

Laboratorio 7: Templates

Jean Carlos Chavarría Hughes B11814

19 de octubre de 2014

Resumen

Laboratorio 8 de el curso IE 0217.

1. Introducción

Este documento corresponde al reporte del laboratorio 8 del curso IE0217, en el cual se trabajó tanto con el concepto de contenedores, como los algoritmos de C++.

Se adjuntan los ficheros fuente, con su respectivo makefile para simplemente ingresar a la carpeta correspondiente y ejecutar el comando *make*.

2. Contenedores

La biblioteca de contenedores es una colección genérica de plantillas de clase que permiten a los programadores implementar estructuras de datos comunes fácilmente, como colas, listas y pilas. Hay tres clases de contenedores - contenedores de secuencias, contenedores asociativos y contenedores asociativos desordenados - cada uno de los cuales está diseñado para dar soporte a un conjunto diferente de operaciones.

El contenedor gestiona el espacio de almacenamiento que es asignado a sus elementos y proporciona funciones miembro para acceder a ellos, ya sea directamente o mediante iteradores (objetos con propiedades similares a los punteros).

La mayoría de los contenedores tienen varias funciones miembro en común, y comparten ciertas funcionalidades. Qué contenedor es mejor para una aplicación particular no sólo depende de la funcionalidad ofrecida, sino también de su eficiencia para los diferentes tipos de trabajo.

3. Contendor Vector

Se realizó lo solicitado en la guía de laboratorio, mediante la creación de un archivo .txt en el cual se tuviese el abecedario ordenado en una sola columna.

Al ejecutar el programa, lo primero que sucede es que se lee el archivo `text.txt`, luego se almacena en un *vector de chars*. En la segunda parte, se realiza la impresión en pantalla del contenido de archivo pero en orden inverso, y finalmente se creó una función *join()*, la cual une los dos vectores, el normal y el inverso, y lo muestra en pantalla.

```
vector<char> join(vector<char> V1, vector<char> V2){
std::vector<char> V3;
V3.reserve( V1.size() + V2.size() ); // preallocate memory
V3.insert( V3.end(), V1.begin(), V1.end() );
V3.insert( V3.end(), V2.begin(), V2.end() );
return V3;
}
```

4. Contenedor list

Igual que el ejercicio anterior, se ejecutó de manera satisfactoria, ya que se realizó un programa que lee un archivo de texto y almacena cada palabra en una lista de *strings*. Luego, con la función creada *findLastOf()*, se encuentra la ultima posición de un valor dado por el usuario, y retorna la posición en la cual este se encuentra. Después, una nueva función que recibe un tipo *string*, y elimina las palabras duplicadas, con el fin de calcular el espacio que utilizaría sin tener palabras duplicadas. Finalmente, se creó una nueva función que toma lista tipo *string* y la ordena alfabéticamente, tal como se puede observar en la figura 1.

5. Contenedor stack

En esta parte de laboratorio, se creó una función llamada *multiBase()*, la cual se encarga simplemente de recibir un numero en base decimal, y pasarlo a otra base especificada por el usuario, entre 2 y 9. Un ejemplo se puede observar en la figura 2.

