**Tecnológico de Monterrey, Campus Monterrey**



Compiler Design

group 1

|  |  |  |
| --- | --- | --- |
| Guillermo A.  Mendoza Soni |  | William Miguel  Rosado Aíza |
| A01550742 |  | A00815329 |

[**Vision of the project**](#_u8g6v6oot4ik) **3**

[**Language main objective**](#_o9tsqiw0j2cs) **3**

[**Language requirements**](#_7g76gqm9surm) **4**

[Basic Elements (tokens)](#_ng2k9d6iu1iz) 4

[Basic Elements (tokens)](#_32la4j8wdwky) 9

[Syntax Diagrams](#_cimborofok32) 12

[Semantic characteristics](#_m12nkrqvgzc0) 15

**a). DESCRIPCION DEL PROYECTO:**

# **Vision of the project**

The purpose of the language will be to be able to write in a familiar and friendly way the manipulation ad usage of a graph. These graphs can be of any kind (as long as declared properly).

We want anyone using the language to feel at ease when declaring a graph and using any property of a graph in a familiar or understandable / intuitive way.

# **Language main objective**

The main objective for the language will be to be able to declare graphs as a kind of variable by itself, and be able to add nodes to this graph in an intuitive / understandable way. With a graph, a user can add nodes, make “basic” graph operations. Display a graph (first on terminal, or atleast the components of said graph). Etc. We believe the language can be considered an assist comprehensive tool for graphs.

# 

# **Language requirements**

## Basic Elements (tokens)

We believe that for a graphs oriented language, we need to be able to store graphs as a style of variable.

Ids

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **Example** |
| Graph | A basic element for the user to refer to a graph they declared. This graph will consist of the main variable they can use. Graphs can be identified by a starting letter in Case followed by a combination of numbers and lower case letters. They are also delimited by the keyword of a graph which is either “directed”, or “undirected” | directed Mygraph2 |
| Node | Nodes will be the back structure of a graph. Graphs consist of a series of nodes connected between themselves. They are delimited by the keyword “node”. | node 1inicial |
| Arcs | Arcs are the connection between nodes. Arcs connect nodes in either a unidirectional way or bi directional. Can sometimes have a weight. They will be delimited by the key word “arc”. Consist of a lower case letter which can be followed by numbers numbers | arc m23 |
| Id | Will be to declare any other type of id for all “clasic” values. Are characterized by starting with a lower case character and followed by a combination of upper case, lower case and numbers. | int myInt  float myFloat |
| constant | Will be to declare any other type of id for all “clasic” values. Are characterized by starting with a lower case character and followed by a combination of upper case, lower case and numbers. CONSTANTS CANNOT BE CHANGED ONCE ASSIGNED | Int cons myCons |

Keywords used

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Description** | **Example** |
| “directed” | Delimits the beginning of a declaration of a Directed Graph. | directed Mygraph2 |
| “undirected” | Delimits the beginning of a declaration of an Undirected Graph | Undirected Mygraph1 |
| “node” | Delimits the beginning of the declaration of a Node | node 1inicial |
| “arc” | Delimits the beginning of the declaration of an Arc | arc m23 |
| = | Assignation of a value to a id. | 1inicial = 98 |
| + | Will represent the operation union between two graphs. Resulting in a new graph containing the unique nodes of both individual graphs. | Mygraph3 = Mygraph1 + Mygraph2 |
| - | Will represent the operation substracion of a graph - the contents of the second graph. | Mygraph3 = Mygraph1 - Mygraph2 |
| / | Will represent the operation of intersection between two graphs. | Mygraph3 = Mygraph1 / Mygraph2 |
| “print” | Denoted keyword for printing a network. | print Mygraph3 |
| “deg” | Returns as a number the degree of a network. | deg Mygraph3 |
| “Spath” “in” | Returns the shortest path of two given nodes in a given graph | spath 1inicial 1segundo in Mygraph2 |
| “diam” | Returns the longest “shortest path” in a graph. Erogo the highest distance you will have to travel from one point to the other | diam Mygraph1 |
| int | To declare integer numbers. Only Integers. No floats | int myInteger |
| string | Declaration of series of characters. Can include numbers inside. Characterised to have “” between them. | string myString |
| float | Decimal numbers. Characterized to be numbers separated by a point. | float myFloat |
| bool | Value which can store a True or False statement. True or False are boolean values and will be corrected as such | bool myTrue |
| func | Keyword to declare a function in a lenguaje. After keyword, we need the type of value it will return. Can start with a lowercase letter and followed with a combination of lower, upper and numbers. Followed by parenthesis to permit parameters. | func int myFunc() |
| if | Function to realize a statement in case there is a true boolean inside the given parenthesis | If (bool) |
| while | Looping function to loop while there is a true statement on the permited value. Followed by a parenthesis where it will check the boolean statement | while (bool) |
| for | Looping function to loop while there is a true statement inside of the given parenthesis. Can also loop a graph, going through all nodes. | for (v in var/int)  for (n graph) |

