Universidad del Valle de Guatemala

Algoritmos y Estructuras de Datos

Catedrático: Moisés Alonso

**Proyecto 1**

**Fase 1**

Alina Carías (22539)

Ignacio Méndez (22613)

Ariela Mishaan (22052)

Diego Soto (22737)

**LISP**

**Historia**

LISP es el segundo lenguaje más viejo de alto nivel, siguiendo únicamente a Fortran, que apareció poco tiempo antes. Lisp fue desarrollado alrededor de 1960 por John McCarthy en el Instituto de Tecnología de Massachusetts (MIT). El lenguaje fue fundamentado en la teoría matemática de las *funciones recursivas* (son las funciones que aparecen en su propia definición) (Hemmendinger, 2023).

Originalmente, Lisp fue fundado como una notación práctica para programas de computación, influenciado por la notación del cálculo de lambda de Alonzo Church (Code Docs, 2021). McCarthy inicialmente publicó el diseño de su lenguaje en un artículo de la revista *Communications* de la *ACM*, con el título “Recursive Functions and Symbolic Expressions and Their Computation by Machine, Part I”. En este artículo se mostró que con pocos operadores simples se puede construir un lenguaje completo para algoritmos (Code Docs, 2021).

La primera vez que Lisp fue implementado fue en una computadora IBM 704, por Steve Russell. El primer compilador completo de Lisp, escrito en Lisp, fue creado en 1962 por Tim Hart y Mike Levin en el MIT (Code Docs, 2021). Rápidamente se convirtió en el lenguaje preferido para la investigación de la inteligencia artificial. También fue Lisp el lenguaje que dio origen a varias ideas de la ciencia de la computación, como las estructuras de datos de árbol, recursión, entre otros (Code Docs, 2021).

El nombre LISP se deriva de “LISt Processor”, ya que las “linked lists” son una de las estructuras de datos más importantes del lenguaje. Lisp ha ido cambiando con los años, y de él se han derivado varios otros lenguajes: Common Lisp, Emacs Lisp (Elisp), Clojure, Racket, Scheme, Fennel y GNU Guile (Kenlon, 2021).

**Características**

LISP es un lenguaje de programación que se caracteriza por estar diseñado para continuar desarrollándose con la tecnología:

*Part of what makes Lisp distinctive is that it is designed to evolve. As new abstractions become popular (object-oriented programming, for example), it always turns out to be easy to implement them in Lisp. Like DNA, such a language does not go out of style* (Graham, s.f.)*.*

Otra característica principal de LISP es que tiene un estilo general organizado alrededor de expresiones y funciones, es decir, es un lenguaje funcional. Todo proceso que se lleva a cabo en LISP es una función, y cuando se le llama, retorna un dato (objeto) como valor (Geeks for Geeks, 2022).

LISP utiliza una notación muy simple, en la cual operaciones y sus operandos están dados en una lista de paréntesis (Postfix) (Hemmendinger, 2023).

Otras características de este lenguaje de programación (Geeks for Geeks, 2022):

* LISP es un lenguaje independiente de la máquina en que se implementa.
* Usa una metodología iterativa de diseño y es fácil de extender.
* Permite crear y actualizar programas y aplicaciones dinámicamente.
* Provee un alto nivel de debug.
* Es un lenguaje orientado a objetos (Graham, s.f.).

**Dónde es empleado**

Desde que fue creado, Lisp siempre ha estado relacionado a la investigación de inteligencia artificial, machine learning y quantum computing. Fuera del ámbito educativo o investigativo, lisp se usa también para implementar programas de inteligencia artificial, aplicaciones web, y otros softwares. Algunos de las empresas que actualmente utilizan Lisp son Grammarly (utiliza AI para las sugerencias y el análisis del texto), Boeing (usa el servidor Allegro NFS, escrito en Common Lisp) (Froehlich, s.f.), London Tube, Circle CI, AutoCAD (utiliza el dialecto AutoLISP como el lenguaje interno de desarrollo de aplicación), entre otros (Galinka, 2021).

**Programación funcional y POO**

En general los lenguajes de programación están divididos dentro de dos tipos generales: los que utilizan la programación declarativa y los que utilizan la imperativa. La programación imperativa consiste de instrucciones, que indican en qué orden y cómo se debe realizar la acción que se desea. Los lenguajes orientados a objetos son de este tipo.

Por el otro lado, en la programación declarativa se definen funciones, fórmulas o relaciones. Estos no especifican cómo las funciones trabajan paso a paso en un programa. Los lenguajes de programación funcional están incluidos en esta categoría, como Lisp.

Programación funcional significa escribir programas que trabajan retornando valores, en vez de provocando efectos secundarios (es decir, cambios destructivos a objetos y asignaciones a variables). Si estos efectos son pocos y localizados, los programas se convierten en fáciles de leer, probar y debugear.

