UTN - FRBA
Sintaxis y Semántica de los Lenguajes
Trabajo Práctico n° 2 - Flex para reconocimiento de categorías léxicas de C

Enunciado

Implementar en lenguaje C un programa que permita reconocer desde un archivo *.i* (archivo *.c* preprocesado)[1] de entrada **todas las categorías léxicas de ANSI C (*C89/C90*).**

El mismo deberá producir como salida por defecto por pantalla (*stdout*) un *reporte*.

Para desarrollar dicho programa se deberá utilizar *Flex* para la generación del código C que implemente un analizador léxico (*scanner*) que cumpla con lo requerido.

Categorías léxicas de ANSI C a reconocer:

- **Constantes**
  - Enteras (decimales, octales, hexadecimales): con y sin sufijo.
  - Reales: con y sin sufijo.
  - Caracter: que a su vez pueden ser de tipo:
    - Simple (ej. *'a' , '0'*)
    - Con secuencia de escape
      - Simple (ej. *'\n' , '\t'*)
      - Octal (ej. *'\0' , '\07', '\007'*)
      - Hexadecimal (ej. *'\xF' , '\x0F'*)
- **Literales cadena**
- **Palabras reservadas** (agrupar por categoría sintáctica: ver abajo)
- **Identificadores**
- **Caracteres de puntuación/operadores**

El reporte deberá consistir en un conjunto de listados que deberá seguir el siguiente orden:

1. **Listado de identificadores** encontrados indicando la cantidad de veces que aparece cada uno de ellos. El listado debe estar ordenado alfabéticamente.

2. **Listado de literales cadena** encontrados indicando la longitud de los mismos y ordenados por longitud ascendente. En caso de igual longitud se desempatan por orden de aparición.

3. **Listado de palabras reservadas** en el orden en el que han aparecido en el archivo, indicando el número de línea y columna de cada una de ellas. Se debe indicar todas las apariciones de cada palabra reservada y separar el listado por especificadores de clase de almacenamiento, especificadores de tipo, calificadores de tipo, struct o union, enumeración, etiqueta, selección, iteración, salto y unario.

4. **Listado de constantes** indicando según su tipo:
   - Para las constantes decimales indicar el valor de cada una y el total acumulado de sumar todas ellas (ver ejemplos para ver como se muestra).
   - Para las constantes hexadecimales indicar su valor entero decimal.
   - Para las constantes octales indicar su valor entero decimal.
   - Para las constantes reales indicar el valor de su mantisa y parte entera.
   - Para las constantes caracter, enumerarlas según orden de aparición.

[1] El archivo .c es el código fuente de entrada original, mientras que el archivo .i es el resultado del preprocesamiento de este código fuente.

Las constantes se deben listar en el orden de aparición y cada vez que se repitan por más que sea la misma.

5. **Listado de operadores/caracteres de puntuación** indicando cantidad de veces que aparecen. Listarlos ordenados según su orden de primera aparición.

6. **Listado de cadenas no reconocidas** indicando el número de línea y el de columna donde se encontraron.

Cuando no hay elemento de un listado se deberá imprimir en el reporte con un “-”. Ver ejemplos de test incluídos en la carpeta correspondiente al trabajo práctico.

Uso

./bin/tp2.exe <ruta archivo entrada> [parámetros adicionales opcionales]

La ruta al archivo de texto de entrada es el único parámetro **obligatorio**.
Adicionalmente, pueden agregar parámetros **opcionales** que consideren útiles o convenientes (ej. activar logs verbosos para facilitar el debug, etc.).

Ejemplo de aplicación

Archivo de entrada (archivo.i) de prueba (no hace falta que compile para que lo tome):

```
int main(void) {
    char *cadena = "Hola, Mundo\n";
    char *cadena_2 = "mundo";
    if cadena = "texto"
    @double@ sizeof
    return (20 * 10) + 07 + 0xFF + 017 + 07 + 1.5 * ('a' + '\0');
}
```

Ejecución:

./bin/tp2.exe archivo.i > salida.txt [2]

Archivo de salida (salida.txt) esperado:

```
* Listado de identificadores encontrados:
cadena: aparece 2 veces
cadena_2: aparece 1 vez
main: aparece 1 vez

* Listado de literales cadena encontrados:
"mundo": longitud 5
"texto": longitud 5
"Hola, Mundo\n": longitud 13

* Listado de palabras reservadas (clase de almacenamiento):
-

* Listado de palabras reservadas (especificadores de tipo):
int: linea 1, columna 1
void: linea 1, columna 10
char: linea 2, columna 5
char: linea 3, columna 5

* Listado de palabras reservadas (calificadores de tipo):
-

* Listado de palabras reservadas (struct o union):
-

* Listado de palabras reservadas (enumeracion):
-

* Listado de palabras reservadas (etiqueta):
-
```