**a.3) Descripción de los principales Test Cases.**

**a.4) Descripción del PROCESO general seguido para el desarrollo del proyecto, incluyendo Bitácoras generales y un pequeño párrafo de reflexión de cada alumno, en relación a los principales aprendizajes logrados (firmarlo).**

During the creation of the project, we were able to follow the suggested calendar by our teacher. We present the different logs created on every deliverable.

#1

For this sprint, we were able to accomplish finishing the Lex and Yacc equivalent for python in our system. Python can use the library PLY to create an almost identical simulation. Although we finished the lexicon and grammatical rules, we were not able to fully test our rules. The main file for the project so far is called: yacc\_c.pyAll other files are automatically created by the PLY library, and are usually used for debugging. The file called: “tests.txt” contains several of our tests realized on our simulation. We are yet to finish some testing cases. For example: Expressions, Arrays, Method calling Other than that, the grammatical and lexical rules work almost perfectly. We are happy with our development so far.

#2

For this sprint we designed the structure capable of holding the directions and the variable table for each direction, as well as designing the entirety of an oracle using only numbers to determine what kind of variables are allowed to use-certain operators. We believe we are right on schedule as assigned by the calendar. We were able to test first the cube by using different iterations of number, testing if the method worked. Next we designed the tables for variables and tried declaring different contexts, succesfully.

#3

For this sprint we were able to achieve the following:

-Associate a variable with an id, defendant of the context.

-Associate different constants to its corresponding space in memory

-create quadruples based on the expressions created

-create quadruples for assignation and printing

It should be noted that the semantic cube to verify the times matching was created in the last sprint.

We were able to test the base results for expressions yet we are missing some “essential” tests

#4

We are able to create quadrupples for the transicion between all possible cyclying statuses.

If, IF ELSE, WHILE, DOWHILE, FOR

we were able to finish this by the 30th of march, and we werent able to submit a copy of our advance

#5

We now include an "END" tag for the end of all quadruples. We updated our class tables to keep a better track of our functions

and their corresponding variables. the class is now improved to handle any kind of change.

Included a function to return the memory used at that time. used to store memry used for functions.

Create rutinary calls for function calling and declaration, including ERA, GOSUB, PARAM, RETURN

detected a new bug on our lenguaje. For some reason we are unable to call empty functions. We arent completely sure

of the reason for this misbehavior and are still tryig to find a reason. We have narrowed it down to possible reasons why this is happening. (verified debugging)

#6

Para este avance se desarrolló la máquina virtual. Lo que la máquina virtual hasta ahora es capaz de hacer es: + Resolver operaciones aritméticas + Resolver expresiones boleanas + Ejecutar comando 'PRINT' + Ejecutar saltos: GOTO, GOTOF, GOTOV + Ejecutar las condiciones if-else La máquina aún presenta fallas con la operación de asignación (ejemplo: mi\_VariableA = 10;). Por lo tanto aún no se puede comprobar que funcionan los estatutos de ciclo. El control de la máquina virtual se maneja a manera de clase, de tal modo que se pueda mantener su funcionamiento independiente de la ejecución del compilador. Todos los archivos de prueba utilizados se encuentran en la carpeta "tests"

#7

For this deliverable, we were able to fully add array storage and accesing. Our arrays are only given an integer to define the size and and int constant or int id to access. Quadruples are generated to verify that this can be accessed correctly. as well as semantic rules were creted to verify if more dimensions were declared than were tried to access and viceversa. Conditional quadruples on the VM were done last deliverable

#8

**b). DESCRIPCION DEL LENGUAJE:**

**b.1) Nombre del lenguaje.**

**b.2) Descripción genérica de las principales características del lenguaje (en forma narrativa).**

**b.3) Listado de los errores que pueden ocurrir, tanto en compilación como en ejecución.**

Our compiler is called Graph Comp. The purpose of this lenguaje is to have a simple functional language which as a plus is able to manage graphs as a simple object, and manipulate them easily. You are able to declare simple variables and functions as in any simple programming language. The usage of graphs is an added bonus.