En los lenguajes que utilizan este paradigma, las funciones son lo más importante. Pueden ser pasadas como parámetros y retornadas por otras funciones, como si fueran cualquier otro tipo de dato. Lisp es un lenguaje de programación funcional. Todas sus expresiones retornan algún valor y todos los procedimientos son sintácticamente una función, que, al ser llamada, retorna como valor un objeto (Healy).

LISP fue el primer representante del paradigma de la programación funcional. Contrario a los lenguajes de procedimiento y los orientados a objetos (cuyo modelo teórico de computación es la máquina Turing), el modelo teórico de computación de Lisp es el cálculo Lambda desarrollado por Alonzo Church (aunque ambos modelos son equivalentes en poder; cualquier algoritmo que se pueda expersar en un modelo se puede expresar también en el otro).

**Java Collections Framework**

Una colección es un objeto que representa a un grupo de objetos (como la clase Vector). Un “collections framework” permite manipular estas colecciones, independientemente de los detalles de implementación. Una de las principales ventajas del framework es que reduce la dificultad para los programadores, ya que provee estructuras de datos y algoritmos ya hechos. Todas las *collection frameworks* contienen (Java Documentation, s.f.):

* Interfaces: son tipos de dato abstractos que representan distintos tipos de colecciones (como listas, sets y mapas). Permiten que las colecciones sean manipuladas, independientemente de los detalles de su representación y son la base del framework. Hay dos tipos de interfaces:
  + Java.util.Colllection (la más básica)
  + Java.util.Map

Algunas de las interfaces son: Collection Interface, List Interface, Set, SortedSet, Map, Map.Entry, sorted Map y Enumeration (Tutorials Point, s.f.).

* Implementaciones: son clases que implementan las interfaces. Hay distintos tipos de implementaciones:
  + Implementaciones de propósito general
  + Implementaciones heredadas
  + Implementaciones para propósitos especiales.
  + Implementaciones concurrentes.
  + Implementaciones wrapper.
  + Implementaciones de conveniencia.
  + Implementaciones abstractas.

Algunas de las clases que implementan las interfaces del collection framework son; AbstractCollection, AbstractList, AbstractSequentialList, LinkedList, ArrayList, AbstractSet, HashSet, LinkedHashSet, TreeSet, AbstractMap, HashMap, TreeMap, WeakHashMap, LinkedHashMap y IdentityHashMap (Tutorials Point, s.f.).

* Algoritmos: son los métodos, pueden ser polimórficos. Son métodos estáticos que llevan a cabo funciones útiles sobre las colecciones (por ejemplo el de ordenar una lista).

Ejemplo: el paquete java.util. Este contiene todas las clases e interfaces del collection framework (Java T Point, s.f.):



