P

La clase *vector* en C++

Pablo R. Ramis

```cpp
#include <iostream>
int main()
{
    std::cout << "In Code We Trust";
    return 0;
}
```

1. INTRODUCCIÓN

Dentro de la librería estándar de C++ ya encontramos varias estructuras de datos básicas las cuales nos resultarán de mucho interés y utilidad.

A diferencia de C, que nos veíamos obligados a generar nuestas propias estructuras abstractas, las de C++ al estar implementadas con *templates*, tienen un grado de polimorfismo que suman mucha potencia de uso.

vector nos permitirá disponer de un arreglo dinámico, sin tener que preocuparnos por el tamaño, teniendo asi una ventaja del tradicional *array* heredado de C.

1.1. vector

Será imprescindible el incluir a *vector.h* en la cabecera de nuestro código.

1.1.1. Construcción.

```
1  //initVector.cpp
2  #include <iostream>
3  #include <vector>
4  #include <string>
5  #include <algorithm>
6
7  using namespace std;
8
9  int main()
10 {
11     //vector sin inicializar ni indicar ñtamao
12     vector<double> Vector_1;
13 //    Vector_1 = 10;    // Error!!!
14 //    Vector[0]= 1;     // Error!!!
15
16     vector<int> Vector_3(10);
17
18     //vector con ñtamao 5 y componentes iniclizadas
19     vector<double> Vector_2(6,3.1416);
20
21     // Vemos el ñtamao del vector con .size()
22     cout << Vector_2.size() << endl;
23
24     //mostrar las componentes con un ciclo
25
26     for(int i=0; i<Vector_2.size() ;i++)
27     { //con el metodo .size() se obtiene el ñtamao del vector
28         cout << Vector_2[i] << endl;
29     }
30
31     cout<<endl;
32 }
```

```
$ c++ -o initVector initVector.cpp
$
$ ./initVector
6
3.1416
3.1416
3.1416
3.1416
3.1416
3.1416

$
```

En las líneas 13 y 14 que están comentadas, vemos que el comentario dice error, efectivamente si quisieramos compilar con dichas lineas activas veriamos el siguiente mensaje en consola:

```
$ c++ -o initVector initVector.cpp
initVector.cpp: In function 'int main()':
initVector.cpp:12:14: error: no match for 'operator=' (operand types are 'std::vector<double>' and 'int')
     Vector_1 = 10;
              ^
initVector.cpp:12:14: note: candidate is:
```

```
In file included from /usr/include/c++/4.9/vector:69:0,
                 from initVector.cpp:2:
/usr/include/c++/4.9/bits/vector.tcc:167:5: note: std::vector<_Tp, _Alloc>& std::vector<_Tp, _Alloc>::operator=(
     const std::vector<_Tp, _Alloc>&) [with _Tp = double; _Alloc = std::allocator<double>]
     vector<_Tp, _Alloc>::
     ^
/usr/include/c++/4.9/bits/vector.tcc:167:5: note:   no known conversion for argument 1 from 'int' to 'const std::
     vector<double>&'
initVector.cpp:13:5: error: 'Vector' was not declared in this scope
     Vector[0]= 1;
     ^


$
```

La forma de inicializar al vector es como se muestra en las lineas siguientes a los comentarios de error.

1.1.2. Copia de un vector. Estas estructuras, a pesar de ser complejas, poseen sobrecargados los operadores, eso implica que están redefinidos para que sean usados de modo transparentes, directos.

```
1  //copiaVector.cpp
2  #include <iostream>
3  #include <vector>
4
5  using namespace std;
6
7  int main()
8  {
9          //
10     // copia de vectores...
11     // declaro e inicializo un vector con valores en 1
12     vector<int> vInt_1(20, 1);
13
14     // No se inicializa
15     vector<int> vInt_2;
16
17     cout<< "vInt_2 antes: " << vInt_2.size() <<endl;
18
19     // Uso el igual para asignar, para hacer la copia
20     vInt_2 = vInt_1;
21
22     cout<< "vInt_2 despues: " << vInt_2.size() <<endl;
23     for(int i=0; i < vInt_2.size() ;i++)
24     {
25         cout << "vInt_2 = " << vInt_2[i] << endl;
26     }
27 }
```

```
$ c++ -o copiaVector copiaVector.cpp
$
$./copiaVector
vInt_2 antes: 0
vInt_2 despues: 20
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1


$
```

