UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ingeniería en sistemas de información.

Sintaxis y semántica de los lenguajes.

TRABAJO PRÁCTICO TEÓRICO.

ÁREA TEMÁTICA: Flex y Bison.

GRUPO N° 22

Apellido y Nombre	N° de Legajo
Gaetan, María Luz	163627-3
Laino, Ramiro Angel	175883-4
Lamothe, Genty Clarke	167828-0
Mendiolar Colombo, Nahuel Nehuen	169188-0
Rolando, Sebastian	176.587-5

CURSO: K2055
DOCENTE A CARGO: Roxana Leituz

FECHA DE VENCIMIENTO: 17/11/2023
FECHA DE PRESENTACIÓN: 17/11/2023
FECHA DE DEVOLUCIÓN: __/__/____

CALIFICACIÓN: _____ FIRMA DOCENTE: _____

Índice

Link GIT https://github.com/MendioX/SSL-GRUPO/tree/main/TP3

Consigna:

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.

Análisis léxico:

Para realizar el análisis léxico se utiliza la herramienta Flex, la cual va a recibir de entrada un flujo de caracteres y va a devolver Tokens para que reciba el analizador sintáctico.

Los Token del lenguaje Micro que se deben implementar son las palabras reservas, los identificadores, las constantes, los operadores, la asignación y los caracteres de puntuación.

Las especificaciones de Flex siguen el siguiente formato

definiciones

%%

reglas

%%

código del Usuario

Se incluye además mediante #include “y.tab.h” el archivo generado por la compilación del analizador sintáctico de Bison, que contiene el nombre de los Tokens a utilizar.

En las reglas definimos los Tokens detectados por el analizador léxico para poder suministrarles al analizador sintáctico a través de “yylval”. Además se declara un mensaje de error léxico en caso de que la cadena ingresada no pertenezca a ningún Token.

```
%{
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

typedef struct ListaId {
        char id[20];
        struct ListaId* next;
    } ListaId;

#include "y.tab.h"
%}

%option noyywrap
identificador   [a-zA-Z][a-zA-Z0-9]{0,31}
numbers         [0-9]*
simbolos        [+\-*/]
reservada       leer|escribir
inicio          inicio
fin             fin

%%
{inicio}        {yylval.inicio = yytext; return(INICIO);}
{fin}           {yylval.fin = yytext; return(FIN);}
{reservada}     {sscanf(yytext, "%s",yylval.reservada); return(RESERVADA);}
{identificador} {sscanf(yytext, "%s",yylval.cad); return(IDENTIFICADOR);}
{numbers}       {yylval.number = atoi(yytext); return (ENTERO);}
{simbolos}      {yylval.simbolos = atoi(yytext); return (SIMBOLOS);}
";"             {return (PUNTOCOMA);}
","             {yylval.coma = yytext; return(COMA);}
"("             {return (PARENTESISOPEN);}
")"             {return (PARENTESISCLOSE);}
":="            {return (ASIGNACION);}
[\n]            {}
[\r]            {}
[\t]            {}
.               {printf("Error lexico.\n"); printf("Mystery character: %s\n", yytext); return(OTHER);}
```

Análisis sintáctico:

Este análisis es el que ejecuta Bison. La estructura del fichero de Bison es la siguiente.

Secciòn de definiciones

%{

/*delimitadores de código de C*/

%}

%%

Sección de reglas

%%

Sección de funciones del usuario

En la sección de definiciones %token define los tokens del lenguaje a usar, que son los que después el analizador sintáctico va a compilar en un archivo “y.tab.h” para que lo pueda usar Flex.

%type lo que hace es definir el tipo de los No Terminales.

```
74
75  %token INICIO FIN RESERVADA IDENTIFICADOR ENTERO SIMBOLOS PUNTOCOMA COMA PARENTESISOPEN PARENTESISCLOSE ASIGNACION OTHER
76
77  %type <cad> IDENTIFICADOR lista_vars
78  %type <number> ENTERO
79  %type <reservada> RESERVADA
80  %type <inicio> INICIO
81  %type <fin> FIN
82  %type <simbolos> SIMBOLOS
83
```

