.
Si se llama a este método con a.eliminar("Carlos") y suponiendo que la tabla t está con la siguiente información:

	0	1	2	3	4	....
Nombre	?	Carlos	?	Juan	Luis	
Sig	3	-2	-1	4	1	

La tabla tras ejecutar el método queda de la siguiente forma:

	0	1	2	3	4	....
Nombre	?	Carlos	?	Juan	Luis	
Sig	3	-1	-1	4	-2	

"Carlos" se borra poniendo Sig a -1. (indica que esta libre).
"Luis" toma el valor -2 para indicar que es el ultimo elemento. */

int buscar(cadena c);

/* Devuelve 1 si la cadena c está en la tabla, 0 si no esta.
Suponiendo que la tabla t está con la siguiente información:

	0	1	2	3	4	....
nombre	?	Carlos	Eva	Juan	Luis	
sig	3	-2	-1	4	1	

Las cadenas existentes en la tabla son, por orden de aparición: Juan, Luis y Carlos. La cadena Eva no existe (está eliminada) */

16.- Añade un método público a la clase anterior que permita mostrar los elementos de la tabla en el orden en el que han sido introducidos.

void mostrar( );
/* Suponiendo que la tabla t tiene los siguientes valores:

	0	1	2	3	4	....
nombre	?	Carlos	?	Juan	Luis	
sig	3	-2	-1	4	1	

Siempre se empieza el tratamiento de la tabla por el elemento 0 y como su campo sig indica 3, el primero es el que está en la posición 3, sacando por pantalla "Juan", a continuación como el campo sig del elemento 3 tiene un 4, el siguiente elemento en orden es el 4, sacando por pantalla "Luis", que a su vez tiene en su campo sig el valor 1, luego el siguiente es el elemento 1, sacando por pantalla "Carlos" y como su campo sig tiene –2 indica que es el final de la tabla.
Es decir, deberá de sacar por pantalla los nombres en orden: Juan, Luis, Carlos */

17.- Sobrecarga el operador == de forma que permita comparar 2 tablas y determinar si las 2 tablas son iguales.

int operator==(tabla t);

/* Devuelve 1 si las tablas tienen los mismos elementos en el mismo orden, 0 si no. Por ejemplo las siguientes tablas son iguales:

	0	1	2	3	4	....
nombre	?	Carlos	?	Juan	Luis	
sig	3	-2	-1	4	1	

	0	1	2	3	4	....
nombre	?	Luis	Juan	?	Carlos	
sig	2	4	1	-1	-2	

Ya que los nombres aparecen en el mismo orden: Juan, Luis, Carlos */