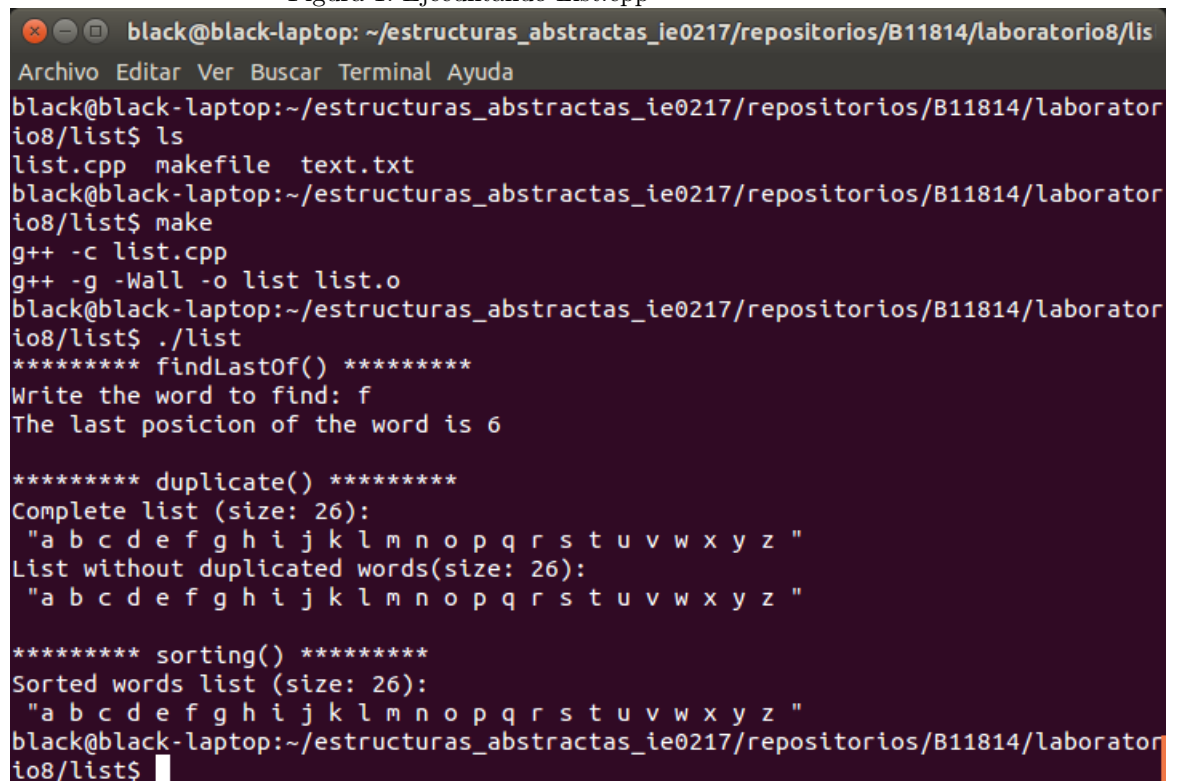
6. Contenedor Queue

Al igual que las anteriores, esta parte se encarga de implementar un algoritmo que utiliza *radix sort*, específicamente para ordenar una secuencia de numeros ingresados por el usuario. Un ejemplo se puede observar en la Figura 3.

7. Contenedor map

Finalmente, se realizó la implementación de un contenedor tipo *map*, en cual se encarga de almacenar los número de carné, nombre del título e integrantes de proyecto del curso. Uno de los aspectos a destacar es que esta función o programa tiene que ser capaz de conocer si un carne digitado por el usuario es

Figura 1: Ejecutando List.cpp

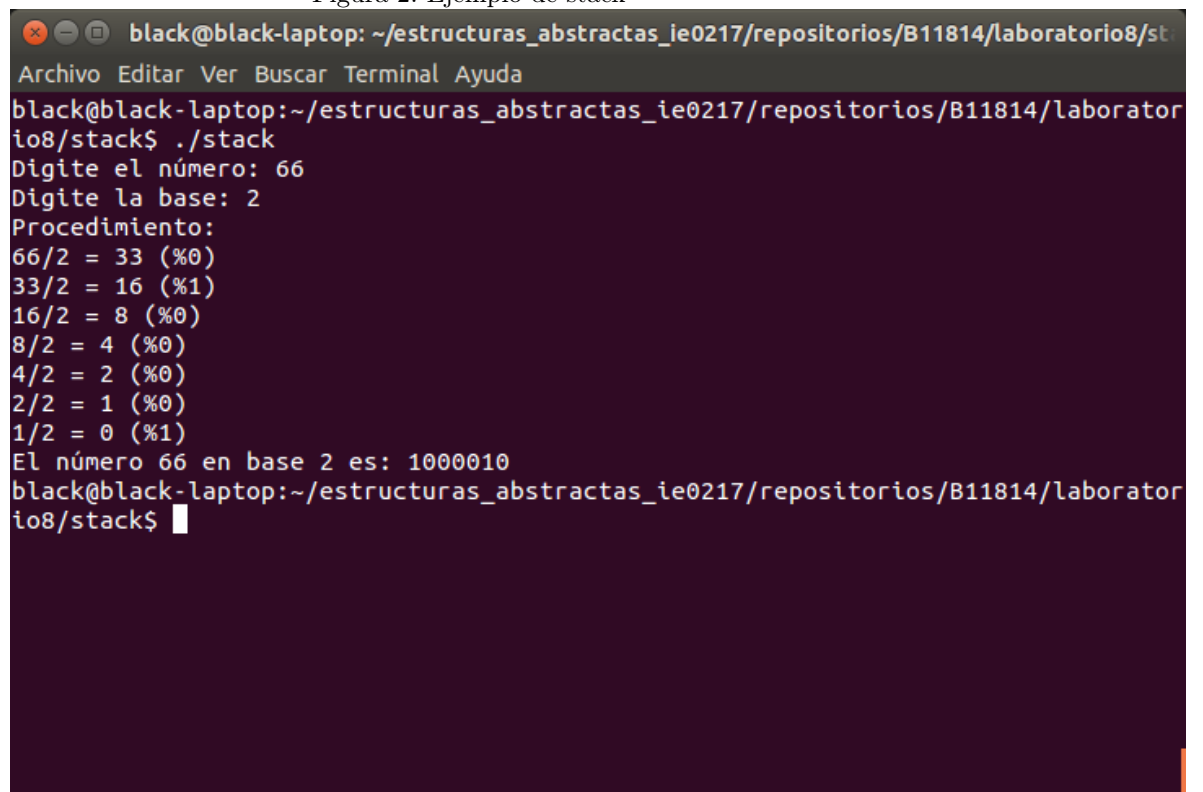


```
black@black-laptop: ~/estructuras_abstractas_ie0217/repositorios/B11814/laboratorio8/lis
Archivo Editar Ver Buscar Terminal Ayuda
black@black-laptop:~/estructuras_abstractas_ie0217/repositorios/B11814/laboratorio8/list$ ls
list.cpp  makefile  text.txt
black@black-laptop:~/estructuras_abstractas_ie0217/repositorios/B11814/laboratorio8/list$ make
g++ -c list.cpp
g++ -g -Wall -o list list.o
black@black-laptop:~/estructuras_abstractas_ie0217/repositorios/B11814/laboratorio8/list$ ./list
***** findLastOf() *****
Write the word to find: f
The last position of the word is 6

***** duplicate() *****
Complete list (size: 26):
"a b c d e f g h i j k l m n o p q r s t u v w x y z "
List without duplicated words(size: 26):
"a b c d e f g h i j k l m n o p q r s t u v w x y z "

***** sorting() *****
Sorted words list (size: 26):
"a b c d e f g h i j k l m n o p q r s t u v w x y z "
black@black-laptop:~/estructuras_abstractas_ie0217/repositorios/B11814/laboratorio8/list$
```

