

## ESTRUCTURAS DE DATOS

### EJERCICIOS DE EVALUACIÓN PILAS Y COLAS

**Nombre del Estudiante:** Giovanni Alexander Sánchez García

1. Un ejemplo de un problema de correspondencia entre paréntesis proviene del lenguaje de marcas de hipertexto (HTML). En HTML, las etiquetas existen tanto en la forma de apertura como en la forma de cierre y deben estar balanceadas para describir correctamente un documento web. El siguiente documento sencillo en HTML: está destinado únicamente a mostrar la estructura de coincidencia y anidamiento de las etiquetas en el lenguaje HTML. Escriba un programa que pueda comprobar que las etiquetas de apertura y cierre en un documento HTML son las adecuadas.

Recomendaciones:

- Use la siguiente expresión regular para obtener las etiquetas: "`(<\w+>|</\w+>)`"
- Cree una pila de etiquetas. Para cada etiqueta identificada en el literal anterior:
  - Si es una etiqueta de apertura, agréguela a la pila de etiquetas
  - Si es una etiqueta de cierre, extraiga la etiqueta que está en el tope de la pila. Verifique que la etiqueta que acaba de extraer corresponda a la etiqueta de cierre (ejemplo: `</h1>` debe cerrar a `<h1>`). Si las etiquetas son diferentes, el documento HTML no es válido.
- El documento HTML es válido si al finalizar el bucle la pila de etiquetas queda vacía.

```
< html>
  < head>
    <title >
      Ejemplo
    </ title >
  </ head>

  < body>
    <h1 > Hola mundo </ h1>
  </ body>
</html >
```

```

#include <iostream>
#include <regex>
#include <iterator>
#include <string>
#include "pila.hpp"
#include <vector>

using std::string;
using std::cin;
using std::cout;
using std::endl;
using std::vector;

bool revisandoEtiquetas(string caadena){

    Pila<string> pilaEtiquetas;

    const std::string s = caadena;

    std::regex words_regex("<\\w+>|</\\w+>");

    auto words_begin = std::sregex_iterator(s.begin(), s.end(), words_regex); //funcion para buscar expresiones regulares en un bucle.
    auto words_end = std::sregex_iterator();

    for(std::sregex_iterator i = words_begin; i != words_end; ++i){
        std::smatch match = *i; //puntero i
        std::string match_str = match.str();
        cout << match_str << endl;
    }
}

```

```
*** COMPROBANDO BALANCEO DE ETIQUETAS HTML ***
```

```
<html>  
<head>  
<title>  
</head>  
<body>  
<h1>  
<p>  
</p>  
</h1>  
</body>  
</html>
```

etiquetas no balanceadas

```
<html>  
<head>  
<title>  
</title>  
</head>  
<body>  
<h1>  
<p>  
</p>  
</h1>  
</body>  
</html>
```

etiquetas balanceadas

```
-----  
Process exited after 0.2219 seconds with return value 0  
Presione una tecla para continuar . . . █
```

```

#include "lista_generica.hpp"
#include <string>

template <typename TIPODATO>
class Pila : private Lista<TIPODATO>{
public:
    //Constructor
    Pila();
    //Agregar un elemento en la parte superior
    push(TIPODATO item);
    //Remover un elemento de la parte superior
    TIPODATO pop();
    //Extraer valor
    TIPODATO extraer();
    bool estaPilaVacía();
    string pilaComoCadena();
};

template<typename TIPODATO>
Pila<TIPODATO>::Pila(){
    Lista<TIPODATO> Lista;
}

template<typename TIPODATO>
Pila<TIPODATO>::push(TIPODATO item){
    this->adjuntar(item);
}

template<typename TIPODATO>
Pila<TIPODATO>::Pila(){
    Lista<TIPODATO> Lista;
}

template<typename TIPODATO>
Pila<TIPODATO>::push(TIPODATO item){
    this->adjuntar(item);
}

template<typename TIPODATO>
TIPODATO Pila<TIPODATO>::pop(){
    return this->remover(this->tamano()-1);
}

template<typename TIPODATO>
bool Pila<TIPODATO>::estaPilaVacía(){
    return this->estaVacía();
}

template<typename TIPODATO>
string Pila<TIPODATO>::pilaComoCadena(){
    return this->comoCadena();
}

template<typename TIPODATO>
TIPODATO Pila<TIPODATO>::extraer(){
    return this->obtener(this->tamano()-1);
}