Known errors:

**c). DESCRIPCION DEL COMPILADOR:**

**c.1) Equipo de cómputo, lenguaje y utilerías especiales usadas en el desarrollo del proyecto.**

Main language to develop the compiler will be in python. The computers used during the development will be the personal computers of each student: a macbook pro 2012 and a Hewlett-Packard 2014. During the production of the compiler, the macbook pro 2012 was changed to an Acer predator 2017.

**c.2) Descripción del Análisis de Léxico. Debe incluir: o Patrones de Construcción (expresados con Expresiones Regulares) de los elementos principales. o Enumeración de los "tokens" del lenguaje y su código asociado.**

## Basic Elements (tokens)

We believe that for a graphs oriented language, we need to be able to store graphs as a style of variable.

Ids

|  |  |  |
| --- | --- | --- |
| **ID** | **Description** | **Example** |
| Graph | A basic element for the user to refer to a graph they declared. This graph will consist of the main variable they can use. Graphs can be identified by a starting letter in Case followed by a combination of numbers and lower case letters. They are also delimited by the keyword of a graph which is either “directed”, or “undirected” | directed Mygraph2 |
| Node | Nodes will be the back structure of a graph. Graphs consist of a series of nodes connected between themselves. They are delimited by the keyword “node”. | node 1inicial |
| Arcs | Arcs are the connection between nodes. Arcs connect nodes in either a unidirectional way or bi directional. Can sometimes have a weight. They will be delimited by the key word “arc”. Consist of a lower case letter which can be followed by numbers numbers | arc m23 |
| Id | Will be to declare any other type of id for all “clasic” values. Are characterized by starting with a lower case character and followed by a combination of upper case, lower case and numbers. | int myInt  float myFloat |
| constant | Will be to declare any other type of id for all “clasic” values. Are characterized by starting with a lower case character and followed by a combination of upper case, lower case and numbers. CONSTANTS CANNOT BE CHANGED ONCE ASSIGNED | Int cons myCons |

Keywords used

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Description** | **Example** |
| “directed” | Delimits the beginning of a declaration of a Directed Graph. | directed Mygraph2 |
| “undirected” | Delimits the beginning of a declaration of an Undirected Graph | Undirected Mygraph1 |
| “node” | Delimits the beginning of the declaration of a Node | node 1inicial |
| “arc” | Delimits the beginning of the declaration of an Arc | arc m23 |
| = | Assignation of a value to a id. | 1inicial = 98 |
| + | Will represent the operation union between two graphs. Resulting in a new graph containing the unique nodes of both individual graphs. | Mygraph3 = Mygraph1 + Mygraph2 |
| - | Will represent the operation substracion of a graph - the contents of the second graph. | Mygraph3 = Mygraph1 - Mygraph2 |
| / | Will represent the operation of intersection between two graphs. | Mygraph3 = Mygraph1 / Mygraph2 |
| “print” | Denoted keyword for printing a network. | print Mygraph3 |
| “deg” | Returns as a number the degree of a network. | deg Mygraph3 |
| “Spath” “in” | Returns the shortest path of two given nodes in a given graph | spath 1inicial 1segundo in Mygraph2 |
| “diam” | Returns the longest “shortest path” in a graph. Erogo the highest distance you will have to travel from one point to the other | diam Mygraph1 |
| int | To declare integer numbers. Only Integers. No floats | int myInteger |
| string | Declaration of series of characters. Can include numbers inside. Characterised to have “” between them. | string myString |
| float | Decimal numbers. Characterized to be numbers separated by a point. | float myFloat |
| bool | Value which can store a True or False statement. True or False are boolean values and will be corrected as such | bool myTrue |
| func | Keyword to declare a function in a lenguaje. After keyword, we need the type of value it will return. Can start with a lowercase letter and followed with a combination of lower, upper and numbers. Followed by parenthesis to permit parameters. | func int myFunc() |
| if | Function to realize a statement in case there is a true boolean inside the given parenthesis | If (bool) |
| while | Looping function to loop while there is a true statement on the permited value. Followed by a parenthesis where it will check the boolean statement | while (bool) |
| for | Looping function to loop while there is a true statement inside of the given parenthesis. Can also loop a graph, going through all nodes. | for (v in var/int)  for (n graph) |

