Vectores – Entrada/Salida desde archivos

Algoritmos y Estructuras de Datos I

Repaso: Operaciones sobre secuencias

- $length(a : seq\langle T\rangle) : \mathbb{Z}$ (notación $|a|$)
- Indexación: $seq\langle T\rangle[i : \mathbb{Z}] : T$
- Igualdad: $seq\langle T\rangle = seq\langle T\rangle$
- $head(a : seq\langle T\rangle) : T$
- $tail(a : seq\langle T\rangle) : seq\langle T\rangle$
- $addFirst(t : T, a : seq\langle T\rangle) : seq\langle T\rangle$
- $concat(a : seq\langle T\rangle, b : seq\langle T\rangle) : seq\langle T\rangle$ (notación a++b)
- $subseq(a : seq\langle T\rangle, d, h : \mathbb{Z}) : \langle T\rangle$
- $setAt(a : seq\langle T\rangle, i : \mathbb{Z}, val : T) : seq\langle T\rangle$

Vectores en C++

- 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.
- 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.
- El vector de C++ es una implementación de las secuencias del lenguaje de especificación ($seq\langle\mathbb{Z}\rangle$, etc.)

Declaración de un vector en C++

- 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 $< \cdots >$.
- Por ejemplo, declaramos un vector para almacenar enteros y otro vector para almacenar reales (`doubles`):

```
1 vector<int> cuenta;
2 vector<double> decimales;
```

- Inicialmente, el vector no contiene ningún elemento.

Declaración - Especificación

- Cuál es la especificación de la operación en C++ que crea un nuevo vector vacío (llamemosla newVector)?

proc newVector(out result: $seq\langle T \rangle$) {
  Pre { True }
  Post { $|result| = 0$ }
}

Agregar elementos al vector

- 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.
- Ejemplo:

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

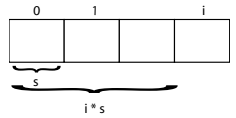
push_back - Especificación

- ¿Cuál es la especificación de la operación en C++ que agrega un elemento al final del vector?

proc push_back(inout s: $seq\langle T \rangle$, in value: T) {
  Pre { $s = S_0$ }
  Post { $(|s| = |S_0| + 1 \wedge_L s[|S_0|] = value)$
    $\wedge subseq(s, 0, |S_0|) = S_0$}
}

Leer elementos almacenados en un vector

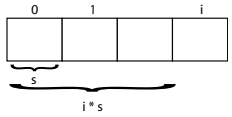
- Internamente, los elementos se guardan en memoria en forma consecutiva.
- Si cada elemento ocupa s bytes, entonces el elemento en la posición i se encuentra en la posición $i \times s$ después del inicio:



- 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>`.

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.
- Veámoslo con un ejemplo:

Demo #1: Almacenar y Leer elementos

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6    vector<int> s;
7    s.push_back(1); // el vector contiene la secuencia <1>
8    s.push_back(2); // el vector contiene la secuencia <1,2>
9    int valor1 = s[0]; // lee el valor almacenado en <1,2>[0]
10   int valor2 = s[1]; // lee el valor almacenado en <1,2>[1]
11   cout << valor1 << endl; // cuanto se imprime por consola?
12   cout << valor2 << endl; // cuanto se imprime por consola?
13   return 0;
14 }
```

Leer una posición - Especificación

- ¿Cual es una especificación para la operación que lee un elemento del vector (llamémosla readVector)?

proc readVector(in s: $seq\langle T \rangle$, in i: $\mathbb{Z}$, out result: T) {
  Pre { $0 \leq i < |s|$ }
  Post { $result = s[i]$ }
}

Reemplazar un elemento

- 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
- Veamos otro ejemplo.

Demo #2: Reemplazar un elemento

```
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 }
```

Especificación de operaciones de vectores

- ¿Cuál es una especificación para la operación de reemplazo de una posición?
- Ejemplo:

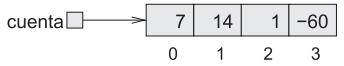
proc writeVector(inout s: $seq\langle T \rangle$, in i: $\mathbb{Z}$, in v: T) {
  Pre { $0 \leq i < |s| \wedge s = S_0$ }
  Post { $s = setAt(S_0, i, v)$}
}

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?



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

Demo #3: Posiciones no válidas

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  int main() {
6    cout << "Hola!" << endl;
7    vector<int> cuenta;
8    cuenta.push_back(1); // el vector contiene la secuencia <1>
9    cuenta[2000] = 10; // ?
10   int valor = cuenta[2000]; // ?
11   cout << "El valor de valor es " << valor << endl;
12   cout << "Chau!" << endl;
13   return 0;
14 }
```

Acceso no válido

- Resultado de la ejecución:

```
Hola!
El valor de valor es 10
Chau!

Process finished with exit code 0
```

- Lamentablemente, C++ no define qué ocurre cuando accedemos a posiciones fuera de rango
- Es decir, el comportamiento está **indefinido**.