```

```

#include "lista_generica.hpp"
#include <string>

template<typename TIPODATO>
class Cola : private Lista<TIPODATO>{
public:
    //Constructor
    Cola();
    //Agregar un elemento al final de la cola
    enqueue(TIPODATO item);
    //Remover un elemento al inicio de la cola
    TIPODATO dequeue();
    bool estaColaVacia();
    string colaComoCadena();
};

template<typename TIPODATO>
Cola<TIPODATO>::Cola(){
    Lista<TIPODATO> Lista;
}

template<typename TIPODATO>
Cola<TIPODATO>::enqueue(TIPODATO item){
    this->adjuntar(item);
}

template<typename TIPODATO>
TIPODATO Cola<TIPODATO>::dequeue(){

};

template<typename TIPODATO>
Cola<TIPODATO>::Cola(){
    Lista<TIPODATO> Lista;
}

template<typename TIPODATO>
Cola<TIPODATO>::enqueue(TIPODATO item){
    this->adjuntar(item);
}

template<typename TIPODATO>
TIPODATO Cola<TIPODATO>::dequeue(){
    return this->remover(0);
}

template<typename TIPODATO>
bool Cola<TIPODATO>::estaColaVacia(){
    return this->estaColaVacia();
}

template<typename TIPODATO>
string Cola<TIPODATO>::colaComoCadena(){
    return this->colaComoCadena();
}

```

```

#ifndef lista_generica_hpp
#define lista_generica_hpp

#include <iostream>
#include <string>
#include <sstream>

using std::string;
using std::ostringstream;
using std::cout;
using std::endl;

template <typename TIPODATO>
class Lista{

private:
    int cuenta;
    int capacidad;
    TIPODATO *items;
    void agrandar();
public:
    Lista(int capacidad);
    Lista();
    ~Lista();
    Lista(const Lista &otra);
    bool estaVacia();
    int tamano();
    void insertar(int indice, TIPODATO item);
    void adjuntar(TIPODATO item);

    bool contiene (TIPODATO item);
    TIPODATO remover(int indice);
    string comoCadena();
};

//constructor
template <typename TIPODATO>
Lista<TIPODATO>::Lista(int capacidad){

    this->cuenta = 0;
    this->capacidad = capacidad;
    this->items = new TIPODATO[capacidad];

}

template <typename TIPODATO>
Lista<TIPODATO>::Lista() : Lista(4){}

template <typename TIPODATO>
Lista<TIPODATO>::~~Lista(){

    delete[] items;
}

```



```

template <typename TIPODATO>
Lista<TIPODATO>::Lista(const Lista<TIPODATO> &otra){
|
    this->cuenta = otra.cuenta;
    this->capacidad = otra.capacidad;
    this->items = new TIPODATO[capacidad];
    for (int i = 0; i < this->cuenta; i++)
        this->items[i] = otra.items[i];
}

//esta la lista vacia ?
template <typename TIPODATO>
bool Lista<TIPODATO>::estaVacia(){
    // cout << "comprobando si la lista esta vacia..." << endl;
    return this->cuenta == 0;
}

//tamaño de la lista
template <typename TIPODATO>
int Lista<TIPODATO>::tamano(){
    // cout << "obteniendo el tamaño de la lista..." << endl;
    return this->cuenta;
}

//Aumentar tamaño de lista
template <typename TIPODATO>