[2] La extensión *.exe* sólo aplica para Windows. Éste comando redirige la salida por pantalla (stdout) del programa a un archivo (salida.txt). Si el archivo ya existe, lo sobreescribe; si no, lo crea.

```
* Listado de palabras reservadas (seleccion):
if: linea 4, columna 5

* Listado de palabras reservadas (iteracion):
-

* Listado de palabras reservadas (salto):
return: linea 6, columna 5

* Listado de palabras reservadas (unario):
sizeof: linea 5, columna 14

* Listado de constantes enteras decimales:
20: valor 20
10: valor 10
Total acumulado de sumar todas las constantes decimales: 30

* Listado de constantes enteras hexadecimales:
0xFF: valor entero decimal 255

* Listado de constantes enteras octales:
07: valor entero decimal 7
017: valor entero decimal 15
07: valor entero decimal 7

* Listado de constantes reales:
1.5: parte entera 1.000000, mantisa 0.500000

* Listado de constantes caracter enumerados:
1) 'a'
2) '\0'

* Listado de operadores/caracteres de puntuación:
(: aparece 3 veces
): aparece 3 veces
{: aparece 1 vez
*: aparece 4 veces
=: aparece 3 veces
;: aparece 3 veces
+: aparece 6 veces
}: aparece 1 vez

* Listado de cadenas no reconocidas:
@double@: linea 5, columna 5
```

Consideraciones

La entrega de este trabajo práctico **grupal** es **obligatoria**, su **fecha límite para consulta, entrega y revisión se encuentra indicada en el cronograma**. Luego de esa fecha, **no** se aceptarán más trabajos, y toda consulta referida quedará para la defensa final que al cierre de la cursada.

Las directivas del preprocesador y los comentarios no estarán presentes en el archivo *.i* que se recibe en esta etapa de la compilación puesto que se asume que todas ya han sido resueltos previamente en la etapa de preprocesamiento por el PREPROCESADOR. Es por eso que el analizador léxico correspondiente al COMPILADOR no los reconoce también – ya no hace falta en esta etapa.[3]

En el *Aula Virtual* tienen a disposición **resúmenes de sintaxis del lenguaje ANSI C (*C89/C90*)** *(Language Syntax Summary)*, los cuales incluyen una gramática léxica *(Lexical Grammar)* que puede ser de utilidad para escribir las reglas en *Flex*.

Basar el desarrollo del trabajo práctico en los ejemplos mostrados en clase que están disponibles **en el repositorio con ejemplos de Flex**.

Para su realización, **se debe llevar un registro del número de línea y el de columna actual** en la que el analizador léxico se encuentra leyendo del archivo de entrada.

Se recomienda **que el programa también tenga una o más opciones para poder depurarse**, para facilitar el seguimiento y prueba del mismo.[4] Por ejemplo:

- Imprimir (o no) qué cadenas el analizador léxico va reconociendo y no reconociendo, a qué categoría léxica pertenecen, y el número de línea y de columna donde fueron encontradas.

En este caso, se espera que la depuración pueda habilitarse/deshabilitarse a través de una o más variables (ej. *bool DEBUG*) y/o de definiciones de preprocesador (ej. *#define DEBUG*) con un valor binario (*1* ó *0*) según necesidad del grupo, y que por la salida estándar (*stdout*) únicamente se imprima la salida esperada (el reporte), de forma que los tests automáticos no fallen. Para imprimir los mensajes de depuración puede entonces utilizarse la salida estándar para errores (*stderr*), la cual también se muestra por pantalla en combinación con *stdout*, pero sin afectar a los tests automáticos.

Cuando se produce un error léxico, la recuperación del analizador léxico no debe ser inmediata, sino recién en el siguiente espacio, tabulación o salto de línea deberá continuar leyendo tokens. Asimismo deberá tomar la cadena **completa** como no reconocida. Ejemplos de cadenas no reconocidas: **ñ**andu , a**ñ**oraba , 07**8**12 , **@**double@ , 0x**X**FF , 123**a**sd

Para la generación del reporte es necesario **utilizar memoría dinámica** en la implementación del analizador léxico. No utilizar arreglos estáticos para implementarlo.

---

[3] En la práctica (*gcc* al menos), el PREPROCESADOR podría dejar algunas directivas de preprocesador sin expandir (por ejemplo, tras resolver un *#include*) que terminan quedando en el archivo *.i* que genera. Es por ello que los archivos *.i* generados por *gcc* no suelen servir como entrada para el TP, que es una versión simplificada y reducida de la realidad, la cual consiste en que el COMPILADOR también resuelve aquellas directivas que pudieran haber quedado para su etapa (e incluso tener que interpretar algún pseudocódigo que haya quedado tras resolverse una directiva *#line* por ejemplo). Como entrada para el TP, pueden utilizar los archivos *.i* de test y/o escribir los suyos.

[4] Si a *flex* se le agrega la opción *-d* al producir el archivo *.lex.yy.c*, al compilar y ejecutar el analizador léxico generado se depura por qué reglas va entrando el mismo.

