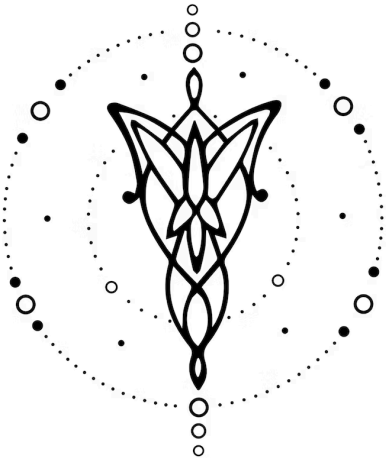

Especificación

Diseño e Implementación de un Lenguaje

v1.0.0

1. Equipo

Nombre	Apellido	Legajo	E-mail
Juan Cruz	Boullosa Gutierrez	63414	jboullosagutierrez@itba.edu.ar
Lucas	Gonzalez Rouco	63366	lgonzalezrouco@itba.edu.ar
Tobias	Ves Losada	63342	tveslosada@itba.edu.ar

2. Repositorio

La solución y su documentación serán versionadas en: **organigraph**.

3. Dominio

Desarrollar un lenguaje que permita crear organigramas para proyectos de una empresa, con proyectos y empleados con atributos (nombre, rol y metadata). El lenguaje debe permitir representar de una forma sencilla la estructura jerárquica del proyecto.

Una vez que se finaliza un proyecto, el organigrama ya no puede modificarse, pero hasta ese momento, se pueden agregar y eliminar empleados, como también visualizar el organigrama del proyecto en distintos momentos, por ejemplo, cuando había 2 empleados, cuando eran 10 y el resultado de cuando finalizó. También se permite la búsqueda de empleados por alguno de sus atributos. Como también la inserción de organigramas en uno nuevo. Por ejemplo, armar el árbol de jerarquía del proyecto 1 y luego lo inserto en el árbol de jerarquía del proyecto 2.

Los empleados y los proyectos serán codificados como simples objetos con variables (similares a formato JSON) para generar una sintaxis simple a la hora de definir alguno de los mismos.

La implementación satisfactoria de este lenguaje permitiría experimentar diferentes estructuras de trabajo para organizar el desarrollo de los proyectos de una empresa como también ver la evolución de dichos organigramas hasta la finalización del proyecto.

4. Construcciones

El lenguaje desarrollado debería ofrecer las siguientes construcciones, prestaciones y funcionalidades:

(I). Se podrán crear uno o varios proyectos.
(II). Se podrán crear uno o varios empleados.
(III). Se podrán asignar uno o varios empleados a un proyecto.
(IV). Se podrán crear y asignar empleados a un proyecto a la vez.
(V). Se podrá asignar un empleado a un proyecto indicando cierto orden en la jerarquía.
(VI). Se podrá finalizar un proyecto.
(VII). Se podrá eliminar a un empleado de un proyecto borrando recursivamente a sus hijos.
(VIII). Se podrá eliminar a un empleado de un proyecto y reemplazarlo por otro nuevo u otro sin subordinados.
(IX). Se podrá imprimir la jerarquía de cada proyecto.
(X). Se podrá imprimir la jerarquía de todos los proyectos a la vez.
(XI). Las variables podrán ser de tipo employee y project.
(XII). Las variables podrán ser vectores de alguno de los tipos anteriores.
(XIII). Se proveerán estructuras de control básicas de tipo FOR y WHILE.
(XIV). Se podrá agregar la estructura de un proyecto a otro.
(XV). Se podrá buscar a un empleado por el contenido de alguno de sus campos en un proyecto, devolviendo el subárbol que lo tiene como raíz.

5. Casos de Prueba

Se proponen los siguientes casos iniciales de prueba de **aceptación**:

(I). Un programa que cree un proyecto y le asigne 5 empleados.
(II). Un programa que cree 2 proyectos distintos.
(III). Un programa que cree un proyecto y le asigne 3 empleados uno por debajo del otro en relación a la jerarquía.
(IV). Un programa que cree un proyecto, le asigne empleados siguiendo cierta jerarquía e imprimirlo.
(V). Un programa que agregue empleados a un proyecto a partir de un vector de empleados utilizando un ciclo WHILE.
(VI). Un programa que agregue empleados a un proyecto a partir de un vector de empleados utilizando un ciclo FOR.
(VII). Un programa que elimine a un empleado de un proyecto al que esté asignado y lo reemplace por uno existente.
(VIII). Un programa que elimine a un empleado y todos sus subordinados del proyecto.
(IX). Un programa que copie la estructura de un proyecto y lo agregue a otra.
(X). Un programa que busque a un empleado por su nombre en un proyecto y devuelva el subárbol que tiene a dicho empleado como raíz
(XI). Un programa que busque a un empleado por su nombre en un proyecto y devuelva su información