En la sección de reglas definimos las gramáticas del lenguaje Micro:

```
103  %%
104  prog: INICIO codigo FIN;
105
106  codigo: /* empty */
107      | codigo stmt
108      ;
109
110  stmt: RESERVADA PARENTESISOPEN lista_vars PARENTESISCLOSE PUNTOCOMA
111      {
112          processReservada($1, $3);
113
114      }
115      | IDENTIFICADOR ASIGNACION expr PUNTOCOMA
116      {
117          stackId($1);
118      }
119      ;
120
121  lista_vars: lista_vars COMA IDENTIFICADOR
122      {
123          $$ = construir_lista($3, $1);
124      }
125      |  IDENTIFICADOR
126      {
127         $$ = construir_lista($1, NULL);
128      }
129      ;
130
131
132  expr: ENTERO
133      | IDENTIFICADOR
134      | expr SIMBOLOS expr
135      ;
136
137  %%
```

En las funciones declaradas por el usuario, podemos definir el main de la siguiente manera:

```
int main(int argc, char **argv){
    printf("      Comienzo de analis... \n");
    printf("---------------------------------------\n");

    yyparse();

    getchar();
    printf("---------------------------------------\n");

     printf("       Fin analisis... \n");
    return 0;
}
```

En el cual hacemos un llamado a la función yyparse().

Análisis Semántico:

Para las rutinas semánticas definimos una lista para poder guardar los tokens:

```
9       /*Estructura para un nodo en la tabla de símbolos*/
10
11    typedef struct SymbolTableNode {
12        char id[20];
13        struct SymbolTableNode* next;
14    } SymbolTableNode;
```

```
21
22     /*Tabla de símbolos*/
23     SymbolTableNode* symbolTable = NULL;
24
```

Se definieron estas dos rutinas semánticas, que se procesan en la función processReservada.

processReservada: definimos si estamos ante una situaciòn donde tenemos que leer o escribir.

De ahí definimos si reservada es LEER, guardamos en una pila todas las variables de la expresiòn

Si la palabra reservada es ESCRIBIR, verificamos que las variables a escribir hayan sido declaradas previamente en el código, es decir si existen en la pila.

```
176
177     void processReservada(char* reservada, ListaId* listaVars) {
178         printf("Procesando palabra reservada: %s\n", reservada);
179         printf("Procesando palabra lista: %s\n", listaVars);
180         if (strcmp(reservada, "leer") == 0) {
181             printf("\nInstruccion LEER\n\n");
182             lista_vars_stack(listaVars);
183             void printSymbolTable();
184         } else if (strcmp(reservada, "escribir") == 0) {
185             printf("\nInstruccion ESCRIBIR\n");
186              lista_vars_check(listaVars);
187
188         }
189
190
191     }
```

construir_lista :

Generamos la lista de variables que se reciben dentro del paréntesis ej leer(a,b,c); y la devolvemos como parámetro **$$** de la iteración completa de la expresión lista_vars.

```
lista_vars: lista_vars COMA IDENTIFICADOR
    {
        $$ = construir_lista($3, $1);
    }
    | IDENTIFICADOR
    {
    $$ = construir_lista($1, NULL);
    }
    ;
```

Esto se resuelve de esta manera par que la función **processReservada** pueda recibir la cadena completa de variables.

```
59
60  struct ListaId* construir_lista(char* id, struct ListaId* lista_id) {
61      struct ListaId* nueva_lista = (struct ListaId*)malloc(sizeof(struct ListaId));
62      if (nueva_lista == NULL) {
63          fprintf(stderr, "Error: No se pudo asignar memoria para la lista de identificadores.\n");
64          exit(EXIT_FAILURE);
65      }
66      strcpy(nueva_lista->id, id);
67      nueva_lista->next = lista_id;
68      return nueva_lista;
69  }
70
```

En nuestro caso utilizamos el siguiente código para poner a prueba el programa:

inicio

leer(a,b,c);

cc:=a+b;

escribir(cc,a);

fin

Ejecución:

Para poder ejecutar el programa utilizamos **script.bat** que contiene las directivas de compilación y ejecuta el programa consumiendo el archivo de texto con el código, en vez de hacer la entrada manual de los datos:

```
@echo off
cls
echo "<inicio bison y flex>"
flex -l lexico.l
bison -yd sintaxis.y
gcc y.tab.c lex.yy.c -lfl -o output
output.exe < testingCode.txt
echo "<fin>"
```

Al final obtenemos la tabla de símbolos impresa por pantalla, junto con algunos printf a modo de debug del programa.

```
"<inicio bison y flex>"
     Comienzo de analis...
---------------------------------------
Procesando palabra reservada: leer
Procesando palabra lista: c

Instruccion LEER

Procesando palabra reservada: escribir
Procesando palabra lista: a

Instruccion ESCRIBIR
---------------------------------------
       Fin analisis...
       ID encontrados
Tabla de simbolos:
cc
a
b
c
"<fin>"

C:\Users\Monster\Desktop\SSL-GRUPO\TP3>
```