1.1.3. Insertando y eliminando datos. Los ejemplos que se mostrarán no son exhaustivos, pero darán una base de como se manipulan los datos en el vector.

```
1
2   #include <iostream>
3   #include <vector>
4
5   using namespace std;
6
7   int main()
8   {
9       vector<int> vInt_2(10,1);
10      //push_back() es el que permite ingresar un nuevo elemento
11      //al vector
12      //
13      cout << "ñTamao de vInt_2 = " << vInt_2.size() << endl;
14
15      vInt_2.push_back(5);
16
17      cout << "Nuevo ñtamao de vInt_2 = " << vInt_2.size() << endl;
18
19      for(int i=0; i < vInt_2.size() ;i++)
20      {
21          cout << "vInt_2 = " << vInt_2[i] << endl;
22      }
23
24      //pop_back() eliminamos el último elemento
25      //Esta ófuncin no lo retorna
26      //
27      vInt_2.pop_back();
28
29      for(int i=0; i < vInt_2.size() ;i++)
30      {
31          cout << "vInt_2 = " << vInt_2[i] << endl;
32      }
33
34      cout << "ñTamao de vInt_2 = " << vInt_2.size() << endl;
35      vInt_2.resize(5);
36      //ahora su tamano es
37      cout<<"El nuevo tamano es: " << vInt_2.size()<<endl;
38
39      //Esto ha provocado una perdida de óinformacin
40      //
41      for(int i=0; i < vInt_2.size() ;i++)
42      {
43          cout << "vInt_2 = " << vInt_2[i] << endl;
44      }
45
46      // Borrado de elementos
47      //
48      vector<int> vInt_3(10);
49
50      for (int i = 0; i < 10; i++)
51          vInt_3[i] = i+1;
52
53      cout << "El ñtamao de vInt_3 es " << vInt_3.size() << endl;
54      cout << "--------------" << endl;
55      for(int i=0; i < vInt_3.size() ;i++)
56      {
57          cout << "vInt_3 = " << vInt_3[i] << endl;
58      }
59
60      vInt_3.erase(vInt_3.begin()+3, vInt_3.begin()+6);
61      cout << "--------------" << endl;
62      for(int i=0; i < vInt_3.size() ;i++)
63      {
64          cout << "vInt_3 = " << vInt_3[i] << endl;
65      }
66      cout << "El ñtamao despues del borrado de vInt_3 es " << vInt_3.size() << endl;
```

```
67
68          for(int i=0; i < vInt_3.size() ;i++)
69          {
70              cout << "vInt_3 = " << vInt_3[i] << endl;
71          }
72
73          // IMPORTANTE erase() no recibe elementos sino posiciones!!!!
74          //
75          // insert permite agregar elementos en cualquier parte del vector
76          //
77          vInt_3.insert(vInt_3.begin()+3, 3);
78          cout << endl;
79          for(int i=0; i < vInt_3.size() ;i++)
80          {
81              cout << "vInt_3 = " << vInt_3[i] << endl;
82          }
83
84          // otra forma (de varias)
85          //
86          vInt_3.insert(vInt_3.begin()+4,3,-1);
87          cout << endl;
88          for(int i=0; i < vInt_3.size() ;i++)
89          {
90              cout << "vInt_3 = " << vInt_3[i] << endl;
91          }
92      }
```

Como se ve, en los comentarios al codigo se hace referencia o se explica como proceder o las consecuencias de lo que se realiza. Al ejecutar el código vemos:

```
$ c++ -o push_popVector push_popVector.cpp
$
$./push_popVectorñ
Tamao de vInt_2 = 10
Nuevo ñtamao de vInt_2 = 11
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 5
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1ñ
Tamao de vInt_2 = 10
El nuevo tamano es: 5
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
vInt_2 = 1
El ñtamao de vInt_3 es 10
--------------
vInt_3 = 1
vInt_3 = 2
vInt_3 = 3
vInt_3 = 4
vInt_3 = 5
vInt_3 = 6
vInt_3 = 7
vInt_3 = 8
vInt_3 = 9
vInt_3 = 10
--------------
vInt_3 = 1
vInt_3 = 2
vInt_3 = 3
```

```
vInt_3 = 7
vInt_3 = 8
vInt_3 = 9
vInt_3 = 10
El ñtamao despues del borrado de vInt_3 es 7
vInt_3 = 1
vInt_3 = 2
vInt_3 = 3
vInt_3 = 7
vInt_3 = 8
vInt_3 = 9
vInt_3 = 10

vInt_3 = 1
vInt_3 = 2
vInt_3 = 3
vInt_3 = 3
vInt_3 = 7
vInt_3 = 8
vInt_3 = 9
vInt_3 = 10

vInt_3 = 1
vInt_3 = 2
vInt_3 = 3
vInt_3 = 3
vInt_3 = -1
vInt_3 = -1
vInt_3 = -1
vInt_3 = 7
vInt_3 = 8
vInt_3 = 9
vInt_3 = 10

$
```