**En este proyecto**

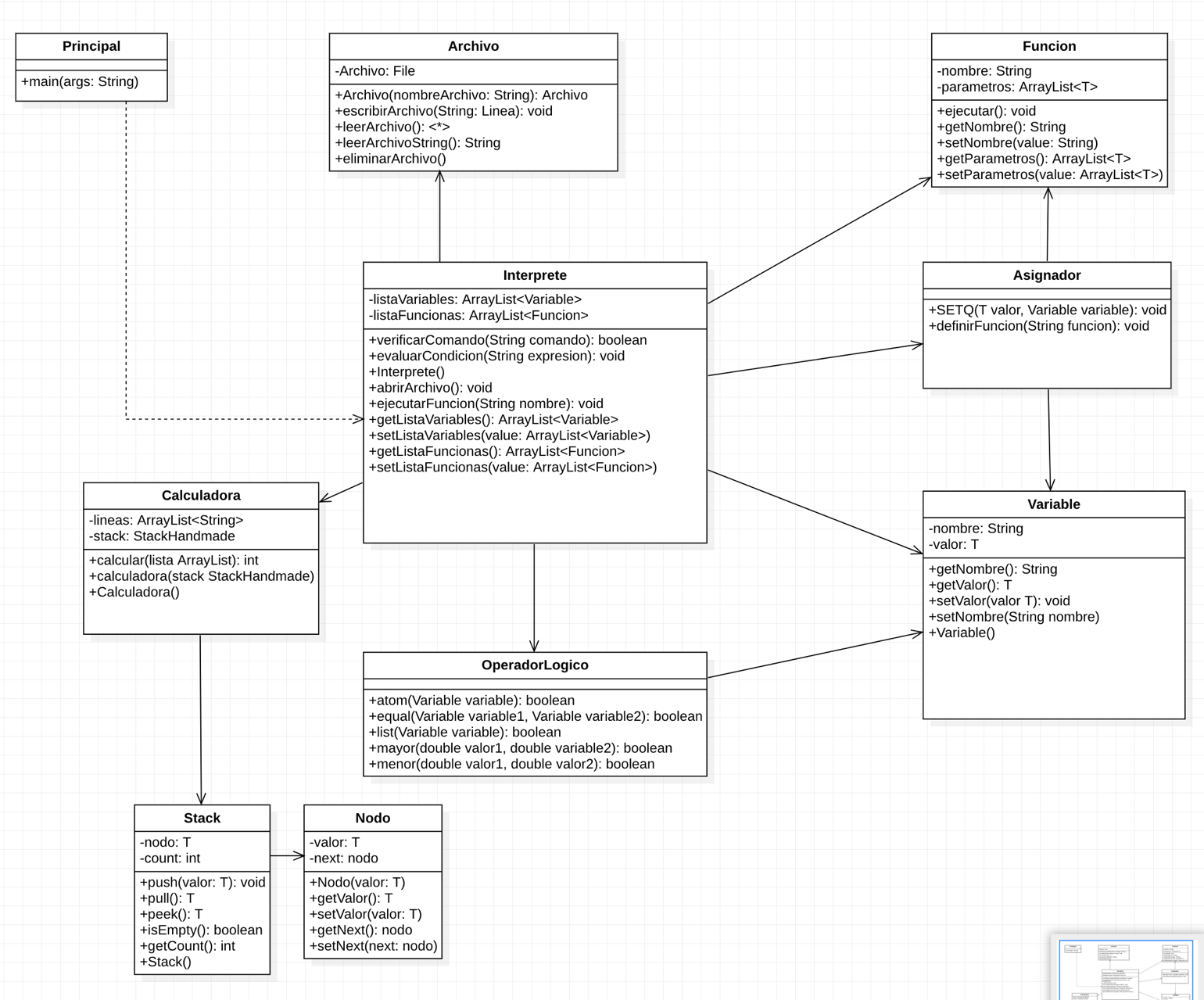
Para leer expresiones postfix se van a utilizar Stacks, con nodos. Además, se utilizará el ArrayList para mantener en el sistema una lista de variables y una de funciones. Para leer archivos y expresiones también se utilizarán ArrayLists y arreglos convencionales.

**Link al repositorio de Github:**

<https://github.com/ArielaMishaan/Proyecto-1.git>

(Nota: el repositorio está privado, si necesitan accesar a él y el link no les permite por favor darnos nombre de usuario en Github).

**Diseño del programa: UML**

**Diagrama de clases**

**Diagram

Description automatically generated with medium confidenceDiagrama de secuencia**

**Diagrama de casos de uso**

**Diagram

Description automatically generated**

**Referencias**

Graham, P. (s.f.). *Common Lisp*. Obtenido de Lisp Language Org: https://lisp-lang.org

Hemmendinger, D. (11 de enero de 2023). *LISP Computer Language*. Obtenido de Britannica: https://www.britannica.com/technology/LISP-computer-language

Geeks for Geeks. (17 de marzo de 2022). *Introduction to LISP*. Obtenido de Geeks for Geeks.org: https://www.geeksforgeeks.org/introduction-to-lisp/

Kenlon, S. (3 de mayo de 2021). *Learn the Lisp programming language in 2021*. Obtenido de Opensource.com: https://opensource.com/article/21/5/learn-lisp

Code Docs. (18 de junio de 2021). *Lisp*. Obtenido de CodeDocs.org: https://codedocs.org/what-is/lisp-programming-language

Froehlich, A. (s.f.). *Lisp (programming Language)*. Obtenido de WhatIs.com: https://www.techtarget.com/whatis/definition/LISP-list-processing

Galinka, C. (4 de octubre de 2021). *Do You Know Where Lisp Is Used Nowadays?* Obtenido de Typeable: https://typeable.io/blog/2021-10-04-lisp-usage.html

Healy, C. (s.f.). *Compare LISP with Object Oriented Programming Languages.* Pengfei Wang.

Java Documentation. (s.f.). *Collections Framework Overview*. Obtenido de Oracle Docs: https://docs.oracle.com/javase/8/docs/technotes/guides/collections/overview.html

Java T Point. (s.f.). *Collections in Java*. Obtenido de JavaTPoint: https://www.javatpoint.com/collections-in-java

Tutorials Point. (s.f.). *Java Collections Framework*. Obtenido de TutorialsPoint: https://www.tutorialspoint.com/java/java\_collections.htm

[**https://gigamonkeys.com/book/**](https://gigamonkeys.com/book/)

[**https://www.youtube.com/watch?v=pCK6prSq8aw**](https://www.youtube.com/watch?v=pCK6prSq8aw)