Universidad de La Laguna

ESIT – INFORMÁTICA
Departamento de Ingeniería Informática y de Sistemas

Computabilidad y Algoritmia
Curso 2019-2020

PRACTICA 2: Análisis de texto

2.1. Objetivos

Se propone que el alumnado utilice este ejercicio para poner en práctica los aspectos del desarrollo de programas en C++ sobre los que ya se ha trabajado en la práctica anterior, enfatizando o añadiendo al menos los siguientes:

- Paradigma de programación orientada a objetos: identifique clases y objetos que permitan modelar adecuadamente el escenario de trabajo que se planeta.
- Paradigma de modularidad. El programa debiera escribirse de modo que las diferentes funcionalidades que se precisen fueran encapsuladas en métodos concretos cuya extensión textual se mantuviera acotada.
- Formato propuesto para la escritura de programas en C++ en esta asignatura.
- El ejercicio plantea una buena excusa para comenzar a trabajar con conjuntos de la STL [1], [2]. Identifique situaciones en las que el uso de conjuntos y la operatoria de los mismos facilite la implementación que se realiza.

Si el alumnado tiene dudas respecto a cualquiera de estos aspectos, debiera acudir al foro de discusiones de la asignatura para plantearlas en el foro. Se espera que a través de ese foro el alumnado intercambie experiencias y conocimientos, ayudándose mutuamente a resolver dichas dudas.

También el profesorado de la asignatura intervendrá en las discusiones que pudieran suscitarse, si fuera necesario.

2.2. Rúbrica de evaluación del ejercicio

Señalamos en este apartado los aspectos más relevantes (la lista no es exhaustiva) que el profesorado tendrá en cuenta a la hora de evaluar el trabajo que el alumnado presentará en la sesión de evaluación de la práctica.

Elementos a evaluar:

- Orientación a objetos: el programa ha de seguir ese paradigma de programación.
- Modularidad: el código ha de ser modular.
- El comportamiento del programa debe ajustarse a lo solicitado en el enunciado.
- El código debe estar escrito de acuerdo a los criterios léxicos de escritura de programas en la asignatura (Google C++ Style Guide).
- Capacidad del programador(a) de introducir cambios en el programa desarrollado.

2.3. Introducción

Un programa que lea un fichero de texto clasificando diferentes elementos que identifica en el texto de acuerdo a un conjunto de categorías es lo que se llama habitualmente un analizador léxico [3]. Los analizadores léxicos se utilizan en diverso tipo de aplicaciones como editores de textos, sistemas de recuperación de información, reconocimiento de patrones o compiladores.

En esta práctica se propone el desarrollo de un programa que realice un análisis sencillo de un texto.

2.4. Ejercicio práctico

Desarrollar un programa cliente `Analiza.cpp` que realice el análisis de un fichero de texto cuyo nombre se le pasará por línea de comandos. El resultado se almacenará igualmente en otro fichero de salida cuyo nombre también se toma por línea de comandos.

El programa clasificará los diferentes elementos del texto de entrada en:

1. Palabras especiales
2. Palabras no especiales
3. Números enteros
4. Números en punto flotante
5. Operadores
6. Signos de puntuación

- Las palabras especiales son exclusivamente las que figuran en la Tabla 2.1, que ha sido tomada de las palabras reservadas del lenguaje Java, muchas de las cuales coinciden con las de C++.

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Tabla 2.1: Palabras especiales

- Cualquier secuencia de letras y dígitos, que comiencen por una letra, y que no sea una palabra especial, será considerada una palabra no especial. Ejemplos de palabras no especiales serían *hola*, *licor43* o *cruel2019*.
- Ejemplos de números enteros serían: 0, 1, 3456, 32728 o 6478765444. El programa considerará exclusivamente números enteros sin signo.
- Ejemplos de números en punto flotante serían: 3.1416, 2.701234 o 6578.123. Tampoco se considerarán números negativos en punto flotante, ni números expresados en formato mantisa y exponente (como 3.14e17).

