Par++

25/Noviembre/2020

Noé Amador Campos Castillo

A008

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Mariana Estefanía Villegas Vázquez

A00513571

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

INDICE

**DESCRIPCIÓN DEL PROYECTO**

1. Propósito y Alcance del Proyecto.

El objetivo de este proyecto es realizar un compilador para el cual se definió un lenguaje de programación que cumple las acciones básicas que usualmente se utilizan en otros lenguajes, se desarrollaron diferentes actividades tales como el léxico, sintaxis, generación de código intermedia para validaciones semánticas, asignación de memoria, estatutos secuenciales, estatutos condicionales, ciclos, funciones, tipos estructurados , así como también simulación de memoria y simulación de maquina virtual, teniendo un alcance para poder soportar operaciones aritméticas, funciones tipo void, int, float y char, así como arreglos y matrices para los tipos estructurados, para los estatutos que pueden ser usados se cuenta con asignaciones, llamadas de funciones, lectura, escritura, decisión y repetición, los operadores que pueden ser utilizados son para comparar y relacionar expresiones, en la secciones siguientes se detallaran a fondo la especificación de cada una de estas acciones.

1. Análisis de Requerimientos y Casos de Uso generales.
2. Descripción de los principales Test Cases.
3. 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 (DEBEN firmarlo).

DESCRIPCION DEL LENGUAJE:

Nombre del lenguaje: PAR ++

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

Listado de los errores que pueden ocurrir, tanto en compilación como en ejecución.

DESCRIPCION DEL COMPILADOR:

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

Python

* Descripción del Análisis de Léxico. Debe incluir:
  + Patrones de Construcción (expresados con Expresiones Regulares) de los elementos principales.

Strings = "([^"])"

ID = [a-zA-Z\_][a-zA-Z\_0-9]\*

Float = \d+\.\d+

Int = \d+

Bool = (true|false)

Nueva línea = \n+

Char = "([^"]\*)"

* + Enumeración de los "tokens" del lenguaje y su código asociado (ej: { : “left-bracket”).

OPERADORES ARITMÉTICOS

- : “MINUS”

+ : “PLUS”

\* : “MULTIPLY”

/ : “DIVIDE”

% : “MOD”

++ : “INCREMENT”

“--" : “DECREMENT”

OPERADORES RELACIONALES

== : “COMPARISON”

!= : “DIFFERENT”

> : “GREATERTHAN”

< : “LESSTHAN”

>= : “GREATERHANOREQUAL”

<= : “LESSTHANOREQUAL”

OPERADORES LÓGICOS

& : “AND”

| : “OR”

OPERADORES DE ASIGNACIÓN

= : “EQUALS”

+= : “PLUSEQUALS”

-= : “SUBSTRACTEQUALS”

OTROS

, : “COMMA”

{ : “LEFTBRACKET”

} : “RIGHTBRACKET”

[ : “LEFTSQRBRACKET”

] : “RIGHTSQRBRACKET”

( : “LEFTPARENTHESIS”

) : “RIGHTPARENTHESIS”

; : “SEMICOLON”

PALABRAS RESERVADAS:

reserved = {

    'int': 'INT',

    'float': 'FLOAT',

    'char': 'CHAR',

    'void': 'VOID',

    'var': 'VAR',

    'module': 'MODULE',

    'return': 'RETURN',

    'while': 'WHILE',

    'for': 'FOR',

    'do': 'DO',

    'to': 'TO',

    'if': 'IF',

    'then': 'THEN',

    'else': 'ELSE',

    'read': 'READ',

    'write': 'WRITE',

    'program': 'PROGRAM',

    'main': 'MAIN'

}

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

**PROGRAMA**: Program id; main() BLOQUE

| Program id; VARS main() BLOQUE

| Program id; FUNCIONES main() BLOQUE

| Program id; VARS FUNCIONES main() BLOQUE

**VARS**: var TIPO : VARS\_ID ;

**VARS\_ID**: id VARS\_ID

| id DIMEN\_DECLARA VARS\_ID

| id DIMEN\_DECLARA DIMEN\_DECLARA VARS\_ID

**TIPO**: int | float | char

**DIMEN\_DECLARA**: [ cte i ]

**FUNCIONES**: FUNC\_RETURN FUNCIONES

| FUNC\_VOID FUNCIONES

**FUNC\_RETURN**: module TIPO id PARÁMETROS FUNC\_VARS { FUNC\_RETURN\_ESTATUTO return ( EXP ) }

**FUNC\_VOID:** module id PARÁMETROSFUNC\_VARS { BLOQUE }

**FUNC\_VARS:** VARS

| E

**FUNC\_RETURN\_ESTATUTO:** ESTATUTO FUNC\_RETURN\_ESTATUTO

**PARÁMETROS:** ( TIPO id PARAM\_TIPO )

**PARAM\_TIPO:** , TIPO id PARAM\_TIPO

| E

**BLOQUE:** { ESTATUTO\_BLOQUE }

**ESTATUTO\_BLOQUE:** ESTATUTO

| ESTATUTO ESTATUTO\_BLOQUE

**ESTATUTO:** ASIGNACION

| LLAMADA\_FUNC

| LECTURA

| ESCRITURA

| DESICIÓN

| REPETICIÓN

ASIGNACIÓN: id DIMENSIONES = EXPRESION ;

| id = EXPRESION ;

