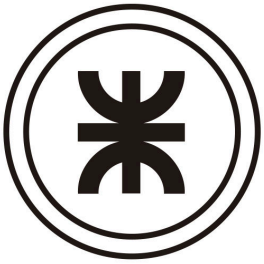

UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Sintaxis y Semántica de los lenguajes

Trabajo Práctico N°3
Flex y Bison

GRUPO 7

<Zoe Iael Mancini>	<206.464-9>	<zmancini@frba.utn.edu.ar>	<zoemancini>
<Sofia Zalazar>	<204.178-9>	<szalazar@utn.frba.utn>	<SOFIAZALAZAR>
<Román Brezman>	<203.432-3>	<rbrezman@frba.utn.edu.ar>	<rabrezman>
<Mariano Bruno>	<138.794-7>	<marbruno@frba.utn.edu.ar>	<marianonahuelbruno>
<Guido Moreyra >	<209.802-7>	<gmoreyra@frba.utn.edu.ar	<GuidoMoreyra>

Docente: *Roxana Leituz* ***Curso:*** *K2055*

Índice

Objetivo

Hacer un programa utilizando Flex y Bison que realice análisis léxico, sintáctico y semántico de micro. Personalizar los errores e implementar al menos 2 rutinas semánticas.

Link al repositorio de GitHub

En el siguiente repositorio se encuentran subidos todos los archivos que se utilizaron para llevar a cabo el presente trabajo: https://github.com/rabrezman/tp-analizador-grupo7

Compilación de los archivos de Flex y Bison

Luego de generar nuestros archivos **scanner.l** y **parser.y**, que pueden ver en el repositorio adjunto más arriba, procedimos a hacer la compilación de los códigos para generar el ejecutable de nuestro compilador.

```
zoema@ASUS-ZOEMANCINI MINGW64 ~/Desktop/ssl/tp-analizador-grupo7/tp3 (main)
$ bison -yd parser.y
```

Esto nos generó los archivos **y.tab.c** e **y.tab.h** que luego se referencia en el archivo **scanner.l**.

```
zoema@ASUS-ZOEMANCINI MINGW64 ~/Desktop/ssl/tp-analizador-grupo7/tp3 (main)
$ flex scanner.l
```

Esto nos genera el archivo **lex.yy.c**, que es el que usaremos para generar el ejecutable junto con el **y.tab.c** generado por bison.

```
zoema@ASUS-ZOEMANCINI MINGW64 ~/Desktop/ssl/tp-analizador-grupo7/tp3 (main)
$ gcc y.tab.c lex.yy.c -o compilador
```

Por último, creamos el ejecutable **compilador.exe**.

Rutinas semánticas implementadas

Tal como la consigna lo pedía se implementaron 2 rutinas semánticas, el funcionamiento de estas mismas se puede ver en algunos apartados de la sección **Ejemplos**. Las rutinas implementadas fueron las siguientes:

1. Rutina para detección de identificadores no declarados: esta rutina se encarga de agregar los identificadores a la tabla de símbolos (una emulación que hicimos de la misma, simplificada con propósitos académicos, como un array de 30x32) en caso de asignación o lectura. Si el identificador se utilizó en la función escribir o se hubiera utilizado para alguna operación, entonces verifica que el mismo esté debidamente declarado y en caso de no estarlo informa el error correspondiente.
   A continuación se muestran algunas secciones del código que se usó para implementar esta rutina, sin embargo el código está completo en el repositorio que se dejó adjunto en la sección **Link al repositorio de GitHub**.

```
primaria: IDENTIFICADOR {
existeIdentificador = buscarIdentifDeclarado();
if(!existeIdentificador){yyerror("error semantico, no se
encuentra el identificador declarado"); return 0;}}
| CONSTANTE
| PARENTIZQUIERDO expresion PARENTDERECHO
;

int buscarIdentifDeclarado(void){
   int idEncontrado = 0;
   for(int j=0;j<i;j++){
      if(strcmp(idtifDeclarados[j], yytext)==0){
         idEncontrado = 1;
      }
   }
  return idEncontrado;
}
```

2. Rutina para detección de identificadores con más de 32 caracteres: dado que el lenguaje Micro tiene como restricción que los identificadores no pueden superar los 32 caracteres de largo, decidimos implementar esta misma como una rutina semántica que lo verifique. En caso de que el identificador indicado supere los 32 caracteres se informará un error acorde por pantalla.
   A continuación se muestran algunas secciones del código que se usó para implementar esta rutina, sin embargo el código está completo en el repositorio que se dejó adjunto en la sección **Link al repositorio de GitHub**.

Se puede ver en el código que cuando se recibe un identificador se verifica su longitud y si esta está OK recién ahí se procede a verificar si se debe agregar o no a la tabla de símbolos.

```
identificador: IDENTIFICADOR {
error = validarLongitud();
if(error){return 0;}
existeIdentificador = buscarIdentifDeclarado();
if(!existeIdentificador){agregarIdentificador();}
};

int validarLongitud(void){
   if(yyleng > 32){
   yyerror("error semantico, el identificador no puede tener mas
de 32 caracteres\n");
   return 1;}
   return 0;
}

void agregarIdentificador(void){
   strcpy(idtifDeclarados[i], yytext);
   i++;
}
```

Ejemplos

Código Micro correcto

```
zoema@ASUS-ZOEMANCINI MINGW64 ~/Desktop/ssl/tp-analizador-grupo7/tp3 (main)
$ ./compilador
inicio
a:=3;
b:=6;
leer(c);
escribir(a+b,c);
fin
```

Código Micro con un error léxico

```
zoema@ASUS-ZOEMANCINI MINGW64 ~/Desktop/ssl/tp-analizador-grupo7/tp3 (main)
$ ./compilador
inicio
a:=3*2;
Ocurrio: error lexico, el caracter utilizado no pertenece al lenguaje
Ocurrio: syntax error
```

Código Micro con un error sintáctico

```
zoema@ASUS-ZOEMANCINI MINGW64 ~/Desktop/ssl/tp-analizador-grupo7/tp3 (main)
$ ./compilador
inicio
a:=3+((2-4);
Ocurrio: syntax error
```

Código Micro con un identificador no declarado

```
zoema@ASUS-ZOEMANCINI MINGW64 ~/Desktop/ssl/tp-analizador-grupo7/tp3 (main)
$ ./compilador
inicio
a:=c+4;
Ocurrio: error semantico, no se encuentra el identificador declarado
```

Código Micro con un identificador con más de 32 caracteres

```
zoema@ASUS-ZOEMANCINI MINGW64 ~/Desktop/ssl/tp-analizador-grupo7/tp3 (main)
$ ./compilador
inicio
avercuantoscaracteressonestosssssssssssssssssssssnoseperoseguroquesonmasdetrentaydos:=4;
Ocurrio: error semantico, el identificador no puede tener mas de 32 caracteres
```