1.1.4. Función sort. Para el uso de ciertas funciones, debemos incluir la lib *algorithm*

```
\\sortVector.cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{

    vector<int> vInt_3(10);

    for (int i = 0, j = 10; i < 10; i++, j--)
        vInt_3[i] = j+1;

    cout << "El ñtamao de vInt_3 es " << vInt_3.size() << endl;
    cout << "--------------" << endl;
    for(int i=0; i < vInt_3.size() ;i++)
    {
        cout << "vInt_3 = " << vInt_3[i] << endl;
    }

    // Con los contenedores podemos usar una ófuncin
   // de la lib algorithm muy útil. Antes hay que
   // incluir la lib #include<algortihm>
   //
   sort(vInt_3.begin(), vInt_3.end());

   cout << endl;
   for(int i=0; i < vInt_3.size() ;i++)
   {
       cout << "vInt_3 = " << vInt_3[i] << endl;
   }
   vector<string> Nombres;
   Nombres.push_back("Pablo");
   Nombres.push_back("Juan");
   Nombres.push_back("alicia");
   for(int i=0; i < Nombres.size() ;i++)
```

```
    {
        cout << Nombres[i] << " ";
    }
    cout << endl;

    sort(Nombres.begin(), Nombres.end());

    for(int i=0; i < Nombres.size() ;i++)
    {
        cout << Nombres[i] << " ";
    }
    cout << endl;
}
```

```
$ c++ -o sortVector sortVector.cpp
$
$./sortVector
El ñtamao de vInt_3 es 10
--------------
vInt_3 = 11
vInt_3 = 10
vInt_3 = 9
vInt_3 = 8
vInt_3 = 7
vInt_3 = 6
vInt_3 = 5
vInt_3 = 4
vInt_3 = 3
vInt_3 = 2

vInt_3 = 2
vInt_3 = 3
vInt_3 = 4
vInt_3 = 5
vInt_3 = 6
vInt_3 = 7
vInt_3 = 8
vInt_3 = 9
vInt_3 = 10
vInt_3 = 11
Pablo Juan alicia
Juan Pablo alicia
$
```

1.1.5. Iteradores. Los iteradores son punteros. Están asociados a los contenedores y nos ayudarán en el momento de sus recorridos principalmente.

```
//itVector.cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
        vector<int> vInt_3(10);

    for (int i = 0, j = 10; i < 10; i++, j--)
        vInt_3[i] = j+1;

    cout << "El ñtamao de vInt_3 es " << vInt_3.size() << endl;
    cout << "--------------" << endl;
    for(int i=0; i < vInt_3.size() ;i++)
    {
        cout << "vInt_3 = " << vInt_3[i] << endl;
    }

   // Iteradores
   //
   // Son punteros
   vector<int>::iterator I;

```

```
27      I = vInt_3.begin();
28
29      cout << "vInt_3[0] = " << *I << endl;
30
31       // como se ve, begin() retorna un iterador
32       // entonces podemos hacer lo siguiente:
33       //
34       vector<int>::iterator I1, I2;
35       I1 = vInt_3.begin();
36       I2 = vInt_3.end();
37
38       cout<< endl << "Ordenado a traves de iteradores" << endl;
39
40       sort(I1, I2);
41
42       for( ; I1 != I2 ;I1++)
43       {
44           cout << "vInt_3 = " << *I1 << endl;
45       }
46
47       return 0;
48    }
```

```
$ g++ -o itVector itVector.cpp
./itVector
El ñtamao de vInt_3 es 10
--------------
vInt_3 = 11
vInt_3 = 10
vInt_3 = 9
vInt_3 = 8
vInt_3 = 7
vInt_3 = 6
vInt_3 = 5
vInt_3 = 4
vInt_3 = 3
vInt_3 = 2
vInt_3[0] = 11

Ordenado a traves de iteradores
vInt_3 = 2
vInt_3 = 3
vInt_3 = 4
vInt_3 = 5
vInt_3 = 6
vInt_3 = 7
vInt_3 = 8
vInt_3 = 9
vInt_3 = 10
vInt_3 = 11

$
```