Figura 2: Ejemplo de stack

A terminal window with a dark background and light text. The title bar shows a window icon, a maximize icon, and a close icon, followed by the text 'black@black-laptop: ~/estructuras_abstractas_ie0217/repositorios/B11814/laboratorio8/st'. Below the title bar is a menu bar with 'Archivo', 'Editar', 'Ver', 'Buscar', 'Terminal', and 'Ayuda'. The terminal content shows the execution of a program named 'stack'. The user enters the number 66 and the base 2. The program outputs the division steps for converting 66 to base 2, resulting in the binary number 1000010.

```
black@black-laptop: ~/estructuras_abstractas_ie0217/repositorios/B11814/laboratorio8/st
Archivo Editar Ver Buscar Terminal Ayuda
black@black-laptop:~/estructuras_abstractas_ie0217/repositorios/B11814/laborator
io8/stack$ ./stack
Digite el número: 66
Digite la base: 2
Procedimiento:
66/2 = 33 (%0)
33/2 = 16 (%1)
16/2 = 8 (%0)
8/2 = 4 (%0)
4/2 = 2 (%0)
2/2 = 1 (%0)
1/2 = 0 (%1)
El número 66 en base 2 es: 1000010
black@black-laptop:~/estructuras_abstractas_ie0217/repositorios/B11814/laborator
io8/stack$
```

