



ITESM - Campus Santa Fe

Diseño de compiladores (TC3048.1)

Profesora: Maricela Quintana López

Artemis 22



Entrega Final

01/12/2022

Por:

Carla Pérez Gavilán Del Castillo - A01023033

Vicente Santamaría Velasco - A01421801

Juan Carlos Hurtado Andrade - A01025193

ÍNDICE

I. Breve Descripción	2
¿En qué se basaron?	2
¿Qué reglas siguen?	2
¿En qué lenguaje lo realizaron?	3
¿Cumplieron con las características mínimas solicitadas?	3
II. Análisis léxico	3
¿Cuáles son sus tokens?	3
¿Cómo serán los tipos que se aceptarán?	4
¿Cuáles serán las palabras reservadas del lenguaje?	5
III. Análisis Sintáctico	6
Definición formal de la gramática	6
IV. Manual del usuario	8
¿Cómo usar el compilador? link a github	8
Archivos que componen el compilador	8
Comandos para correr el programa en terminal	8
Comandos para correr el programa desde un archivo .txt:	9
¿Cómo programar Artemis 22?	9
1. ¿Cómo declarar una variable?	9
2. ¿Cómo asignar una variable?	9
3. ¿Cómo definir una función (con return)?	10
4. ¿Cómo hacer un ciclo?	10
5. ¿Cómo hacer un condicional (if/else)?	10
6. ¿Cómo hacer un arreglo?	11

I. Breve Descripción

¿En qué se basaron?

Artemis22 está basado en C++, pero retoma varios aspectos de Javascript y Python con el fin de simplificar su sintaxis y facilitar su uso. Su temática retoma los temas de un viaje espacial, especialmente en el uso de palabras reservadas. En cuanto a sus tipos de datos y manejo de operaciones intenta tener las mismas funcionalidades que los lenguajes de programación más comunes, pero con especial enfoque en el ámbito astronómico. En otras palabras, busca ser un lenguaje especializado para viajes espaciales.

¿Qué reglas siguen?

Nuestro lenguaje tendrá como temática el espacio, por lo tanto todas las palabras reservadas están relacionadas con el despegue de un cohete. Algunas de las reglas de nuestro lenguaje incluyen:

1. Los tipos de datos se sustituyen por las siguientes palabras reservadas:
 - real: **space-coordinates**
 - booleanos: **mission-status**
 - carácter: **message**
 - entero: **code**
2. Las palabras reservadas como *for*, *if*, *while* y *else* se sustituyen por palabras relacionadas con el espacio:
 - for: **countdown**
 - if: **ignition**
 - while: **autopilot**
 - else: **abort**
3. Un arreglo se define o inicializa con la palabra reservada “**flight-journal**”.
4. Mientras que una estructura se le define con la palabra reservada “**ufo**” en lugar de la *struct* como su forma de definirlo en C.
5. El resto de la lógica se asimila al lenguaje C + + en el uso de llaves para delimitar funciones, estructuras de datos, ciclos y condicionales.
6. La lectura y escritura de archivos de entrada se realizan con las palabras reservadas: **out** para leer y **over** para escribir
7. El lenguaje va a determinar el término de la sentencia usando punto y coma “;”.
8. El fin del programa debe leer “**onesmallstep**”.

¿En qué lenguaje lo realizaron?

Nuestro lenguaje de programación está basado en la NASA, y para poder construirlo utilizamos Yacc para la parte sintáctica, Lex para la parte léxica y C para el compilado del lenguaje.

¿Cumplieron con las características mínimas solicitadas?

- ☒ Utilizar 4 tipos de datos: Entero, carácter, booleano, real.
- ☒ Definir arreglos de N dimensiones sobre los tipos de datos
- ☒ Incluir una estructura de datos heterogéneos (struct en C)
- ☒ Incluir una estructura de decisión (if-else, switch)
- ☒ Incluir al menos una estructura repetitiva (do-while, while, for)
- ☒ Definir estatuto compuesto ({} de C, obegin-end de Pascal)
- ☒ Instrucciones de entrada y salida (leer, escribir, y escribir cambiando de línea)
- ☒ Funciones con paso de parámetros
- ☒ Operadores aritméticos (+, -, *, /, %(residuo), **(potencia))
- ☒ Operadores relacionales (<, <=, >, >=, ==, <>)
- ☒ Operadores lógicos (Y, Or, Not)
- ☒ Asignación

II. Análisis léxico

¿Cuáles son sus tokens?