Vectores: acceso fuera de rango en C++

- Algunos posibles resultados al leer o escribir una posición fuera de rango en C++:
  - Puede dar un error (exception) durante la ejecución
  - 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.
- En otras palabras, recordar siempre que:
  - la `precondición` de leer o escribir una posición (`[...]`) es que la posición sea válida

Longitud del vector

- Dado un vector, podemos obtener su longitud utilizando la operación `size`
- Ejemplo:

```
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
```

- `size` implementa la función *length* (notación $|.|$) de secuencias.

Especificación de operaciones de vectores

Obtener la longitud del vector

- ¿Cuál es una especificación de la función `size` de vectores?

  proc size(in s: $seq\langle T \rangle$, out result: $\mathbb{Z}$) {
  Pre { True }
  Post { result= $|s|$ }
  }

Eliminar una posición

- Dado un vector, podemos eliminar la última posición válida con la operación `pop_back`.
- 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'); // 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' >
```

- Al hacer `push_back` la longitud crece
- Al hacer `pop_back` la longitud decrece.

Eliminar una posición - Especificación

- ¿Cuál es una especificación para la operación de eliminación de la última posición del vector?

  proc pop_back(inout s: $seq\langle T \rangle$) {
  Pre { $s = S_0 \wedge |s| > 0$ }
  Post { $s = subseq(S_0, 0, |S_0| - 1)$}
  }

- ¿Qué ocurre si ejecutamos `pop_back()` sobre un vector sin elementos?

Demo #4: pop_back de un vector sin elementos

```
1  #include <vector>
2  #include <iostream>
3  using namespace std;
4
5  int main() {
6    cout << "Hola!" << endl;
7    vector<int> v;
8    v.push_back(1);
9    v.pop_back();
10   v.pop_back(); // que pasa al ejecutar este comando?
11   cout << "Chau!" << endl;
12   return 0;
13 }
```

Demo #4: pop_back de un vector sin elementos

```
Hola!
Chau!

Process finished with exit code 15
```

- Se ejecuta hasta el final pero se "cuelga".
- ¿Por qué?
  - Porque falla la función que libera la memoria cuando termina de usarse el vector.
- Moraleja: No cumplir con la precondición puede tener consecuencias MUY, pero MUY difíciles de predecir en C++.

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 }
```

- Opción 2: Usar el operador de asignación =.

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

- Ambas opciones tienen el mismo resultado.

Retorno de vectores

- ¿Cómo declaramos una función que retorne un vector?

```
1 vector<int> funcionQueRetornaVector(int n)
2 {
3   ...
4 }
```

- Dentro del código de la función declaramos un "vector< *int* >" y lo retornamos con `return`.
- Internamente: el vector se va a copiar a otro vector.

Retorno de vectores

- Especificar el problema `crearVectorN` que, dado un número que no sea negativo, retornar un vector de enteros de esa longitud donde todos los elementos son 0.
- proc $crearVectorN(\text{in } n : \mathbb{Z}, \text{out } result : seq\langle\mathbb{Z}\rangle)\{$
  Pre $\{n \geq 0\}$
  Post $\{|result| = n \wedge \#apariciones(result, 0) = n\}$
  $\}$
- ¿Cómo lo implementamos en C++?

Retorno de vectores

- Implementación:

```
1  vector<int> crearVectorN(int n) {
2    vector<int> v;
3    for (int i=0; i<n;i=i+1) {
4      v.push_back(0);
5    }
6    return v;
7  }
```

Demo #5: Retorno de vectores

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4
5  vector<int> crearVectorN(int n) {
6    vector<int> v;
7    for (int i=0; i<n;i=i+1) {
8      v.push_back(0);
9    }
10   return v;
11 }
12
13 int main() {
14   vector<int> new_vector = crearVector(10);
15   cout << "new_vector[0]=" << new_vector[0];
16   return 0;
17 }
```

Retorno de vectores

- Resultado de la ejecución:

```
new_vector[0]=0

Process finished with exit code 0
```

- ¿Por qué?
  - Porque se realizó una copia del vector retornado por `crearVectorN` a `new_vector` debido a la asignación (=) entre vectores en C++.

Demo #6: Vectores como Parámetros de Funciones

- Del mismo modo que cuando se retorna un vector, cuando un vector se pasa por parámetro, por defecto se pasa por copia.