Figura 3: Ejemplo de queue

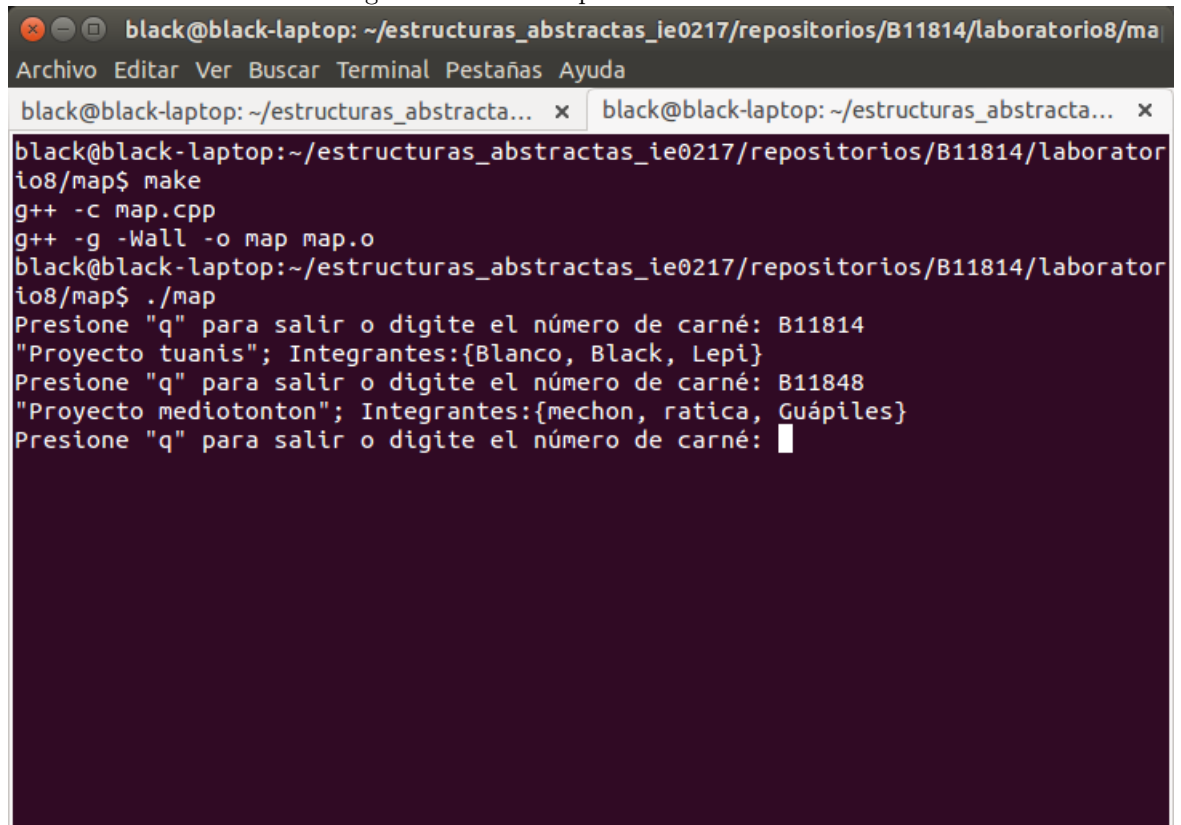
```
black@black-laptop: ~/estructuras_abstractas_ie0217/repositorios/B11814/laboratorio8/queue
Archivo Editar Ver Buscar Terminal Ayuda
black@black-laptop:~/estructuras_abstractas_ie0217/repositorios/B11814/laboratorio8/queue$ ./queue
digite los numeros que quiere ordenar y escriba 'fin' al lado del ultimo elemento para finalizar
33
Se agrego un 33
45
Se agrego un 45
789
Se agrego un 789
2
Se agrego un 2
01
Se agrego un 1
3
Se agrego un 3
fin
La cola tiene 6 elementos
33 45 789 2 1 3

1 2 3 33 45 789

black@black-laptop:~/estructuras_abstractas_ie0217/repositorios/B11814/laboratorio8/queue$
```

válido o inválido, y no crashear en el intento. Por ejemplo, se puede observar la Figura 4, donde evidentemente se utilizaron numeros de carné ficticios debido al desconocimiento de los mismos.

Figura 4: Uso de map



```
black@black-laptop: ~/estructuras_abstractas_ie0217/repositorios/B11814/laboratorio8/ma
Archivo Editar Ver Buscar Terminal Pestañas Ayuda
black@black-laptop: ~/estructuras_abstracta... x black@black-laptop: ~/estructuras_abstracta... x
black@black-laptop:~/estructuras_abstractas_ie0217/repositorios/B11814/laborator
io8/map$ make
g++ -c map.cpp
g++ -g -Wall -o map map.o
black@black-laptop:~/estructuras_abstractas_ie0217/repositorios/B11814/laborator
io8/map$ ./map
Presione "q" para salir o digite el número de carné: B11814
"Proyecto tuanis"; Integrantes:{Blanco, Black, Lepi}
Presione "q" para salir o digite el número de carné: B11848
"Proyecto mediotonton"; Integrantes:{mechon, ratica, Guápiles}
Presione "q" para salir o digite el número de carné: █
```

8. Algoritmos

En C++ los algoritmos se refieren a una librería que se utiliza para una gran variedad de propósitos, por ejemplo, búsquedas, ordenamientos, cuentas, manipulación, etc. Siempre se utilizan los índices (*first, last*) para definir los rangos. En este caso se utilizó el algoritmo *copy*, *sort*, *count*, *accumulate* y *reverse*.

8.1. Copy

Copia elementos en el rango definido a otro rango comenzando con *d first*.

```

template<class InputIt, class OutputIt>
OutputIt copy(InputIt first, InputIt last,
              OutputIt d_first)
{
    while (first != last) {
        *d_first++ = *first++;
    }
    return d_first;
}

```

8.2. Sort

Ordena los elementos en el rango definido en orden ascentente. Si se desea un orden descendente se utiliza el parámetro *greater*. Header

```

template< class RandomIt >
void sort( RandomIt first, RandomIt last );

```

8.3. Count

Retorna el numero de elementos en el rango definido

```

template< class InputIt, class T >

typename iterator_traits<InputIt>::difference_type
count( InputIt first, InputIt last, const T &value );

```

8.4. Accumulate

Calcula la suma de valores dados en el rango definido.

```

template< class InputIt, class T >
T accumulate( InputIt first, InputIt last, T init );

```

8.5. Reverse

Reordena los elementos del rango definido en orden inverso.

```

template< class BidirIt >
void reverse( BidirIt first, BidirIt last );

```