INF-341: Compiladores
Tarea #1
"Le hacemos al léxico"

Anghelo Carvajal
201473062-4

26 de octubre de 2018

Para poder crear el analizador léxico, crearemos una gramática que acepte palabras. Para que una palabra sea aceptada como tal, esta debe contener al menos una letra (minúscula o mayúscula). Una palabra puede contener caracteres numéricos o guiones. Esta gramática quedaría formada de la siguiente forma:

$$LETRA = [a - zA - Z]$$
$$DEMAS = [0 - 9]|\text{”} - \text{”}$$
$$Gramatica = (\{DEMAS\}|\{LETRA\})^*\{LETRA\}^+(\{DEMAS\}|\{LETRA\})^*$$

Tanto como para la parte 1 (leer el archivo con `getc(3)`), la parte 2 (usar `read(2)`) y la parte 3 (`mmap(2)`), el analizador léxico es el mismo. Se abre el archivo, saltan los caracteres no validos, hasta encontrar algún carácter valido. Se lee y analiza para saber si es una palabra valida y se contabiliza si es el caso. Luego se sigue leyendo y saltando caracteres inválidos hasta llegar al fin del archivo y finalmente imprimir por pantalla el conteo de palabras.

La implementación del código descrito seria la siguiente:

Código 1: `main.c`

```c
#include <stdio.h>
#include <stdbool.h>

extern int c;

extern void getFile(int argc, char **argv);

extern void closeFile(void);

extern int getCharacter();


/**
 * Returns true if character is a letter.
 */
bool isLetter(int character){
    bool valid = false;
    valid = valid || (character >= 'A' && character <= 'Z'); // UpperCase letter
    valid = valid || (character >= 'a' && character <= 'z'); // lowercase letter
    return valid;
}

/**
 * return true if the char is valid as part of a word.
*/
bool isValidChar(int character){ // Multi-line for readability.
    bool valid = false;
    valid = valid || (character >= '0' && character <= '9'); // number
    valid = valid || isLetter(character); // Letter
    valid = valid || character == '-'; // -
    return valid;
}

/**
 * Scans a letter from file. Skips anything that is not a letter, - or number.
 * Returns false if EOF is found.
*/
bool scan(){
    for(; !isValidChar(c) && c != EOF; c = getCharacter());
    return c != EOF;
}

/**
 * Reads until a space, newline, or any other non valid character (for a word) is found.
 * Returns true if the readed word is a valid word.
 */
bool readWord(){
    bool hasLetter = false;
    for(; isValidChar(c); c = getCharacter()){
        if(isLetter(c)){
            hasLetter = true;
```

```
52          }
53      }
54      return hasLetter;
55  }
56
57  int main(int argc, char **argv){
58      getFile(argc, argv);
59      c = getCharacter();
60
61      int counter = 0;
62      while(scan()){
63          counter += readWord()?1:0;
64      }
65      printf("%i\n", counter);
66
67      closeFile();
68      return 0;
69  }
```

La función `getFile()` se encarga de abrir el archivo de la forma correspondiente. `closeFile()` se encarga de cerrar el archivo (o vaciar la memoria según corresponda). `getCharacter()` entrega el siguiente carácter a leer (o `EOF` según corresponda).

La implementación con flex es bastante sencilla. Tan solo implementar la gramática explicada anteriormente. La implementación seria la siguiente:

Código 2: `4/tarea-1-4.l`

```
1  %option nounput yylineno
2  %option interactive
3  LETRA [a-zA-Z]
4  DEMAS [0-9]|"-"
5
6  %%
7
8  ({DEMAS}|{LETRA})*{LETRA}+({DEMAS}|{LETRA})*    return 1;
9  \n   ;
10 .    ;
11
12 %%
13
14 int yywrap(){
15     return 1;
16 }
```

Finalmente, analizamos los tiempos de ejecución de cada uno de los programas y los comparamos. Para esta comparación, usaremos el programa `time(1)`.

Los resultados son los siguientes:

1. `time ./tarea-1-1 MobyDick.txt`
   Tiempo de ejecucion: 29ms.
2. `time ./tarea-1-2 MobyDick.txt`
   Tiempo de ejecucion: 24ms.
3. `time ./tarea-1-3 MobyDick.txt`
   Tiempo de ejecucion: 19ms.
4. `time ./tarea-1-4 <MobyDick.txt`
   Tiempo de ejecucion: 36ms.
5. `time ./tarea-1-5 <MobyDick.txt`
   Tiempo de ejecucion: 32ms.

Lógicamente, la solución mas rápida fue con `mmap(2)` debido al mapeo directo a la memoria virtual.

Las implementaciones usando flex, aunque mas rápidas de programar, son las mas lentas en *runtime*, probablemente por el *overhead* que produce todo el código auto-generado por flex.

La opción que utiliza `getc(3)` es la mas lenta (de las programadas a manos), debido a que lee carácter a carácter desde el disco. Leer desde la ram (parte 2, pasar el archivo a memoria con `read(2)`) es mas rápido, debido a que hay una única lectura del archivo.