Vectores

Algoritmos y Estructuras de Datos I

17 de Septiembre de 2018

Tipos de datos de C++

- Como ya hemos visto, en C++ tenemos tipos de datos que implementan (en algunos casos parcialmente) cada uno de los tipos de datos del lenguaje de especificación:
  - El tipo `int` para números enteros ($\mathbb{Z}$)
  - El tipo `float` para números reales ($\mathbb{R}$)
  - El tipo `bool` para valores booleanos (Bool)
  - El tipo `char` para caracteres (Char)

Tipos de datos de C++

- Como ya hemos visto, en C++ tenemos tipos de datos que implementan (en algunos casos parcialmente) cada uno de los tipos de datos del lenguaje de especificación:
  - El tipo `int` para números enteros ($\mathbb{Z}$)
  - El tipo `float` para números reales ($\mathbb{R}$)
  - El tipo `bool` para valores booleanos (Bool)
  - El tipo `char` para caracteres (Char)

¿Y las secuencias?

Vectores en C++

- Un vector en C++ es una **estructura de datos** con las siguientes propiedades:
  1. Cada valor está identificado por un índice.
  2. Todos los valores son del mismo tipo.
  3. Nuevos elementos pueden agregarse.
  4. Elementos existentes pueden eliminarse.
- Una estructura de datos es una **colección** de datos en memoria, con una organización predeterminada.
  1. Generalmente, esta organización facilita operaciones sobre la **colección**
  2. Ejemplo: Agregar un elemento, consultar un elemento, consultar el valor mínimo, etc.
- El vector de C++ es una implementación de las secuencias del lenguaje de especificación ($seq\langle T \rangle$)

Ejemplos de definiciones de vectores:

```
1 vector<int> cuenta;
2 vector<float> decimales;
3 vector<vector<int> > matriz;
```

Vectores en C++

```
1  vector<int> cuenta; //declara un vector de int
2  vector<float> decimales; //declara un vector de float
3  vector<vector<int> > matriz; //declara un vector de vectores de int
```

- Para usar vectores en C++, hay que incluir la biblioteca con `#include <vector>`
- Se pueden definir vectores con elementos de cualquier tipo de datos (incluso otros vectores).
- El tipo de un vector se escribe como el tipo de los valores que contiene, encerrado en <...>.
- Inicialmente, el vector no contiene ningún elemento.

Ejemplo

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  int main ()
7  {
8    vector<int> vacio;
9    vector<int> inicializado(4);
10   vector<int> inicializados(4,100);
11
12   return 0;
13 }
```

Declaración de un vector en C++

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

using namespace std;

int main ()
{
  vector<int> vacio; // vector de enteros vacio
  vector<int> inicializado(4); // <0, 0, 0, 0>
  vector<int> inicializados(4,100); // <100, 100, 100, 100>

  return 0;
}
```

Los vectores se pueden inicializar con una cantidad determinada de elementos y podemos definir cual va a ser el valor inicial de cada uno.

Ejemplo

```
1  #include <vector>
2  using namespace std;
3
4  int main() {
5    vector<int> cuenta;
6    cuenta.push_back(1);
7    cuenta.push_back(2);
8    cuenta.push_back(3);
9    cuenta.push_back(4);
10   return 0;
11 }
```

Agregar elementos al vector

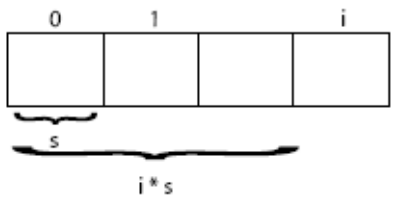
```cpp
#include <vector>
using namespace std;

int main() {
  vector<int> cuenta; // crea el vector vacio
  cuenta.push_back(1); // el vector contiene la secuencia <1>
  cuenta.push_back(2); // el vector contiene la secuencia <1,2>
  cuenta.push_back(3); // el vector contiene la secuencia <1,2,3>
  cuenta.push_back(4); // el vector contiene la secuencia <1,2,3,4>
  return 0;
}
```

- La forma más sencilla de agregar elementos al vector es utilizando la operación push_back.
- push_back modifica el vector agregando el elemento al final.

Leer elementos almacenados en un vector

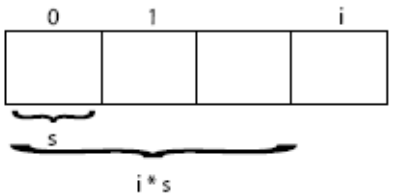
- Internamente, los elementos se guardan en memoria en forma consecutiva.
- Si cada elemento ocupa s bytes, entonces el elemento en la posición i se encuentra en la posición $i \times s$ después del inicio:



- Obtener un elemento cualquiera tiene un tiempo de ejecución constante, independientemente del tamaño del vector.

Leer elementos almacenados en un vector

- Si internamente los elementos se guardan en memoria en forma consecutiva, ¿qué pasa cuando se ejecuta un `push_back()`?