Important Regular expressions:

**c.3) Descripción del Análisis de Sintaxis. Debe incluir: o Gramática Formal empleada para representar las estructuras sintácticas (Sin “codificar”).**

## Syntax Diagrams

**c.4) Descripción de Generación de Código Intermedio y Análisis Semántico. Debe incluir: o Código de operación y direcciones virtuales asociadas a los elementos del código. o Diagramas de Sintaxis con las acciones correspondientes. o Breve descripción de cada una de las acciones semánticas y de código (no más de 2 líneas). o Tabla de consideraciones semánticas.**

## Semantic characteristics

|  |  |  |
| --- | --- | --- |
| **Rule** | **Description** | **Example** |
| *Operation restriction* | Only graphs can be parts of the simple operators like +, -. | Directed + Directed = Directed  Undirected + Undirected = Undirected |
| *Graph compatibility* | In the case of an operation happening between a directed and an undirected graph, the result will be Directed. (Arcs from the directed will transform with a cost of 0) | Directed + Undirected = Directed |
| *Variable depth* | We will manage two kinds of variable. Global and Local in 1 depth. We can have a global variable with a name and a variable in the main with the same name. The lookup table will refer first to the local variable rather than the global. | Directed x  Main{  Undirected x  } |
| *Undeclared variables* | Undeclared variables should and will not be used. In case of the usage of an undeclared variable, we will return an error | Node x  graphx.add({x, y}, 0)  [ERROR undeclared x] |
| *Declaration of same name on the same level* | In case a variable is declared again, the last declaration will NOT override and will return an error. | node x  arc x  [Error. Name already taken on this level.] |
| *Parameters* | Parameters in special functions will only take the types declared and in the intended way. If a parameter is not of the type which was declared, an error will be returned | Graph.add({node, node}, node)  [error, expected arc] |

**c.5) Descripción detallada del proceso de Administración de Memoria usado en la compilación. o Especificación gráfica de CADA estructura de datos usada (Dir.Funciones, Tablas de Var's, Cuádruplos, etc...)**

The directory for functions is managed with a dictionary. Each function is a distinctive key and easily accessed. Inside of the dictionary, we manage a dictionary able to store specific information. Where the function starts, if it returns something, etc.

Between these specifications, we store the variable table as a dictionary of its own. Each variable for the function is it own key. Every variable is able to store their direction, between other values.

Each of these dictionaries were created inside a class, which contains several functions which allow us to easily verify if a variable exists, between other jobs.

Quadruples on the other hand were created as an array of simple objects called quadruple. Each object contains four spaces for each of the components of the quadruples, which allows us simple access.

**d). DESCRIPCION DE LA MÁQUINA VIRTUAL:**

**d.1) Equipo de cómputo, lenguaje y utilerías especiales usadas (en caso de ser diferente que el compilador).**

**d.2) Descripción detallada del proceso de Administración de Memoria en ejecución (Arquitectura). Incluir: o Especificación gráfica de CADA estructura de datos usada para manejo de scopes (Memoria Local, global, etc..) o Asociación hecha entre las direcciones virtuales (compliación) y las reales (ejecución).**

**e). PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE :**

**e.1) Incluir pruebas que "comprueben” el funcionamiento del proyecto: o Codificación de la prueba (en su lenguaje). o Resultados arrojados por la generación de código intermedio y por la ejecución.**

**f). LISTADOS PERFECTAMENTE DOCUMENTADOS DEL PROYECTO:**

**f.1) Incluir comentarios de Documentación, es decir: para cada módulo, una pequeña explicación de qué hace, qué parámetros recibe, qué genera como salida y cuáles son los módulos más importantes que hacen uso de él.**

**f.2) Dentro de los módulos principales se esperan comentarios de Implementación, es decir: pequeña descripción de cuál es la función de algún estatuto que sea importante de ese módulo**.