```

```

void Lista<TIPODATO>::insertar (int indice, TIPODATO item){
    //validar el indice
    if (indice < 0 || indice > this->cuenta) throw "indice fuera de rango";

    if (this->cuenta >= this->capacidad) this->agrandar();

    for (int i = cuenta - 1; i >= indice; i--){
        cout << "desplazando elemento " << items [i] << "del indice";
        cout << i << "al indice" << (i+1) << endl;
        this->items[i+1] = this->items[i];
    }

    //insertar nuevo elemento
    // cout << "insertando elemento " << item << "en el indice" << indice << endl;
    this->items[indice] = item;

    //incrementar la cuenta de elementos
    // cout << "incrementando la cuenta de elementos" << endl;
    this->cuenta++;
}

//agregar un elemento al final
template <typename TIPODATO>
void Lista<TIPODATO>::adjuntar(TIPODATO item){
    this->insertar(this->cuenta, item);
}

```



```

//valor del elemento en la posicion indice
template <typename TIPODATO>
TIPODATO Lista<TIPODATO>::obtener(int indice){
    //validar el indice
    if (indice < 0 || indice >= this->cuanta) throw "indice fuera de rango";
    if (this->estaVacia()) throw "No se puede recuperar elementos de una lista vacia";

    // cout << "Recuperando elemento en el indice" << indice << endl;
    return this->items[indice];
}

//esta el item en la lista?
template <typename TIPODATO>
bool Lista <TIPODATO>::contiene(TIPODATO item){
    for (int i =0; i <this->cuanta; i++){
        // cout << "recorriendo elemento con indice " << i << endl;
        if (this->items[i] == item)
            return true;
    }

    return false;
}

//remove elemento en el indice indice'
template <typename TIPODATO>
TIPODATO Lista<TIPODATO>::remove(int indice){
    //validar el indice

    //lee el valor en el indice y desplaza elementos a la izquierda
    TIPODATO valor = this->items[indice];
    for (int i = indice; i <cuanta -1;i++){
        // cout << "desplazando elemento " << items[i] <<"del indice ";
        // cout << (i+1) << "al indice " <<i << endl;
        this->items[i] = this->items[i+1];
    }

    //reducir la cuenta de elementos
    // cout <<"reduciendo la cuenta de elementos "<< endl;
    this->cuanta--;
    return valor;
}

template <typename TIPODATO>
string Lista<TIPODATO>::comoCadena(){
    ostringstream s;
    s<< "[";
    for (int i =0; i<(this->cuanta);i++){
        s << this->items[i];
        if (i < this->cuanta-1)
            s << ", ";
    }
    s << "]";
    return s.str();
}

#endif //lista_generica_hpp

```

```

#include <iostream>
#include <string>
#include <cctype>
#include <algorithm> //Libreria que ayuda a eliminar espacios y signos de puntuacion
#include "pila.hpp"
#include "cola.hpp"

using namespace std;

bool evaluarPalindromo(string palabraEvaluar);
bool evaluarTexto(string mensaje);

int main(){

    string texto;
    string textoIgnorando;

    cout << " *** EVALUADOR DE PALABRAS PALINDROMO *** " << endl << endl;
    cout << "Digita la palabra a evaluar: ";
    getline(cin, texto);

    textoIgnorando = texto;
    textoIgnorando.erase(std::remove(textoIgnorando.begin(), textoIgnorando.end(), ' '), textoIgnorando.end());
    textoIgnorando.erase(std::remove_if(textoIgnorando.begin(), textoIgnorando.end(), [](unsigned char c){ return !std::isalpha(c); }), textoIgnorando.end());
    transform(textoIgnorando.begin(), textoIgnorando.end(), textoIgnorando.begin(), [](unsigned char c){ return std::tolower(c); });

    for (int i = 0; i < textoIgnorando.length(); i++) {
        // Y cambiar cada letra por su representación
        // mayúscula
        textoIgnorando[i] = toupper(textoIgnorando[i]);
    }

    cout << endl;
    if(evaluarPalindromo(textoIgnorando)){
        cout << "La palabra: '" << texto << "' es Palindromo!!" << endl;
    }else{
        cout << "La palabra: '" << texto << "' NO es Palindromo!!" << endl;
    }

    cout << endl;
    cout << endl;

    return 0;
}

//funcion para evaluar palabra
bool evaluarPalindromo(string palabraEvaluar){

    char letra;
    string palabraPila;
    string palabraLetra;