+	-	*	/	=
<	>	<=	<=	!=

Tabla 2.2: Operadores

- El programa considerará operadores a los 10 que figuran en la Tabla 2.2. Téngase en cuenta que hay operadores que tienen un único carácter mientras que otros tienen dos caracteres.
- Por último, el programa considerará los 9 signos de puntuación que figuran en la Tabla 2.3.

(	)	[	]	.
{	}	;	,	

Tabla 2.3: Signos de puntuación

En el fichero de texto de salida el programa imprimirá una secuencia de palabras, cada una de las cuales corresponderá con la categoría a la que pertenezca cada una de los elementos (*lexemas*) que el programa detecta en el fichero de entrada.

La lista de palabras (*tokens*) que corresponde con cada una de las categorías en las que se clasifica el texto de entrada son:

1. Palabra especial: la palabra especial, escrita en mayúsculas. Por ejemplo, ante la palabra '`while`', el programa imprimirá en el fichero de salida el token '`WHILE`'.
2. Palabras no especiales: PAL
3. Número entero: INT
4. Número en punto flotante: FLOT
5. Operadores: OP
6. Signos de puntuación: SIG

Salvo para el caso de las palabras especiales, el programa imprimirá uno de estos tokens seguido de la secuencia de caracteres (lexema) del fichero de entrada que dio lugar a la clasificación que ha establecido el programa. Así, por ejemplo, si el programa detecta el texto `3.1416`, ese texto debería dar lugar a que se imprima en el fichero de salida el texto `FLOT 3.1416`.

Las siguientes deben tomarse como especificaciones del programa a desarrollar:

- Todo el código fuente ha de estar escrito de acuerdo a la guía de estilo de Google para C++ [4]. Dentro de esa guía prestaremos particular atención en este ejercicio a los siguientes aspectos:
  - Formateo del código (apartado *Formatting* en [4])
  - Comentarios de código de diverso tipo (apartado *Comments* en [4])
  - Nominación de identificadores, clases, ficheros, etc. (apartado *Naming* en [4])
- Al ejecutarse en línea de comandos sin parámetros: `./Analiza` el programa mostrará un texto de ayuda indicando la forma de ejecución del programa en la que el usuario puede pasar como parámetros los nombres de los ficheros de entrada y salida que utiliza el programa.
- El programa no es sensible a mayúsculas / minúsculas: tanto la palabra *Adiós* como *ADIÓS* provocan la impresión del token '`PAL`'.
- Se considera que los diferentes elementos del texto (lexemas) se encuentran separados entre sí por espacios en blanco y saltos de línea.
- Cuando el programa detecte alguna situación (secuencia de caracteres) que no consiga clasificar exitosamente, imprimirá en el fichero de salida la palabra `DESCONOCIDA` e intentará seguir clasificando el resto de los caracteres de la entrada.

```
while ( x < 5 ) {
  hola mundo cruel
  if Computabilidad Algoritmia
    3.1416
}
```

Listado 2.1: Ejemplo de fichero de entrada

Ante un fichero de entrada como el que se muestra en el Listado 2.1 el programa debiera generar un fichero de salida como el del Listado 2.2

```
WHILE
SIG (
PAL x
OP <
INT 5
SIG )
SIG {
PAL hola
PAL mundo
PAL cruel
IF
PAL Computabilidad
PAL Algoritmia
FLOT 3.1416
SIG }
```

Listado 2.2: Fichero de salida correspondiente al fichero de entrada 2.1

Bibliografía

[1] STL sets, http://www.cplusplus.com/reference/set/set/

[2] STL sets example, https://tinyurl.com/cyasetexample

[3] Lexical analysis, https://en.wikipedia.org/wiki/Lexical_analysis

[4] Google C++ Style Guide, https://google.github.io/styleguide/cppguide.html