LLAMADA\_FUNC: id PARÁMETROS ;

LECTURA: read ( LECTURA\_ELEM ) ;

LECTURA\_ELEM: id DIMENSIONES

| id DIMENSIONES , LECTURA\_ELEM

ESCRITURA: write ( ESCRITURA\_ELEM ) ;

ESCRITURA\_ELEM: cte string

| EXPRESION

| cte string , ESCRITURA\_ELEM

| EXPRESION , ESCRITURA\_ELEM

DECISION

REPETICION

CONDICIONAL

NO-CONDICIONAL

EXPRESION

EXP

TERMINO

OPERA\_EXP

OPERA\_TERM

FACTOR

VAR\_OPT

* Descripción de Generación de Código Intermedio y Análisis Semántico. Debe incluir:
  + Código de operación y direcciones virtuales asociadas a los elementos del código.
  + Diagramas de Sintaxis con las acciones correspondientes marcadas sobre ellos.
  + Breve descripción de cada una de las acciones semánticas y de generación de código (no más de 2 líneas).
  + Tabla de consideraciones semánticas (combinaciones factibles y errores de tipo).

|  |  |  |  |
| --- | --- | --- | --- |
| **CUBO SEMÁNTICO** | | | |
| **Operando Izquierdo** | **Operando Derecho** | **Operador** | **Resultado** |
| int | int | - | int |
| int | int | + | int |
| int | int | \* | int |
| int | int | / | float |
| int | int | % | int |
| int | int | == | bool |
| int | int | >= | bool |
| int | int | <= | bool |
| int | int | > | bool |
| int | int | < | bool |
| int | int | != | bool |
| int | int | & | err |
| int | int | | | err |
| int | float | - | float |
| int | float | + | float |
| int | float | \* | float |
| int | float | / | float |
| int | float | % | float |
| int | float | == | bool |
| int | float | >= | bool |
| int | float | <= | bool |
| int | float | > | bool |
| int | float | < | bool |
| int | float | != | bool |
| int | float | & | err |
| int | float | | | err |
| int | char | - | err |
| int | char | + | err |
| int | char | \* | err |
| int | char | / | err |
| int | char | % | err |
| int | char | == | err |
| int | char | >= | err |
| int | char | <= | err |
| int | char | > | err |
| int | char | < | err |
| int | char | != | err |
| int | char | & | err |
| int | char | | | err |
| float | int | - | float |
| float | int | + | float |
| float | int | \* | float |
| float | int | / | float |
| float | int | % | float |
| float | int | == | bool |
| float | int | >= | bool |
| float | int | <= | bool |
| float | int | > | bool |
| float | int | < | bool |
| float | int | != | bool |
| float | int | & | err |
| float | int | | | err |
| float | float | - | float |
| float | float | + | float |
| float | float | \* | float |
| float | float | / | float |
| float | float | % | float |
| float | float | == | bool |
| float | float | >= | bool |
| float | float | <= | bool |
| float | float | > | bool |
| float | float | < | bool |
| float | float | != | bool |
| float | float | & | err |
| float | float | | | err |
| float | char | - | err |
| float | char | + | err |
| float | char | \* | err |
| float | char | / | err |
| float | char | % | err |
| float | char | == | err |
| float | char | >= | err |
| float | char | <= | err |
| float | char | > | err |
| float | char | < | err |
| float | char | != | err |
| float | char | & | err |
| float | char | | | err |
| char | int | - | err |
| char | int | + | err |
| char | int | \* | err |
| char | int | / | err |
| char | int | % | err |
| char | int | == | err |
| char | int | >= | err |
| char | int | <= | err |
| char | int | > | err |
| char | int | < | err |
| char | int | != | err |
| char | int | & | err |
| char | int | | | err |
| char | float | - | err |
| char | float | + | err |
| char | float | \* | err |
| char | float | / | err |
| char | float | % | err |
| char | float | == | err |
| char | float | >= | err |
| char | float | <= | err |
| char | float | > | err |
| char | float | < | err |
| char | float | != | err |
| char | float | & | err |
| char | float | | | err |
| char | char | - | err |
| char | char | + | err |
| char | char | \* | err |
| char | char | / | err |
| char | char | % | err |
| char | char | == | bool |
| char | char | >= | err |
| char | char | <= | err |
| char | char | > | err |
| char | char | < | err |
| char | char | != | bool |
| char | char | & | err |
| char | char | | | err |
| bool | bool | == | bool |
| bool | bool | != | bool |
| bool | bool | & | bool |
| bool | bool | | | bool |
| bool | bool | = | bool |

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

DESCRIPCION DE LA MÁQUINA VIRTUAL:

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

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).

PRUEBAS DEL FUNCIONAMIENTO DEL LENGUAJE :

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.