```

```

//Declaracion de pila y cola a partir de lista generica
Pila<char> pilaLetra;
Cola<char> colaLetra;

//Agregando caracteres de la palabra a la pila y cola
for(int i = 0; i < palabraEvaluar.length(); i++){
    letra = palabraEvaluar[i];
    pilaLetra.push(letra);
    colaLetra.enqueue(letra);
}

//Extrayendo caracteres de la pila y cola
for(int i = 0; i < palabraEvaluar.length(); i++){
    palabraPila += pilaLetra.pop();
    palabraLetra += colaLetra.dequeue();
}

if(palabraPila == palabraLetra){
    cout << "Palabra al derecho: " << palabraPila << endl;
    cout << "Palabra al revés: " << palabraLetra << endl;
    return true;
}else{
    cout << "Palabra al derecho: " << palabraPila << endl;
    cout << "Palabra al revés: " << palabraLetra << endl;
    return false;
}

```



Edificio de Atención al Estudiante, Nivel 3, 55 Av. Sur, Condominio Centro Roosevelt, entre Av. Olímpica y Alameda Roosevelt, San Salvador, El Salvador, C.A.



jaleman@ufg.edu.sv Página 1 de 2 (503) 2209-2930

2. Use una pila y una cola para determinar si una palabra es un palíndromo (una palabra que se lee igual al derecho y al revés, por ejemplo, “anona”). Para cada carácter de la palabra, agregue ambos una cola (donde quedarán en el mismo orden) y a una pila (donde quedarán en orden inverso). Luego, remueva las letras, una por una de ambas estructuras (operaciones extraer y avanzar) y compare las letras. Si todos los pares de letras removidas son iguales, la palabra es un palíndromo. Si alguna de los pares de letras es diferente, la palabra no es un palíndromo.

Use las clases pila.h y cola.h creados por usted a partir de la lista genérica, para resolver estos problemas.

```
*** EVALUADOR DE PALABRAS PALINDROMO ***

Digita la palabra a evaluar: oso

Palabra al derecho: OSO
Palabra al revés: OSO
La palabra: 'oso' es Palindromo!!

-----
Process exited after 1.402 seconds with return value 0
Presione una tecla para continuar . . .
```

```
*** EVALUADOR DE PALABRAS PALINDROMO ***

Digita la palabra a evaluar: lamina

Palabra al derecho: ANIMAL
Palabra al revés: LAMINA
La palabra: 'lamina' NO es Palindromo!!

-----
Process exited after 2.859 seconds with return value 0
Presione una tecla para continuar . . .
```

## CRITERIOS DE EVALUACIÓN

### Problema 1

| Criterio   | Ponderación | Calificación |
|--|-------------|--------------|
| Creación de lista de etiquetas con expresiones regulares | 20%         |              |
| Uso de pila para verificar etiquetas balanceadas         | 20%         |              |
| Orden y legibilidad del código                           | 20%         |              |
| Solución devuelve el resultado esperado (True   False)   | 40%         |              |
| <b>Total</b>   | <b>100%</b> |              |

## Problema 2

| Criterio   | Ponderación | Calificación |
|--|-------------|--------------|
| Solución usa una pila  | 20%         |              |
| Solución usa una cola  | 20%         |              |
| Solución ignora espacios en blanco, signos de puntuación y otros caracteres no alfanuméricos | 10%         |              |
| Orden y legibilidad del código   | 10%         |              |
| Solución devuelve el resultado esperado (True   False)                                       | 40%         |              |
| <b>Total</b>   | <b>100%</b> |              |



Edificio de Atención al Estudiante, Nivel 3, 55 Av. Sur, Condominio Centro Roosevelt, entre Av. Olímpica y Alameda Roosevelt, San Salvador, El Salvador, C.A.