```
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

- Rta: Imprime 5 y ya que se modificó una copia del vector original
- La clase que viene veremos como pasar un vector por referencia en lugar de por copia

Sumar los elementos de una secuencia

- Sea la siguiente especificación para sumar todos los elementos de una secuencia de enteros.

  proc $sumar(\text{in } s : seq\langle\mathbb{Z}\rangle, \text{out } result : \mathbb{Z})\{$
  Pre $\{True\}$
  Post $\{result = \sum_{i=0}^{|s|-1} s[i]\}$
  $\}$

- ¿Cuál puede ser una implementación (programa) posible que satisfaga la especificación?

Sumar los elementos de una secuencia

- Recorremos la secuencia con una variable de control y vamos guardando en un acumulador la suma de los elementos recorridos.
- Solución usando `while`:

```
1  int suma(vector<int> s) {
2    int result = 0;
3    int i =0;
4    while( i < s.size() ) {
5      result = result + s[i];
6      i = i + 1;
7    }
8    return result;
9  }
```

- También se puede implementar usando `for`.

Resumen: Vectores en C++

`vector<int> a;`	Declara un nuevo vector sin elementos
`a.push_back(7);`	Almacena el valor 7 al final del vector
`a[0] = 7;`	Reemplaza la posición 0 del vector con el valor 7
`int b = a[0];`	Lee la posición 0 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**.

Intervalo

Break!

Entrada/salida desde archivos

- La entrada/salida de datos con archivos es similar a la E/S por consola.
- Con la *consola* (i.e. teclado y pantalla) hacemos ...

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5    int a = 0;
6    cout << "Ingrese un entero: "; // Salida x consola
7    cin >> a; // Entrada x consola
8    cout << "El numero ingresado es: " << a; // Salida x consola
9    return 0;
10 }
```

E/S con archivos

- El `cout` es un **stream** de salida para imprimir por pantalla un valor de un tipo de datos y el `cin` es un **stream** de entrada para leer del teclado un valor de un tipo de datos.
- Leemos o escribimos archivos a través de `ifstream` (para leer) y de `ofstream` (para escribir). Hay dos tipos de iostream:
  - ofstream: para escribir (write/salida)
  - ifstream: para leer (read/entrada)
- Escribir texto en un archivo de texto plano en C++ es similar a escribir texto por consola.

Escribir valores en un Archivo

- Para escribir y leer archivos tenemos que incluir (además de `iostream`) la biblioteca `fstream`.
- Debemos seguir el siguiente protocolo:
  - Declarar un `ofstream`
  - Abrir el archivo en modo escritura (`open`)
  - Escribir (1 o más veces) (usar <<)
  - Cerrar el archivo (¿por qué?) (`close`)
- Veámoslo con un ejemplo.

Demo #7: Escribir valores a un Archivo

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int main() {
6    int a = 5;
7    ofstream fout;
8    fout.open("archivo.txt"); // abre el archivo para escritura
9    fout << "Hola, archivo!" << endl;
10   fout << "Ahora, un entero: " << a << endl;
11   fout << "Tambien una expresion: " << (a+2) << endl;
12   fout.close(); // cerramos el archivo
13   return 0;
14 }
```

Escribir valores en un Archivo

- Contenido final de `archivo.txt`:

```
Hola, archivo!
Ahora, un entero: 5
Tambien una expresion: 7
```

- Otra visualización (usando $\backslash n$)

```
Hola, archivo!\nAhora, un entero: 5\nTambien una expresion: 7
```

Escribir valores de distintos Tipos de datos

- Hasta ahora escribimos únicamente enteros (`int`)
- También podemos escribir valores `bool`, `float`, `char`, etc.
- Por ejemplo:

Escribir valores de distintos Tipos de datos

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int main() {
6    char c = 'x';
7    float f = 1.5;
8    bool b = true;
9    ofstream fout;
10   fout.open("archivo.txt"); // abre archivo
11   fout << c << endl; // escribe x (char)
12   fout << f << endl; // escribe 1.5 (float)
13   fout << b << endl; // escribe 1 (bool)
14   fout.close(); // cierra archivo
15   return 0;
16 }
```

Ejemplo: escribir archivos

- Escribir una función `writeToFile` que escriba en un archivo `salida.txt` 2 enteros `a` y `b` y luego 2 reales `f` y `g` separados con coma en una única línea.
- Solución:

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4  void writeToFile(int a, int b, float f, float g) {
5    ofstream fout;
6    fout.open("salida.txt");
7    fout << a << ',';
8    fout << b << ',';
9    fout << f << ',';
10   fout << g << ',' << endl;
11   fout.close();
12 }
```

