**Sintaxis y Semántica de los Lenguajes**

**TPN3 - Flex y Bison**

Integrantes - Grupo 7

| ALUMNO | Legajo |
| --- | --- |
| Giovani Quispe | 159978-1 |
| Guido Biotti | 171.434-0 |
| Santiago Hamamura | 203.717-8 |

Profesora Ing. Roxana Leituz

Curso K2006 - Sábados Turno mañana

2023

Enunciado

Hacer un programa utilizando flex y bison que realice análisis léxico, sintáctico y semántico de micro. Deben personalizar los errores e implementar al menos 2 rutinas semánticas.

Lineamientos de entrega:

• Carátula con todos los integrantes

• Archivos con el código fuente (\*.l y \*.y)

• Ejecutable

• Documento con un detalle de **como se resolvió el problema**, **manual de usuario** y **pantallas que muestren que el programa funciona**

• Archivos anexos: se deben incluir los archivos que se utilicen en las pruebas

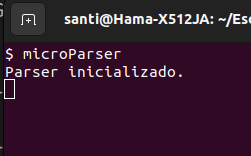
• Grupo: el mismo de todo el año

• Caso de copia: desaprueban la materia y deberán recuperar en marzo perdiendo por completo la posibilidad de promoción.

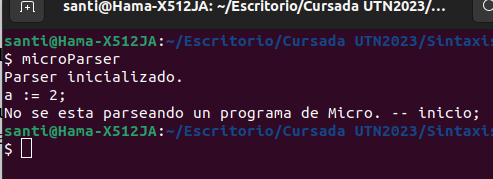
• El trabajo para darse por cumplido debe entregarse funcionando con todos los ítems desarrollados.

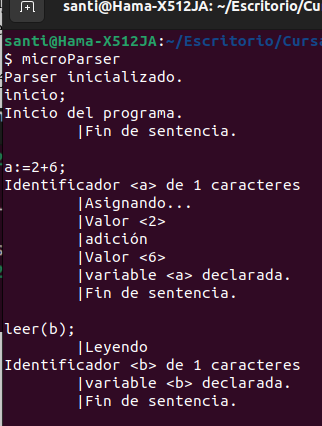
• Se entrega por campus una copia por grupo.

Fecha de entrega: 17/11

Manual de Usuario

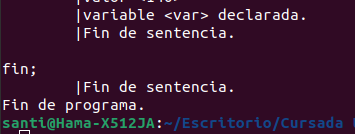
En la entrega se incluye un binario ejecutable llamado “microParser” diseñado para interactuar por terminal con el usuario. Se imprimirá el mensaje “Parser inicializado”, tras lo cual podrá comenzar a introducir sentencias del lenguaje Micro, que se procesarán según las especificaciones en este documento.

La primera instrucción debe ser “inicio;”. Caso contrario el programa no es reconocido como un programa de Micro válido y se abortará tras el mensaje “No se está parseando un programa de Micro.”. Luego de inicio; se procede al reconocimiento de cualquier sentencia propia del lenguaje Micro. 



Se abortará por errores semánticos en caso de escribir() o realizar una asignación con una variable que no ha sido inicializada por asignación o lectura, o como ya se ha mostrado, en caso de introducir una sentencia fuera de un bloque inicio; - fin;.

Si se introdujera la sentencia fin;, el proceso terminará con código EXIT\_SUCCESS.



Resolución

Se tomó como referencia de desarrollo del presente compilador de Macro a la especificación del capítulo 2 del libro de Charles Fischer, “Crafting a Compiler with C” (1991), proporcionada por la cátedra de Sintaxis y Semántica de los Lenguajes de la FRBA - UTN. Se enumeran las especificaciones:

- El único tipo de dato es entero.

- Todos los identificadores son declarados implícitamente y con una longitud máxima de 32 caracteres.

- Los identificadores deben comenzar con una letra y están compuestos de letras y dígitos.

- Las constantes son secuencias de dígitos (números enteros).

Hay dos tipos de sentencias:

**Asignación**

– ID := Expresión;

Expresión es infija y se construye con identificadores, constantes y los operadores + y –; los paréntesis están permitidos.

**Entrada/Salida**

– leer (lista de IDs); escribir (lista de Expresiones);

– Cada sentencia termina con un "punto y coma" (;). El cuerpo de un programa está delimitado por inicio y fin.

– inicio, fin, leer y escribir son palabras reservadas y deben escribirse en minúscula.

En esta implementación **el scanner permite los espacios en blanco “ ”** entre tokens, ignorándolos al detectarlos.

**NO se permiten las asignaciones compuestas** (a := b := 2;).

Si se realiza una rutina sintáctica sobre los argumentos de leer() y escribir(), más NO se realiza una rutina semántica; **si se invoca a las funciones sin argumentos, se terminará el programa por error será sintáctico**, sin procesamiento semántico. NO se implementó un análisis semántico de funciones genéricas que considere cantidades de parámetros.

Rutinas Semánticas

La implementación de una tabla de símbolos es un vector simple que almacena los nombres de las variables declaradas. Las palabras reservadas leer y escribir no están en esa tabla por simplificación, aunque lo adecuado sea que existan allí. En cambio, la funcion tipoIdentificador() devuelve los respectivos códigos de ambas palabras reservadas al scanner, para que este luego devuelva al parser el token correcto.

int tipoIdentificador(char\* constante)

{

int tipo = T\_VARIABLE;

if (strcmp(constante, "leer")==0) {

tipo = T\_ENTRADA;

}

else if (strcmp(constante, "escribir")==0) {

tipo = T\_SALIDA;

}

return tipo;

}

En caso de que se utilice un identificador que no corresponde a leer/escribir, el procedimiento dependerá del contexto en que se utilice al identificador.

Si se utiliza al identificador en una expresión de asignación o en escribir, se supone que el identificador representa a una variable declarada y se procede a buscarlo en la tabla de símbolos con la rutina verificarDeclaracionPrevia().

void verificarDeclaracionPrevia(char\* var)

{

if (estaDeclarada(var)){

return;

}

printf("Se utilizo una variable no declarada.\n");

exit(1);

}

Si el identificador se utilizase en el LHS de una asignación, o en leer(), se está realizando una declaración (implícita) y por lo tanto se procede a declarar (si no está ya declarada) la variable, almacenando el identificador en la tabla de símbolos con la rutina declararVariable().

void declararVariable(char\* var)

{

if (estaDeclarada(var)){

return;

}

symbolEntry new\_entry = (char\*)malloc(strlen(var)+1);

strcpy(new\_entry, var);

symbolTable[entryCount]=new\_entry;

entryCount++;

printf("\t|variable <%s> declarada.\n", var);

}

No se almacenan estructuras de inicialización (el valor asociado a una asignación o lectura) junto con la variable, sino únicamente al identificador para saber si ya existe o no.

El Scanner

El módulo flex reconoce los tokens ":=", ";", "inicio", "fin", "<espacio>" ,"<coma>", "(" , ")", "+", "-", “<constEntera>” y “identificador”.

La expresión regular de los identificadores es

{LETRA}({LETRA}|{DIGITO})\*

DIGITO [0-9]

LETRA [a-zA-Z]

Y al detectarse uno, se utiliza la funcion tipoIdentificador() que consulta al parser por el tipo de identificador leído, distinguiendo entre los reservados leer/escribir y cualquier otro que sirva de nombre de variable.