- Cada vez que se ejecuta `push_back`, es posible que internamente el vector deba ser copiado a otra porción de la memoria .
- Esta copia la realiza internamente la biblioteca `<vector>`.

Ejemplo

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> s;
  s.push_back(1);
  s.push_back(2);
  int valor1 = s[0];
  int valor2 = s[1];
  cout << valor1 << endl;
  cout << valor2 << endl;
  return 0;
}
```

Almacenar y Leer elementos

```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
  vector<int> s;
  s.push_back(1); // el vector contiene la secuencia <1>
  s.push_back(2); // el vector contiene la secuencia <1,2>
  int valor1 = s[0]; // lee el valor almacenado en <1,2>[0]
  int valor2 = s[1]; // lee el valor almacenado en <1,2>[1]
  cout << valor1 << endl; // Imprime 1
  cout << valor2 << endl; // Imprime 2
  return 0;
}
```

Leer elementos almacenados en un vector

- En C++, para acceder a un elemento del vector se debe escribir el nombre del vector seguido del índice entre corchetes.
- La expresión `cuenta[0]` es el primer elemento del vector, `cuenta[1]` el segundo, etc.

Ejemplo

```
1  vector<double> vectorDeReales;
2  int size0 = vectorDeReales.size();
3  vectorDeReales.push_back(1.5);
4  int size1 = vectorDeReales.size();
5  vectorDeReales.push_back(2.5);
6  int size2 = vectorDeReales.size();
```

Longitud del vector

```
1  vector<double> vectorDeReales;
2  int size0 = vectorDeReales.size(); // Longitud ==0
3  vectorDeReales.push_back(1.5);
4  int size1 = vectorDeReales.size(); // Longitud ==1
5  vectorDeReales.push_back(2.5);
6  int size2 = vectorDeReales.size(); // Longitud ==2
```

- Dado un vector, podemos obtener su longitud utilizando la operación `size`
- `size` implementa la función *length* (notación $|.|$) de secuencias.

Ejemplo

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6    vector<int> s;
7    s.push_back(1);
8    int valor1 = s[0];
9    s[0] = 351;
10   int valor2 = s[0];
11   cout << valor1 << endl; // cuanto vale valor1?
12   cout << valor2 << endl; // cuanto vale valor2?
13   return 0;
14 }
```

Ejemplo

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6    vector<int> s;
7    s.push_back(1);
8    int valor1 = s[0];
9    s[0] = 351;
10   int valor2 = s[0];
11   cout << valor1 << endl; // cuanto vale valor1? Rta: 1
12   cout << valor2 << endl; // cuanto vale valor2? Rta: 351
13   return 0;
14 }
```

- Para *reemplazar* un elemento del vector se usa la misma sintaxis, pero el vector y su posición se escriben en la parte izquierda de la asignación

Leer elementos almacenados en un vector

- Los elementos de un vector pueden ser utilizados como si fueran una variable:

```
1  cuenta[0] = 7;
2  cuenta[1] = cuenta[0] * 2;
3  cuenta[2] = 0;
4  cuenta[2] = cuenta[2]+1;
5  cuenta[3] = -60;
```

- ¿Cuál es el resultado después de ejecutar el código anterior?

Leer elementos almacenados en un vector

- Los elementos de un vector pueden ser utilizados como si fueran una variable:

```
1  cuenta[0] = 7;
2  cuenta[1] = cuenta[0] * 2;
3  cuenta[2] = 0;
4  cuenta[2] = cuenta[2]+1;
5  cuenta[3] = -60;
```

- ¿Cuál es el resultado después de ejecutar el código anterior?

`cuenta == <7, 14, 1, -60>`

Ejemplo

```
1  vector<char> s;
2  s.push_back('H');
3  s.push_back('o');
4  s.push_back('l');
5  s.push_back('a');
6  s.pop_back();
7  s.pop_back();
```

Eliminar una posición

```
1  vector<char> s;
2  s.push_back('H');
3  s.push_back('o');
4  s.push_back('l');
5  s.push_back('a'); // contiene la secuencia <'H','o','l','a'>
6  s.pop_back(); // contiene la secuencia <'H','o','l'>
7  s.pop_back(); // contiene la secuencia <'H','o' >
```

- Dado un vector, podemos eliminar la última posición válida con la operación `pop_back`.
- Al hacer `push_back` la longitud crece
- Al hacer `pop_back` la longitud decrece.

Posiciones no válidas

¿Qué ocurre cuando por error queremos acceder a una posición del vector que no está definida?

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6    vector<int> cuenta;
7    cuenta.push_back(1); // el vector contiene la secuencia <1>
8    cuenta[20000] = 10; // ?
9    int valor_prueba = cuenta[20000]; // ?
10   cout << "El valor de valor_prueba es " << valor_prueba << endl;
11   return 0;
12 }
```

Acceso no válido

- Resultado en la máquina 1:

```
El valor de valor_prueba es: 10
Process finished with exit code 0
```

- Resultado en la máquina 2:

```
Process finished with exit code 139
(interrupted by signal 11: SIGSEGV)
```