Escribir al final de un archivo existente

- ¿Qué hace la operación `ofstream.open(‘‘archivo.txt’’)` si `archivo.txt` ya existe?
  - Si no existe, crea el archivo
  - Si existe, sobreescribe todo su contenido (borra lo que había antes)
- ¿Cómo podemos hacer para que el contenido anterior sea respetado?
  - Para escribir al final del archivo hay que abrirlo en modo append
  - Para abrir un archivo en modo append, hay que usar `ofstream.open(‘‘archivo.txt’’,ios_base::app)`

Leer datos desde un archivo

- Para leer declaramos un `ifstream` y usamos el operador >>.
- Es importante conocer de antemano el orden y tipo de datos de los elementos a leer del archivo.
- Debemos seguir el siguiente protocolo:
  - Declarar un `ifstream`
  - Abrir el archivo en modo lectura (`open`)
  - Leer (1 o más veces) (usando >>)
  - Cerrar el archivo (¿por qué?) (`close`)
- Ejemplo: Leer un archivo `entrada.txt` que contiene dos enteros separados por un espacio en blanco (ejemplo: “15 20”)

Demo #8: Leer datos desde un archivo

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int main() {
6    int a =0;
7    int b =0;
8    ifstream fin;
9    fin.open("entrada.txt", ifstream::in); // abre el archivo para lectura
10   fin >> a; // lee el 15
11   fin >> b; // lee el 20
12   fin.close(); // cierra el archivo
13   return 0;
14 }
```

Ejemplo: leer archivos

- Leer de un archivo `entrada.txt` un valor entero y almacenarlo en una variable llamada `a` y luego leer un valor real y almacenarlo en un variable `f`.
- Ambos valores están separados por una coma y hay una única línea en el archivo.
  - Ejemplo:
  - `-234,1.7`

Ejemplo: leer archivos

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int main() {
6      int a = 0;
7      float f = 0.0;
8      ifstream fin;
9      fin.open("entrada.txt", ifstream::in); // abre archivo lectura
10     char c;
11     fin >> a; // lee el entero a (-234)
12     fin >> c; // lee el simbolo coma (,)
13     fin >> f; // lee el real f (1.7)
14     fin.close(); // cierra archivo
15     return 0;
16 }
```

Función end-of-file (eof)

- Además de `open` y `close` tenemos la función `eof()`.
- La función `eof()` retorna `true` si ya no hay más contenido del archivo para leer.
- Usaremos `eof()` sólo cuando abrimos un archivo para lectura.
- Ejemplo:
  - Leer de un archivo una lista de enteros y calcular la suma de sus elementos.
  - Los enteros se encuentran separados por un espacio vación
  - No sabemos de antemano cuantos enteros hay almacenados en el archivo.

Usando eof()

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int main() {
6    int suma = 0;
7    ifstream fin;
8    fin.open("archivo.txt", ios::in); // abrir para lectura
9
10   while( !fin.eof() ) {
11     int a = 0;
12     fin >> a;
13     suma += a;
14   }
15
16   fin.close(); // cerrar archivo
17   cout << "La suma de la lista es: " << suma << endl;
18   return 0;
19 }
```

Manejo de errores

- Hasta ahora tenemos las funciones `open`, `close` y `eof` para operar con archivos.
- ¿Qué pasa cuando queremos abrir un archivo para lectura que no existe?
- ¿Qué pasa cuando no tenemos permisos para leer un archivo?
- ¿Qué pasa cuando no tenemos permisos para sobreescribir un archivo?
- Para todos esos casos, se puede consultar a la función `fail()`
- La función `fail()` retorna `true` si hubo una falla al intentar ejecutar una operación (por ejemplo: `open`, `close`)
- Ejemplo:
  - Escribir un programa que intente leer un archivo e imprima `Abierto` si lo pudo abrir y `Error` si no lo pudo hacer

Demo #9: Manejo de errores

```
1  #include <iostream>
2  #include <fstream>
3  using namespace std;
4
5  int main() {
6      int a =0;
7      int b =0;
8      ifstream fin;
9      fin.open("archivoNoExiste.txt", ifstream::in);
10     if (fin.fail()) { // true si hubo error al abrir
11         cout << "Error" << endl;
12     } else {
13         cout << "Abierto" << endl;
14     }
15     fin.close();
16     return 0;
17 }
```

Resumen: E/S con archivos en C++

- ifstream: stream de lectura de archivos
- ofstream: stream para escritura de archivos
- open(): abre un archivo para escritura o lectura dependiendo del tipo de stream
- close(): cierra un archivo
- << (escribe un valor) y >> (lee un valor)
- eof(): retorna `true` si la lectura del archivo llegó al final
- fail(): retorna `true` si la última operación falló

Bibliografía

- B. Stroustrup. The C++ Programming Language.
  - 31.4.1: El STL container *vector*
  - 38.2.1: File Streams