Project 1 Lisp Interpreter

Sprint1: Lisp-like S-Expression Reader

Goal

Construct a program that reads and prints S-Expressions, including atoms and lists.

Language Requirements

- **Primary:** C
- **Acceptable Alternate:** C++ (without polymorphism or inheritance)
- **Other Languages:** May be considered with a 10-20% penalty on the final evaluation.

Assignment Details

1. Data Structure

Your primary task is to design a data structure capable of representing
both **atoms** and **consecutive cells** (also known as "cons cells"). This structure should be able to:

- Differentiate between an **atom** (number, symbol, or string).
- Represent a **cons cell** with two pointers: one to the `car` (the head of the list) and one to the `cdr` (the rest/tail of the list).
- Handle the special value **`nil`** (representing an empty list or the end of a list). Representing `nil` as `NULL` is a common approach, but you should consider if a different representation might simplify your code.

2. Core Methods

You need to implement the following methods to manage your data structure:

- **`cons(s-expression car, s-expression cdr)`**: This function should create and return a new cons cell.
  - `(cons 'a '(b c))` should result in `(a b c)`.
  - `(cons 'a '())` should result in `(a)`.
- **`car(s-expression list)`**: Returns the first element of a cons cell.
- **`cdr(s-expression list)`**: Returns the rest of the cons cell's list.

3. Reading and Printing S-Expressions

Your program should be able to:

- **Read Atoms:** Recognize and store numbers (longs or doubles), symbols (any sequence of non-blank, non-parenthesis characters that don't start with a quote), and strings (text enclosed in double quotes).
- **Read Lists:** Parse S-Expressions that are lists, including nested lists. For example:
  - `(a b c)`
  - `((a (7 8 9) x) (b (4 5 6) y))`
- **Print:** Display S-Expressions correctly.
  - **Lists:** A list like `(a b c)` should be printed with spaces and no trailing dot. `nil` at the end of a list is implied and should not be printed.
  - **Dotted Pairs:** If a list is not properly terminated with `nil`, it should be printed using **dotted notation**. For example, a cons cell structure `(a . (b . (c . d)))` should be printed as `(a b c . d)`. Your program does not need to read dotted pairs, but it must be able to print them if they occur.

Technical Notes

S-Expression Syntax (Simplified BNF)

- **S-EXPRESSION** => **ATOM** | **(LIST**
- **ATOM** => **number** | **symbol** | **string**
- **LIST** => **)** | **S-EXPRESSION LIST**

Key Concepts

- **Atoms:** The most basic elements of S-Expressions.
- **Cons Cells (Pairs):** The building blocks of lists.
  - `car` (or `head`): Points to the first S-Expression in a list.
  - `cdr` (or `tail`): Points to the rest of the list, which can be another list or `nil`.
- **Lists:** A chain of cons cells where the last `cdr` is `nil`.
- **`nil`:** A special value representing the empty list `()`. This is crucial for properly terminating lists.
- **Dotted Pairs:** A cons cell whose `cdr` is not a list and is not `nil`. These are not "proper" lists and must be printed with a dot. Your program should avoid creating them but must be able to print them.

This project will require you to understand and implement a fundamental concept of functional programming. Pay close attention to how you manage memory, especially with linked structures like cons cells.