Trabajo Práctico Nº 3 – Escáner

Hecho con flex

1. **Objetivos**
   a. Armar un escáner para lenguaje mini.
   b. Profundizar el conocimiento de la herramienta flex.
2. **Temas**
   a. Reconocedor léxico.
   b. Generadores de código.
3. **Tareas**
   a. Se pide armar un escáner para el lenguaje mini que es descripto en la documentación entregada en el TP anterior. Esta vez utilizaremos la herramienta flex para construir el scanner.
   El escáner debe trabajar contra stdin y stdout. Luego para probar podremos redireccionar el archivo `entrada.txt` al flujo stdin. Se provee el archivo `entrada.txt` para que puedan probar. Debería dar un salida similar a la mostrada en el archivo `salida.txt`
   Comentarios:
      1. El archivo `salidaOpt.txt` muestra una alternativa, si se quiere mantener los mensajes de error sobre el operador asignación como en el TP2, pero es optativo.
      2. Mostrar el token Fin de texto también es optativo.
   b. Programar usando los siguientes fuentes
      1. `main.c` llama al escáner hasta que este devuelve el token `FDT`. Con cada devolución del escáner informa en `stdout` que token fue reconocido y el lexema correspondiente. Notar que los caracteres de puntuación y los operadores solo mostramos el carácter, sin ponerle nombre.
      2. `scanner.h` tiene el enumerado de tokens. Notar que como usamos flex queremos reconocer por separado cada palabra reservada, operador o carácter de puntuación. Los operadores y caracteres de puntuación no necesitan un token, pueden devolver el mismísimo carácter como token.
      3. `scanner.l` tiene la definición flex del mismo. Ponga las opciones para generar scanner.c y scanner.h como archivos.
   c. Puede usar un makefile si lo considera útil, pero no es requerido. En caso de usarlo, suponiendo que usa vscode con msys2 en windows, al compilar, para poder usar la biblioteca de flex, deberá indicar en que directorio está: `-L/usr/lib/ -lfl`
   d. Los errores a reconocer son los indicados en la especifiación de mini, o sea:
      1. Cadena inválida: secuencia de caracteres no pertenecientes a nuestro alfabeto.
      2. Identificador inválido: comienza como identificador, pero puede entremezclar caracteres no pertenecientes a nuestro alfabeto.
      3. Constante inválida: comienza como constante, pero puede luego entremezclar letras.
4. **Sugerencias**
   a. Utilice las directivas noinput y nounput para evitar warnings innecesarios.
   b. No hace falta reconocer EOF, cuando flex lo lea enviará un token cero, por tanto basta que al armar el enun de los tokens ubique como primero a FDT

c. El header generado por flex incluye la declaración de yytext, por tanto y dado que no tenemos necesidad de guardar los distintos lexemas (basta mostrarlos en el momento) puede usar yytext en main para mostrar el lexema

5. Productos

```
`24-002-xx                      //Repositorio del grupo
    |-- readme.md               // Carátula del grupo, ya hecha en TP1
    `-- TP3                     //Directorio para el TP2
         |-- readme.md          // Carátula del TP
         |-- main.c             // Inicio del programa
         |-- scanner.l          // Especificación flex del scanner
         |-- tokens.h           // Tokens a usar por los dos anteriores
```

6. Fechas de entrega

a. Última fecha para primera entrega: 08/10/2024

b. Última fecha para segunda entrega: 22/10/2024