Tabla 1. Tokens y expresiones regulares

Token	Expresión regular
num	[0-9]
PCOMA	;
OPLOG	“and” “or”
NOT	“not”
TIPO	message code mission-status
ID	{car}({car} {digito})*
LLA	{
LLC	}

PA	(
PC)
CORA	[
CORC]
DIV	,
ASIG	=
OPAR	$(\backslash+)(\backslash+)?(\backslash-)(\backslash-)?(\backslash*)(\backslash*)? \backslash /\%$
OPCOMP	$(\text{"="} \text{"<="} \text{">="} \text{"<"}$
entero	NUM+
real	NUM+ (.NUM+(E(+ -)?NUM+) .NUM+ E(+ -)?NUM+)
FINPROGRA	"onesmallstep"
CAR	[a-zA-Z]
cadena	\"(.*?)\"

¿Cómo serán los tipos que se aceptarán?

Tabla 2. Tipos de datos:

Tipo de dato	¿Cómo se define?	Palabra reservada	Ejemplo
<i>entero</i>	Cualquier número entero	code	code ex = 2;
<i>carácter</i>	Cualquier carácter o grupo de caracteres, uno o más letras que se distinguen con el uso de comillas simples o dobles	message	message ex = 'z';
<i>real</i>	Cualquier número decimal, un número entero también puede definirse como	space-coordinates	space-coordinates ex = 3.45;

	número real. Es posible utilizar notación científica con la letra E.		
<i>boolean</i>	Puede ser verdadero o falso, definido por las palabras reservadas.	mission-status	mission-status a =true;
<i>arreglos</i>	Un arreglo se define de manera similar al lenguaje python, utilizando un paréntesis cuadrangular [] y comillas divisorias ([,,,])	flight-journal	flight-journal test = [2,3,4,5]
<i>Datos heterogéneos (struct en C)</i>	Se define de la misma forma que en C, pero con la palabra reservada ufo.	ufo	ufo andromeda { message first = 'hello!'; code second = 3456; };

¿Cuáles serán las palabras reservadas del lenguaje?

Tabla 2. Palabras reservadas.

Palabra reservada	Equivalencia
ignition	if
abort	else
countdown	while
return	return
flight-journal	arreglo
ufo	struct

Tabla 3. Operadores

Descripción	Operador
Para asignar una variable a un valor.	= (Asignación)

Para comparar dos números o variables si son iguales dentro de un condicional	==
Menor que	<
Mayor que	>
Menor igual	<=
Mayor igual	>=
Suma dos valores	+
Resta de dos valores	-
Multiplicación de dos valores	*
División de dos valores	/
Módulo de dos valores	%
Un valor a la potencia de otro	**
Permite delimitar arreglos, estructuras, funciones, término de ciclos y condicionales	[] { }
Operador lógico “y”, será verdad solo si ambos son verdad	And (“and”)
Operador lógico “o”, será verdad si uno es verdad	Or (“or”)
Permite cambiar un valor de verdadero a falso o de falso a verdadero	Not (“not”)

III. Análisis Sintáctico

Definición formal de la gramática

4 elementos $G = (V, T, S, P)$

- V es el conjunto de variables, aquellos elementos que deben ser desarrollados
- T conjunto de terminales, este conjunto contendrá los tokens que debe reconocer el analizador léxico
- S es el símbolo inicial, y se refiere a la variable con la que se inicia todo el proceso.