Además, los siguientes casos de prueba de **rechazo**:

(I). Un programa que asigne un empleado a un proyecto inexistente
(II). Un programa que asigne un empleado inexistente a un proyecto
(III). Un programa que asigne un empleado a un proyecto estando ya en el mismo
(IV). Un programa que lea de un archivo que no existe
(V). Un programa que finalice un proyecto inexistente
(VI). Un programa que elimine a un empleado existente de un proyecto existente pero que no esté asignado al mismo

6. Ejemplos

Creación de un proyecto, con la asignación de empleados al mismo

```
// Crear un nuevo empleado
employee empleado3 {
    name = “Tomas Rodriguez”;
    role = “junior developer”;
}

// Crear un nuevo empleado
```

```
employee empleado4 {
    name = “Alba Martinez”;
    role = “tech lead”;
}

// Crear un nuevo proyecto
project mercadoLibre;

// Crear y asignar un empleado a un proyecto
employee empleado1 in mercadoLibre {
    name = ”Juan Perez”;
    role = ”senior developer”;
};

// Crear y asignar un empleado a un proyecto bajo cierto empleado
employee empleado2 in mercadoLibre under empleado1  {
    name = ”Martin Gonzalez”;
    role = ”junior developer”;
};

// Asignar un empleado a un proyecto bajo cierto empleado
empleado3 in mercadoLibre under empleado1;

// Asignar un empleado a un proyecto bajo cierto empleado
empleado4 in mercadoLibre under empleado1;

// Asignar un empleado a un proyecto bajo cierto empleado
empleado5 in mercadoLibre under empleado2, empleado3 {
    name = ”Martin Gonzalez”;
    role = ”junior developer”;
};
```

Finalización de un proyecto

```
project mercadoLibre;

(...)

finish mercadoLibre;
```

Remover a un empleado y sus subordinados de un proyecto

```
// Crear un nuevo empleado
employee empleado {
    name = “Alba Martinez”;
    role = “tech lead”;
}
// Crear un nuevo proyecto
project mercadoLibre;
```

```
// Asignar un empleado a un proyecto bajo cierto empleado
empleado in mercadoLibre;

// Asignar un empleado subordinado a Alba Martinez
employee empleado2 in mercadoLibre under empleado {
    name = ”Martin Gonzalez”;
    role = ”junior developer”;
};

// Eliminar un empleado y sus subordinados de un proyecto
remove empleado from mercadoLibre
```

Remover y reemplazar a un empleado

```
// Crear un nuevo empleado
employee empleado {
    name = “Alba Martinez”;
    role = “tech lead”;
}
// Crear un nuevo proyecto
project mercadoLibre;

// Asignar un empleado a un proyecto bajo cierto empleado
empleado in mercadoLibre;

// Asignar un empleado subordinado a Alba Martinez
employee empleado2 in mercadoLibre under empleado  {
    name = ”Martin Gonzalez”;
    role = ”junior developer”;
};
// Crear un nuevo empleado
employee empleado3 {
    name = “Tomas Rodriguez”;
    role = “junior developer”;
}

// Elimino a Alba Martinez y la reemplazo por Tomas Rodriguez
replace empleado from mercadoLibre with empleado3

// Obtener un vector con todos los subarboles que tienen un empleado raiz que se
llama Tomas Rodriguez
employee empleados1[] = SearchByNameFrom mercadolibre(“Tomas Rodriguez”);

// Obtener un vector con todos los subarboles que tienen un empleado raiz que
tiene el rol senior developer
employee empleados2[] = SearchByRoleFrom mercadolibre(“senior developer”);

// Obtener un vector con todos los subarboles que tienen un empleado raiz que
tiene como metadata hola
```

```
employee empleados3[] = SearchByDataFrom mercadolibre(“hola”);

employee empleado5 = empleados1[0];

// Añade a la estructura del proyecto mercadolibre el árbol que tiene como raíz
al empleado5 y todos lo que le responden
empleado5 in mercadolibre withSubordinates under empleado1;
```