Taller 2: Estructuras lineales

Kevin Estiven Velasco Pinto
Ingeniería de Sistemas
Pontificia Universidad Javeriana
Bogotá, Colombia
kevinevelasco@javeriana.edu.co

Juan Andrés Ramírez Pachón
Ingeniería de Sistemas
Pontificia Universidad Javeriana
Bogotá, Colombia
ramirezpj@javeriana.edu.co

Juan Diego Córdoba Fernández
Ingeniería de Sistemas
Pontificia Universidad Javeriana
Bogotá, Colombia
cojuandiego@javeriana.edu.co

***Abstract*— The following report will contain an explanation of each part of the code that is executed to obtain the anagrams of a given sequence of strings or characters. In addition, it will include the fragments that we completed in order to test it satisfactorily.**
Keywords—anagram, string, list, stack, queue (key words**)**

I. Introducción

El siguiente informe contendrá la explicación de cada parte del código que se ejecuta para obtener los anagramas de una secuencia de strings o de caracteres dada, además, los fragmentos que completamos para así poder probarlo de manera satisfactoria.

II. Desarrollo

1. *Descripción de la función principal (main)*
   Lee la entrada, computa como *strings* o caracteres, calcula el número de anagramas; los imprime.

2. *Descripción de la salida de la función "ReadAsCharacterList".*
   Dado un flujo de entrada, lo almacena en un vector de caracteres y lo retorna.

3. *Descripción de la salida de la función "ReadAsStringList".*
   Dado un flujo de entrada, almacena en vector de *strings* y lo retorna.

4. *Estudiar el archivo "NextAnagram.h".*

5. *TODO #1: a partir de la secuencia "lst" de entrada a la función, llenar la pila "s".*

```
for(lIt = lst.begin(); lIt != lst.end(); lIt++){
  s.push(*lIt);
}
```

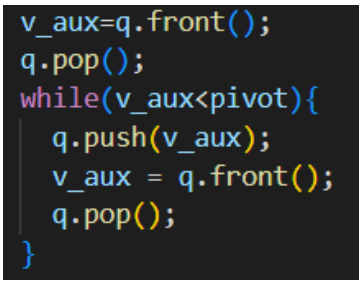
Fig. 1. TODO #1. Fuente: Elaboración propia.

6. *TODO #2: extraer el tope de la pila "s" y almacenarlo en la variable "v_aux".*

```
v_aux = s.top();
s.pop();
```

Fig. 2. TODO #2. Fuente: Elaboración propia.

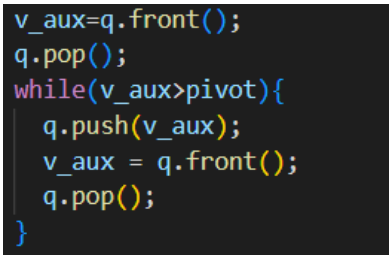
7. *TODO #3: iterativamente, extraer el frente (cabeza) de la cola "q" y adicionar este valor a la misma cola. Este ciclo se detiene cuando el elemento que se acaba de extraer es menor que lo contenido en la variable "pivot".*



```
v_aux=q.front();
q.pop();
while(v_aux<pivot){
  q.push(v_aux);
  v_aux = q.front();
  q.pop();
}
```

Fig. 3. TODO #3. Fuente: Elaboración propia.

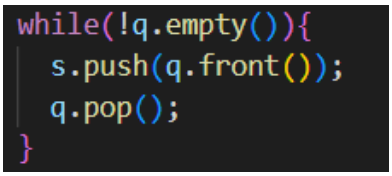
8. *TODO #4: iterativamente, extraer el frente (cabeza) de la cola "q" y adicionar este valor a la misma cola. Este ciclo se detiene cuando el elemento que se acaba de extraer es mayor que lo contenido en la variable "pivot".*



```
v_aux=q.front();
q.pop();
while(v_aux>pivot){
  q.push(v_aux);
  v_aux = q.front();
  q.pop();
}
```

Fig. 4. TODO #4. Fuente: Elaboración propia.

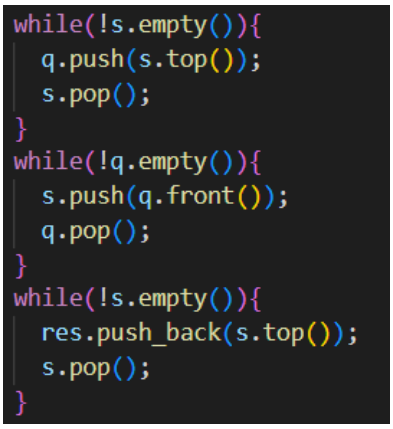
9. *TODO #5: iterativamente, vaciar la cola "q" en la pila "s".*



```
while(!q.empty()){
  s.push(q.front());
  q.pop();
}
```

Fig. 5. TODO #5. Fuente: Elaboración propia.

10. *TODO #6: iterativamente, vaciar la pila "s" e insertar por la cabeza cada elemento en la secuencia resultado "res".*



```
while(!s.empty()){
  q.push(s.top());
  s.pop();
}
while(!q.empty()){
  s.push(q.front());
  q.pop();
}
while(!s.empty()){
  res.push_back(s.top());
  s.pop();
}
```

Fig. 6. TODO #6. Fuente: Elaboración propia.

11. *TODO #7: Completar la función que calcula la cantidad de anagramas que resultan de una secuencia de entrada. Si el tamaño de la secuencia de entrada es n, la cantidad de anagramas posibles es n!*

```cpp
unsigned long ComputeNumberOfAnagrams( const TList& lst )
{
  /** TODO #7 **/
  unsigned long nAnagrams = 1;
  for(int i = 1; i <= lst.size(); i++){
    nAnagrams *= i;
  }
  return(nAnagrams);
}
```

Fig. 7. TODO #7. Fuente: Elaboración propia.

12. *Para probar, se comprueba el archivo de entradas de muestra (input_00.in) con su correspondiente archivo de salida (output_00.in).*

output_00.out	1 amor
1 amor	amor
2 amro	amro
3 aomr	aomr
4 aorm	aorm
5 armo	armo
6 arom	arom
7 maor	maor
8 maro	maro
9 moar	moar
10 mora	mora
11 mrao	mrao
12 mroa	mroa
13 oamr	oamr
14 oarm	oarm
15 omar	omar
16 omra	omra
17 oram	oram
18 orma	orma
19 ramo	ramo
20 raom	raom
21 rmao	rmao
22 rmoa	rmoa
23 roam	roam
24 roma	roma

Fig. 8. Pruebas del código. Fuente: Elaboración propia.

13. *Investigar si existe una función que calcule anagramas en la STL. Ponga en su reporte el encabezado de la función y una descripción concisa (10 palabras o menos) de su forma de uso.*

next_permutation(): Genera la siguiente permutación lexicográfica ordenada de una secuencia dada [1, p. 941].

III. Conclusiones

1. *Las pilas y las colas se complementan muy bien en ambientes de desarrollo que son propicios a tener grandes cantidades de infomación y combinaciones posibles donde nos importe el orden.*

2. *Las pilas y las colas no pueden recorrerse buscando un elemento, sino que es necesario extraer los elementos para así poder asignarlos a otro tipo de estructura de datos.*

3. *Existen funciones muy útiles en la STL (Standard Template Library) que nos permite realizar funciones de una forma más rápida y sencilla, como por ejemplo next_permutation().*

Referencias

[1] B. Stroustrup, *The C++ programming language*, Fourth edition. Upper Saddle River, NJ: Addison-Wesley, 2013.