➤ P es el conjunto de reglas de producción o de sobreescritura

V (Sintáctico y Semántico):

V = {S, MAIN, LINE, FUNCION, DECLARACIONES, FUNCION, LLAMADA, RETURN, DECLARACION, ARREGLO, ESTRUCTURA, ATRIBUTOS, ASIGNACION, OPERACION, CONDICIONAL, CICLOS, CONDICIONAL, COMPARACIONES, IO, VALOR}

T (Léxico):

T = {FINPROGRA, PYC, DIV, PA, PC, LLA, LLC, ID, TIPO, RETURN, CORA, CORC, ASIG, ID, OPCOMP, OPAR, OPLOG, num, bool, real, entero, cadena, *return, ignition, abort, countdown, out, over* }

S (Símbolo inicial):

S

P (conjunto de reglas de producción o de sobreescritura)

S → MAIN FINPROGRA

MAIN → LINE | MAIN LINE

LINE → (DECLARACION | ASIGNACION | LLAMADA | OPERACION | CONDICIONAL | CICLOS | IO | FUNCION | RETURN) PYC

DECLARACIONES → DECLARACIONES DIV DECLARACION | DECLARACION

FUNCION → TIPO ID PA DECLARACIONES PC LLA MAIN LLC

LLAMADA → ID PA VALOR PC

RETURN → *return* | *return* ID | *return* VALOR | *return* OPERACION

DECLARACION → TIPO ID | ARREGLO | ESTRUCTURA

ARREGLO → *flight-journal* ID CORA VALOR CORC

ESTRUCTURA → *ufo* PA ATRIBUTOS PC

ATRIBUTOS → DECLARACION ATRIBUTOS | DECLARACION

ASIGNACION → DECLARACION ASIG VALOR | ID ASIG VALOR | ID CORA num
CORC ASIG VALOR

OPERACION → ID OPAR ID | ID OPAR VALOR | VALOR OPAR VALOR | VALOR OPAR
ID

CONDICIONAL → *ignition* PA COMPARACIONES PC LLA MAIN LLC | *ignition* PA
COMPARACIONES PC LLA MAIN LLC *abort* LLA MAIN LLC

CICLOS → *countdown* PA COMPARACIONES PC LLA MAIN LLC

COMPARACIONES → ID OPCOMP ID | ID OPCOMP VALOR | VALOR OPCOMP VALOR |
VALOR OPCOMP ID | COMPARACIONES OPLOG COMPARACIONES

IO → *over* PA VALOR PC | ID ASIG *out* PA VALOR PC | TIPO ID ASIG *out* PA
VALOR PC

VALOR → num | cadena | bool | real | entero | OPERACION | LLAMADA

IV. Manual del usuario

¿Cómo usar el compilador? [link a github](#)

Archivos que componen el compilador

- **Lexico.l:** es el analizador léxico, nos permite saber si todos los tokens del programa son válidos
- **sintactico.y:** nos permite detectar problemas de sintaxis en el programa.

Comandos para correr el programa en terminal

1. Para correr el bison (extensión .y):

```
bison -vd sintactico.y
```

2. Para correr el flex (extensión .l):

```
flex lexico.l
```

```
gcc lex.yy.c
```

```
.\a.exe
```

Para correr el programa completo:

```
gcc lex.yy.c sintactico.tab.c -o compilado
```

```
.\compilado.exe
```

Comandos para correr el programa desde un archivo .txt:

Estando en la carpeta del código una vez descargado, correr los comandos en el siguiente orden:

```
cd compilador  
bison -vd sintactico.y  
flex ./Lexico.l  
cd ..  
gcc main.c ./compilador/*.c  
./a.exe
```

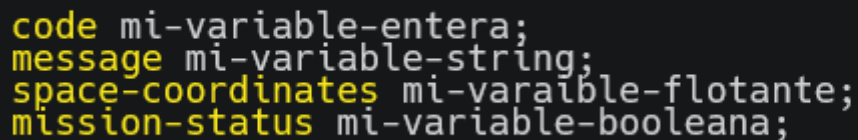
¿Cómo programar Artemis 22?

1. ¿Cómo declarar una variable?

Una variable se define escribiendo su TIPO seguido de un ID para guardarla y poder utilizarla en otras partes del código.

