Manejo del tipo vector

Una variable de tipo `vector` contiene una secuencia ordenada de variables del mismo tipo. El tipo de la variable repetida se elige en la declaración del vector, así como también la cantidad de elementos (dimensión). Para usar este tipo de dato debemos incluir la librería correspondiente al principio del código:

```
#include <vector>
```

En el siguiente ejemplo declaramos un vector de 3 variables de tipo double, y le asignamos los valores correspondientes al primer vector canónico (1, 0, 0).

```
#include <vector>

using namespace std;

int main() {
        vector<double> e1(3);
        e1[0] = 1.0;
        e1[1] = 0.0;
        e1[2] = 0.0;
        return 0;
}
```

La variable de tipo `vector<double>` que definimos en el ejemplo precedente es lo que se considera un *array* estático, es decir, un conjunto ordenado de variables de un mismo tipo (en este caso `double`) cuyo cardinal está dado en tiempo de compilación (en este caso, 3 elementos). En contraposición a estos, en lenguajes modernos como C podemos declarar *arrays* dinámicos cuyo tamaño (cantidad de elementos) está dado en tiempo de ejecución. Es decir que el programa no sabe cuánta memoria deberá reservar para ese `vector` al comenzar el programa, y se enterará cuando llegue a la línea de declaración del mismo. En el siguiente ejemplo podemos ver la declaración de un `vector` dinámico, cuyo tamaño es ingresado por el usuario en tiempo de ejecución. De esta forma definimos el primer vector canónico en un espacio de dimensión `N`.

```
#include <vector>

using namespace std;

int main() {
        int N;
        cout << "Ingrese la dimension del espacio: ";
        cin >> N;

        vector<double> e1(N);
        e1[0] = 1.0;
        for(int i=1; i<N ; i++) {
```

```
                e1[i] = 0.0;
        }
        return 0;
}
```

Arrays de tamaño variable

En ocasiones vamos a querer ampliar la cantidad de elementos de un `vector` en tiempo de ejecución, de acuerdo a la información que va surgiendo en el programa. Esto sucede cuando queremos leer información de un archivo cuyo tamaño ignoramos. El código del siguiente ejemplo lee un archivo "funcion.dat" con dos columnas de valores reales, y las deposita en vectores cuyo tamaño inicial es nulo, pero que se agrandan en la medida en que lo necesitan. Al final del programa guardamos en `N` la cantidad de elementos del vector, y en `DX` el paso espacial.

```
int main() {
        ifstream filein("funcion.dat");

        if (filein.good() == false) {
                cout << "problema al abrir funcion.dat" << endl;
                exit(1);
        }

        vector<double> x, v;

        int i;
        for(i=0; filein.eof() == false ; i++) {
                double tmp;
                filein >> tmp;
                x.insert(x.end(), tmp);
                filein >> tmp;
                v.insert(v.end(), tmp);
        }
        int N = i-1;
        double DX = x[1] - x[0];
        filein.close();
        return 0;
}
```

El comando `insert` es el que nos permite ir agregando elementos a `vector` sin preocuparnos por un tamaño máximo. Existen varias formas de indicar la inserción de elementos en un `vector`. Con `vector.end()` nos aseguramos de que se vayan agregando al final de la lista. El efecto contrario se logra con `vector.begin()`.