Fundamentos de Programación
Sesión 11

Actividades a realizar en casa

Actividad: Resolución de problemas.

Resolved los siguientes problemas de la relación 4B. Recordad que, **ANTES** del inicio de esta sesión en el aula de ordenadores, hay que subir las soluciones a PRADO. Se debe subir un fichero llamado sesion11.zip que incluya todos los ficheros cpp (y solamente estos).

3 (Complejo)
4 (Fecha)
5 (Libro)
6 (Conjunto)
7 (EnteroLargo)

Actividades a realizar en las aulas de ordenadores

Clase vector

Cuando hemos utilizado vectores (array) en ésta asignatura, hemos tenido un problema recurrente con la memoria: debemos establecer a priori (en tiempo de compilación) cuántas componentes reservamos para los vectores y luego (en tiempo de ejecución) pedirle al usuario que especifique cuántas componentes necesita (siempre menos de las reservadas previamente, claro).

Este problema se resuelve utilizando memoria dinámica, que se tratará en la asignatura *Metodología de la Programación*. Mientras tanto, C++ ofrece una solución utilizando la clase vector. En realidad, es una clase que esconde un vector dinámico, pero no hace falta saber nada de memoria dinámica para utilizarlo, al igual que pasa con la clase string. vector es una plantilla (*template*, se verá qué eso en la asignatura Estructuras de Datos) definida en la STL (Standard Template Library), que contiene plantillas para las funciones y estructuras de datos más comunes en programación.

Crea, en U:\FP, la carpeta 11_Vector y copiaremos dentro el fichero 11_VectorEnteros.cpp disponible en PRADO. Este fichero contiene el programa de la Figura 1.

```cpp
#include <iostream>
#include <vector> // biblioteca con la definicion de la clase

using namespace std;

int main(){
    // Declaracion de objetos de la clase vector
    vector<int> vec1;
    vector<int> vec2(10); // inicializa componentes a cero
    vector<int> vec3(10,5); // inicializa componentes a 5

    // La funcion miembro size devuelve la longitud del vector
    cout << "\nLa longitud de los vectores es: ";
    cout << vec1.size() << " " << vec2.size() << " " << vec3.size() << "\n";

    cout << "El vector 2 es: ";
    for (int i=0; i<vec2.size(); i++)
        cout << vec2[i] << " "; // acceso a componentes con el operador []
    cout << "\n";

    cout << "El vector 3 es: ";
    for (int i=0; i<vec3.size(); i++)
        cout << vec3[i] << " ";
    cout << "\n";

    vec1 = vec3; // operador de asignacion sobrecargado

    cout << "Ahora, el vector 1 es: ";
    for (int i=0; i<vec1.size(); i++)
        cout << vec1[i] << " ";
    cout << "\n";

    cout << "Introduce valores pare el vector 2:";
    for (int i=0; i<vec2.size(); i++)
        cin >> vec2[i];

    cout << "Ahora, el vector 2 es: ";
    for (int i=0; i<vec2.size(); i++)
```

Figura 1. Fichero VectorEnteros.cpp

1. Declaración

Lo primero que vemos es que se debe incluir la librería vector para poder operar con la clase.

```
1  #include <iostream>
2  #include <vector> // biblioteca con la definicion de la clase
3
4  using namespace std;
```

Para declarar un vector seguimos la sintaxis vector<tipo_dato> v; que declara un vector vacío donde las componentes tienen un tipo de dato tipo_dato. Habría que añadir std:: si no hubiéramos añadido el espacio de nombres estándar.

```
6  int main(){
7      // Declaracion de objetos de la clase vector
8      vector<int> vec1;
9      vector<int> vec2(10); // inicializa componentes a cero
10     vector<int> vec3(10,5); // inicializa componentes a 5
11
```

Pero existen otros constructores:

- Pasando un entero como argumento, crea un vector de esa longitud.
- Pasando un entero, y un valor por defecto, creo un vector de esa longitud en el que todas las componentes toman el valor por defecto
  vector<char> lista_char(10, 'p'); // vector de 10 componentes con 'p'
- Constructor de copia, esto es, pasando como argumento otro vector con el mismo tipo de dato base
  vector<int> lista_enteros(otro_vector_enteros); // copia otro_vector_enteros