- Variable entera: se usa la palabra reservada “code”
- Variable string: se usa la palabra reservada “message”
- Variable flotante: se usa la palabra reservada “space-coordinates”
- Variable booleana: se usa la palabra reservada “mission-status”

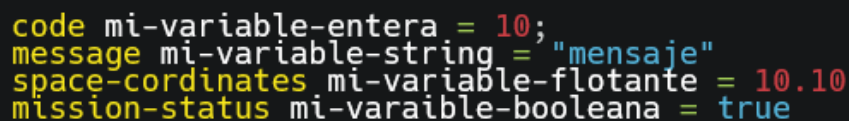
Ejemplo:



```
code mi-variable-entera;  
message mi-variable-string;  
space-coordinates mi-variable-flotante;  
mission-status mi-variable-booleana;
```

2. ¿Cómo asignar una variable?

Para asignarle un VALOR a una variable ya declarada se utiliza el operador = y el VALOR. Ejemplo:



```
code mi-variable-entera = 10;  
message mi-variable-string = "mensaje"  
space-coordinates mi-variable-flotante = 10.10  
mission-status mi-variable-booleana = true
```

3. ¿Cómo definir una función (con return)?

Una función permite realizar ciertas operaciones de forma continua, y llamarse desde cualquier lugar del código con acceso a la misma. Se define con un TIPO (qué regresará la función), ID (nombre para llamar a la función), DECLARACIONES (elementos que se utilizarán dentro de la función) y MAIN (operaciones que realiza la función). Ejemplo:

```
code mifuncion(code param1, mission-status param2)
{
    MAIN
}
```

4. ¿Cómo hacer un ciclo?

Un ciclo se define con: palabra reservada “countdown” y una condición de entrada entre paréntesis (COMPARACIONES). Siempre y cuando se cumpla la condición de entrada se realizarán las operaciones del MAIN. Ejemplo:

```
countdown (n < 0)
{
    MAIN
}
```

5. ¿Cómo hacer un condicional (if/else)?

Una condición se define con la palabra reservada “ignition”, a partir de la cual se utilizan paréntesis donde se ingresa la condición. Si la condición es verdadera se llevan a cabo las operaciones que se encuentren en el MAIN. En el caso contrario, se utiliza la palabra “abort” y se realiza lo que se encuentra entre corchetes. Ejemplo:

```
ignition (n < 0) {
    MAIN
} abort {
    MAIN
}
```

6. ¿Cómo hacer un arreglo?

Un arreglo se define con la palabra reservada “flight-journal”, un ID para poder llamarlo durante el resto del código y un [VALOR] entre corchetes que indica su tamaño. Ejemplo:

```
flight-journal miArreglo = [1,2,3,4,5]
```

7. ¿Cómo hacer una estructura?

Una estructura se define con: palabra reservada “ufo” seguida del nombre de la estructura (ID) y llaves. Dentro de las llaves se declaran atributos, que pueden ser 0 a N y de cualquiera de los tipos anteriormente mencionados. Ejemplo:

```
ufo miestructura  
{  
    message first = 'hello!';  
    code second = 3456;  
};
```

8. ¿Cómo imprimir y leer?

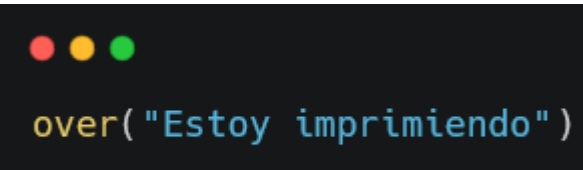
Es posible leer de 2 formas:

1. ID = out(VALOR)
2. TIPO ID = out(VALOR).

En ambos ID guardará aquello que ingrese el usuario, y el valor se imprimirá en la consola antes de recibir el input. Ejemplos:

```
mivariable = out("Ingresa el valor");  
code mivariable = out("Ingresa el valor")
```

Por otro lado, para poder imprimir se debe usar la palabra reservada “over”, seguido de paréntesis, aquello que se quiere imprimir debe estar delimitado por comillas. Ejemplos:



```
over("Estoy imprimiendo")
```