El agrupamiento por tipo de las palabras reservadas es:

- Especificadores de clase de almacenamiento: *auto , register , static , extern , typedef*
- Especificadores de tipo: *void , char , short , int , long , float , double , signed , unsigned*
- Calificadores de tipo: *const , volatile*
- Struct o Union: *struct , union*
- Enumeración: *enum*
- Etiqueta: *case , default*
- Selección: *if , else , switch*
- Iteración: *do , while , for*
- Salto: *goto , continue , break , return*
- Unario: *sizeof*

Para escribir el código fuente del programa pueden utilizar cualquier estándar de C, como: ANSI C (*C89/C90)*, *C99*, *C11*, *C17/C18*, etc.[5]

Como compilador de C se utilizará *gcc* (***G**NU **C** **C**ompiler*). Sin embargo, se recomienda **no utilizar** extensiones del lenguaje C específicas del compilador (*GNU C Extensions*) que no formen parte del estándar de C utilizado.[6]

El programa debe poder compilarse independientemente del sistema operativo, utilizando únicamente bibliotecas estándar de C según necesidad.

En otras palabras, **el código C escrito debe ser portable**; para ello, no debe incluir bibliotecas específicas de ningún sistema (ej. *Windows*, *Linux*, *Mac OS*, etc.). Por ejemplo, bibliotecas definidas en los estándares *POSIX* (***P**ortable **O**perating **S**ystem **I**nterface/Uni**x***), ya que no son compatibles nativamente con Windows, entre otros sistemas.

Tener en particular consideración las bibliotecas:

- *stdio.h*: Funciones printf, fprintf, fopen, fclose.
- *stdlib.h*: Funciones atoi, atof, strtoul, strtof, strtod (además de malloc, realloc, free, exit).
- *string.h*: Funciones strlen, strcmp, strcasecmp, strcpy, strcat, strerror.
- *math.h*: Funciones modf, modff.
- *errno.h*: Códigos de error del sistema (variable de tipo int errno).
- *limits.h*: Límites de los tipos de datos (UINT_MAX, ULONG_MAX, etc.).

Pueden utilizar cualquier versión de Flex. [7] Como suelen ser retrocompatibles entre sí, solamente deberían consultar la documentación correspondiente a la versión más antigua de Flex de las que tengan instaladas entre los integrantes del equipo, para que se aseguren de utilizar características que estén presentes en todas ellas y que de esa forma todos puedan construir el proyecto. Se sugiere la documentación de Flex 2.5, la cual está presente en el Aula Virtual junto con la de las demás versiones.

---

[5] En *gcc* el estándar de C utilizado se especifica agregando una opción *-std* al compilar, entre las que están: *-std=c89*, *-std=c99*, *-std=c11*, *-std=c17*, etc. Para conocer todas las opciones *-std* (ergo todos los estándares) que soporta una versión de *gcc*, pueden consultar la documentación de GCC que corresponda a dicha versión. Con *gcc --version* pueden consultar la versión instalada.

Si no se indica una opción *-std* al compilar, *gcc* por defecto utiliza el último estándar de C que soporte pero incluyendo las extensiones del lenguaje propias del compilador.

[6] *gcc* puede alertar del uso de extensiones de C como warnings si se le agrega la opción *-Wpedantic* y/o como errores si se le agregan las opciones *-pedantic -pedantic-errors* al compilar.

[7] Con *flex --version* pueden consultar la versión instalada de Flex.

El **entorno de desarrollo a utilizar queda a elección de cada grupo** (*Eclipse*, *CodeBlocks*, CLion, ***Visual Studio Code***, etc.). Se recomienda un entorno que aplique resaltado de sintaxis para todos los archivos de especificación, que facilite la depuración de código C, así como que se integre con la consola de *Git* para poder realizar el trabajo de una forma más práctica.

Para compilar el programa, pueden utilizar herramientas como ***GNU Make*** con archivos de especificación correspondientes como ***makefiles***, ya sean de su propia autoría o los que les hayan sido provistos (en el *aula virtual* o en las plantillas de TPs en los repositorios grupales designados). En dicho caso, **esos archivos deberán formar parte de la entrega**.

Para el programa, sólo formarán parte de la entrega los **archivos de especificación** de Lex/Flex (*.l*)**, archivos fuente** de C (*.c*) **y/o archivos de cabecera** (*header files*) (*.h*) escritos por el equipo. No se considerarán archivos generados por Lex/Flex (ej. *.lex.yy.c*), ejecutables (ej. *.exe*) ni archivos intermedios de salida del compilador, como los archivos objeto (ej. *.o*), ensamblador (ej. *.s*) o preprocesados (ej. *.i*).

Completar el archivo *README.md* dentro del directorio del TP, el cual actúa como carátula del mismo, con los datos que allí consignan.

La **entrega será a través del repositorio** de GitHub designado para el equipo en la carpeta correspondiente al TP **mediante Pull Request (PR)** desde la rama creada para el TP hacia la rama principal (main), como se indica en el instructivo para entrega de trabajos prácticos.

Pueden **agregar también las aclaraciones que consideren necesarias mencionar** en relación con el trabajo práctico realizado **en dicho Pull Request (PR)**.

Las **consultas** podrán ser realizadas a través de Discord en el canal correspondiente a cada grupo, o bien, si es una consulta más general del trabajo referida al enunciado por el canal de consultas denominado TP-**#**, siendo **#** el número de TP.