Acceso no válido

- Resultado en la máquina 1:

```
El valor de valor_prueba es: 10
Process finished with exit code 0
```

- Resultado en la máquina 2:

```
Process finished with exit code 139
(interrupted by signal 11: SIGSEGV)
```

- Lamentablemente, C++ no define qué ocurre cuando accedemos a posiciones fuera de rango
- Es decir, el comportamiento está **indefinido** (a veces, da resultados razonables, pero otras veces no).
- Algunos posibles resultados al leer o escribir una posición fuera de rango en C++:
  - Puede generar un `segmentation fault` y terminar la ejecución del programa.
  - Puede leer/escribir una posición cualquiera de la memoria. La ejecución continúa pero dañamos la integridad del sistema.

Copiar vectores

- ¿Cómo copiamos un vector a en otro vector b?

Copiar vectores

- ¿Cómo copiamos un vector a en otro vector b?
- Opción 1: Copiar elemento a elemento.

```
1  vector<double> b;
2  for(int i=0; i<a.size(); i=i+1) {
3    b.push_back(a[i]);
4  }
```

Copiar vectores

- ¿Cómo copiamos un vector a en otro vector b?
- Opción 1: Copiar elemento a elemento.

```
vector<double> b;
for(int i=0; i<a.size(); i=i+1) {
  b.push_back(a[i]);
}
```

- Opción 2: Usar el operador de asignación =.

```
vector<double> b;
b = a;
```

- Ambas opciones tienen el mismo resultado.

Retorno de vectores

- ¿Cómo declaramos una función que retorne un vector?

Retorno de vectores

- ¿Cómo declaramos una función que retorne un vector?

```
1  vector<int> funcionQueRetornaVector(vector<int> v)
2  {
3          vector<int> res;
4
5          ...
6
7          return res;
8  }
```

- Internamente: el vector pasado como parámetro se va a copiar a otro vector.
- dentro del código de la función declaramos un `vector<int>` y lo retornamos con `return`.

Vectores como Parámetros de Funciones

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  void cambiarVector(vector<int> a) {
6    a[0]=35;
7  }
8
9  int main() {
10   vector<int> b;
11   b.push_back(5);
12   cout << "Antes:" << b[0] << endl;
13   cambiarVector(b);
14   cout << "Despues:" << b[0] << endl; // que imprime? 5 o 35?
15   return 0;
16 }
```

Vectores como Parámetros de Funciones

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  void cambiarVector(vector<int> a) {
6    a[0]=35;
7  }
8
9  int main() {
10   vector<int> b;
11   b.push_back(5);
12   cout << "Antes:" << b[0] << endl;
13   cambiarVector(b);
14   cout << "Despues:" << b[0] << endl; // que imprime? 5 o 35?
15   return 0;
16 }
```

Rta: Imprime 5 ya que se modificó una copia del vector original
Del mismo modo que cuando se retorna un vector, cuando un vector se pasa por parámetro, por defecto se pasa por copia.

Vectores como Parámetros de Funciones

```cpp
#include <iostream>
#include <vector>
using namespace std;

void cambiarVector(vector<int>& a) {
  a[0]=35;
}

int main() {
  vector<int> b;
  b.push_back(5);
  cout << "Antes:" << b[0] << endl;
  cambiarVector(b);
  cout << "Despues:" << b[0] << endl; // que imprime? 5 o 35?
  return 0;
}
```

Vectores como Parámetros de Funciones

```cpp
#include <iostream>
#include <vector>
using namespace std;

void cambiarVector(vector<int>& a) {
  a[0]=35;
}

int main() {
  vector<int> b;
  b.push_back(5);
  cout << "Antes:" << b[0] << endl;
  cambiarVector(b);
  cout << "Despues:" << b[0] << endl; // que imprime? 5 o 35?
  return 0;
}
```

Rta: Imprime 35 ya que se modificó el vector original

Retorno de vectores

- ¿Cómo declaramos una función que modifique un vector?

Retorno de vectores

- ¿Cómo declaramos una función que modifique un vector?

```
1  void funcionQueModificaVector(vector<int>& v)
2  {
3    ...
4  }
```

- tenemos un vector pasado en algún parámetro como una referencia.

Resumen: Vectores en C++

- `vector<int> a;` Declara un nuevo vector sin elementos
- `vector<int> a(n);` Declara un nuevo vector con n elementos.
- `vector<int> a(n, x);` Declara un nuevo vector con n elementos inicializados en x.
- `a.push_back(n);` Almacena el valor n al final del vector
- `a[i] = n;` Reemplaza la posición i del vector con el valor n
- `int b = a[i];` Lee la posición i del vector
- `a.pop_back();` Elimina la última posición del vector
- `a.size();` Informa la longitud del vector
- `v1 = v2;` Borra todas las posiciones de v1 y las reemplaza copiando los elementos de v2. v1 y v2 deben tener el mismo **tipo**.

Ejercicios

Resolver los ejercicios de la sección Vectores.