UNIVERSIDAD DE GUADALAJARA CUCEI

Tarea: Analizador sintáctico (implementando usando objetos)

Nombre: Ivan Nudelstejer Gomez

Código: 218130122

Materia: Seminario de solucion de problemas de traductores de lenguaje II

Sección: D02

Profesor: Michel emanuel lopez franco

Objetivo:
En esta práctica utilizamos una pila de objetos en lugar de enteros, de esta forma al
En el momento que imprimas la pila aparecerán los símbolos de forma similar a cuando realizas el análisis manualmente.

Requerimientos:
Para esta práctica necesitarás crear una clase ElementoPila y modificaras la clase pila para que acepte objetos de este tipo en lugar de enteros.
Necesitarás crear 3 clases más, las cuales heredan de ElementoPila, las clases son:

- Terminal
- No terminal
- Estado

```
#include <iostream>
#include <list>
#include <string>

using namespace std;

// Clase base ElementoPila
class ElementoPila {
public:
    virtual ~ElementoPila() {}  // Destructor virtual
    virtual void muestra() = 0;  // Función virtual pura
};

// Clase Terminal hereda de ElementoPila
class Terminal : public ElementoPila {
protected:
    string nombre;
public:
    Terminal(string nombre) : nombre(nombre) {}

    void muestra() override {
        cout << "Terminal: " << nombre << endl;
    }
};

// Clase NoTerminal hereda de ElementoPila
class NoTerminal : public ElementoPila {
```

```
protected:
    string nombre;
public:
    NoTerminal(string nombre) : nombre(nombre) {}

    void muestra() override {
        cout << "No Terminal: " << nombre << endl;
    }
};

// Clase Estado hereda de ElementoPila
class Estado : public ElementoPila {
protected:
    string nombre;
public:
    Estado(string nombre) : nombre(nombre) {}

    void muestra() override {
        cout << "Estado: " << nombre << endl;
    }
};

// Clase Alumno (base para Bachillerato y Licenciatura)
class Alumno : public ElementoPila {
protected:
    string codigo;
public:
    Alumno(string codigo) : codigo(codigo) {}

    virtual ~Alumno() {}  // Destructor virtual
    virtual void muestra() override {}
};

// Clase Bachillerato hereda de Alumno
class Bachillerato : public Alumno {
protected:
    string preparatoria;
public:
    Bachillerato(string codigo, string preparatoria) : Alumno(codigo),
preparatoria(preparatoria) {}
```

```
    void muestra() override {
        cout << "Alumno Bachillerato" << endl;
        cout << "Codigo: " << codigo << endl;
        cout << "Preparatoria: " << preparatoria << endl << endl;
    }
};

// Clase Licenciatura hereda de Alumno
class Licenciatura : public Alumno {
protected:
    string carrera;
    int creditos;
public:
    Licenciatura(string codigo, string carrera, int creditos) :
Alumno(codigo), carrera(carrera), creditos(creditos) {}

    void muestra() override {
        cout << "Alumno Licenciatura" << endl;
        cout << "Codigo: " << codigo << endl;
        cout << "Carrera: " << carrera << endl;
        cout << "Creditos: " << creditos << endl << endl;
    }
};

// Clase Pila modificada para usar ElementoPila*
class Pila {
private:
    list<ElementoPila*> lista;
public:
    ~Pila() {
        // Liberar la memoria de los elementos de la pila
        for (ElementoPila* elemento : lista) {
            delete elemento;
        }
    }

    void push(ElementoPila *x) {
        lista.push_front(x);
    }
```

```
    ElementoPila* pop() {
        ElementoPila* x = *lista.begin();
        lista.erase(lista.begin());
        return x;
    }

    ElementoPila* top() {
        return *lista.begin();
    }

    void muestra() {
        list<ElementoPila*>::reverse_iterator it;
        ElementoPila *x;
        cout << "Pila: ";
        for (it = lista.rbegin(); it != lista.rend(); it++) {
            x = *it;
            x->muestra();
        }
        cout << endl;
    }
};

// Función de ejemplo
void ejemplo() {
    Pila pila;
    pila.push(new Licenciatura("345678", "Computacion", 200));
    pila.push(new Bachillerato("456789", "Preparatoria 12"));
    pila.push(new Licenciatura("456789", "Informatica", 50));
    pila.muestra();
    cout << "********************************" << endl;
    pila.pop();
    pila.muestra();
}

int main() {
    ejemplo();
    return 0;
}
```

Según la información proporcionada, parece que la tarea se centra en la implementación de una pila (`Pila`) que maneja objetos de tipos diversos que heredan de la clase base `ElementoPila`. Estos tipos incluyen `Terminal`, `NoTerminal`, `Estado`, `Bachillerato`, y `Licenciatura`.

1. **Clases Base (`ElementoPila`):**
   - Se proporciona una clase base abstracta llamada `ElementoPila` con una función virtual pura llamada `muestra()`. Esta clase actúa como la base para las clases derivadas.

2. **Clases Derivadas (`Terminal`, `NoTerminal`, `Estado`, `Bachillerato`, `Licenciatura`):**
   - Se han creado varias clases derivadas que heredan de `ElementoPila`. Cada una de estas clases implementa la función `muestra()` de manera específica para su tipo.

3. **Pila (`Pila`):**
   - Se ha modificado la clase `Pila` para trabajar con punteros a `ElementoPila` en lugar de objetos directos.
   - Los métodos `push` y `pop` ahora trabajan con punteros. `push` agrega un puntero al frente de la lista, y `pop` elimina y devuelve el primer elemento de la lista como un puntero.
   - Se ha añadido un destructor para liberar la memoria de los elementos de la pila.

4. **Ejemplo en `main()`:**
   - Se proporciona una función de ejemplo llamada `ejemplo()` que crea una instancia de la clase `Pila`, realiza algunas operaciones de `push` y `pop` con objetos de `Licenciatura` y `Bachillerato`, y finalmente muestra el contenido de la pila.