Además, contamos con la función miembro size() que devuelve la longitud actual del vector.

```
12     // La funcion miembro size devuelve la longitud del vector
13     cout << "\nLa longitud de los vectores es: ";
14     cout << vec1.size() << " " << vec2.size() << " " << vec3.size() << "\n";
15
```

2. Operadores básicos

La clase vector tiene sobrecargados el operador [] de acceso a las componentes. Así, por ejemplo, podemos realizar las operaciones de entrada y salida de datos de igual forma que con un vector tipo C.

```
16     cout << "El vector 2 es: ";
17     for (int i=0; i<vec2.size(); i++)
18         cout << vec2[i] << " "; // acceso a componentes con el operador []
19     cout << "\n";
20
```

```
33     cout << "Introduce valores pare el vector 2:";
34     for (int i=0; i<vec2.size(); i++)
35         cin >> vec2[i];
36
```

Una gran ventaja respecto a los *arrays* es que también está sobrecargado el operador de asignación =, por lo que podemos asignar completamente (y no necesariamente componente a componente) un vector en otro.

```
25
26      vec1 = vec3; // operador de asignacion sobrecargado
27
28      cout << "Ahora, el vector 1 es: ";
29      for (int i=0; i<vec1.size(); i++)
30          cout << vec1[i] << " ";
31      cout << "\n";
```

También están sobrecargados los operadores relacionales ==, !=, <, >, <=, >=.

```
42      if ( vec1 != vec2 )
43          cout << "El vector 1 es diferente al vector 2" << "\n";
44
```

El operador de igualdad primero comprueba el tamaño y, si son iguales, entonces compara elemento a elemento. El operador < compara los vectores según el orden lexicográfico.

3. Otras funciones

- empty(). Devuelve true si el vector no tiene elementos, o false, en caso contrario.
- front(). Devuelve el primer elemento del vector.
- back(). Devuelve el último elemento del vector.
- push_back(elemento). Añade un elemento al final del vector.
- pop_back(). Elimina el último elemento del vector.
- swap(vector<T> vec). Intercambia el contenido del vector con vec.
- clear(). Elimina todos los elementos del vector (lo deja con longitud 0).

```
45      while ( !vec1.empty() )
46          vec1.pop_back();
47
48      cout << "El vector 1 tiene tamanno: " << vec1.size() << "\n";
49
50      while ( !vec2.empty() ){
51          vec1.push_back( vec2.back() );
52          vec2.pop_back();
53      }
54
55      cout << "El vector 1 tiene tamanno: " << vec1.size() << "\n";
```

4. Clases y funciones

Un objeto de la clase vector puede pasarse como argumento de una función, como retorno de una función y puede ser un dato miembro de otra clase. Descarga 11_OrdenaVector.cpp desde PRADO para ver un ejemplo.

```cpp
#include <iostream>
#include <vector>

using namespace std;

void OrdenaSeleccion ( vector<int> & vec ){ // paso por referencia
    int minimo;
    int posicion_minimo;
    int intercambia;

    for (int izda = 0 ; izda < vec.size(); izda++){
        minimo = vec[izda];
        posicion_minimo = izda;
        for (int i = izda + 1; i < vec.size(); i++){
            if ( vec[i] < minimo ){
                minimo = vec[i];
                posicion_minimo = i;
            }
        }
        intercambia = vec[izda];
        vec[izda] = vec[posicion_minimo];
        vec[posicion_minimo] = intercambia;
    }
}

```

Más información sobre la clase en http://www.cplusplus.com/reference/vector/vector/

NOTA: Salvo que se indique lo contrario, no se puede utilizar la clase vector en los exámenes.

Actividad: Resolución de problemas.

Resolved los siguientes problemas de la relación 4B.

6 (Conjunto), usando la clase vector como dato miembro.
9 (Frase)
10 (VectorParejasCaracterEntero)