

Explotación de la Información.

Curso 2017-2018

Grado en Ingeniería Informática
Universidad de Alicante

Práctica 1: Tokenizador

Contenido

| | |
|---|----|
| Fecha entrega..... | 3 |
| Qué se pide..... | 3 |
| Prototipo de la clase <i>TokenizadorClase</i> | 3 |
| Método que tokeniza ficheros | 4 |
| Cómo realizar el acceso a un directorio | 5 |
| Prototipo de la clase <i>Tokenizador</i> | 5 |
| Heurísticas para la detección de palabras compuestas y casos especiales | 6 |
| Algoritmo general..... | 6 |
| Guiones | 7 |
| URLs..... | 8 |
| e-mails | 8 |
| Acrónimos | 9 |
| Números..... | 10 |
| Aclaraciones | 11 |
| Evaluación de la práctica..... | 11 |
| Control de acentos y caracteres multi-byte..... | 11 |
| Comprobación automática de los ficheros de prueba | 11 |
| Fichero makefile para la corrección de la práctica | 11 |
| Control de tiempos de ejecución | 12 |

| | |
|--|----|
| Cálculo de eficiencia espacial | 12 |
| Optimización de código de C++..... | 13 |
| Librería STL: listas y mapas..... | 13 |
| Aclaraciones comunes a todas las prácticas | 13 |
| Tratamiento de excepciones | 14 |
| Forma de entrega..... | 14 |
| Ficheros a entregar y formato de entrega | 14 |
| Evaluación | 15 |

Fecha entrega

Del 5 de marzo al 9 de marzo de 2018

Qué se pide

Se pide construir la clase *TokenizadorClase* que segmenta strings en palabras, según el algoritmo visto en la clase de teoría. A partir de la clase *TokenizadorClase*, se pide crear la clase *Tokenizador* que optimiza su eficiencia y detecta una serie de casos especiales de palabras. Asimismo, **hay que calcular la complejidad temporal y espacial del algoritmo de tokenización de casos especiales.**

Prototipo de la clase *TokenizadorClase*

A continuación, se muestra el prototipo de la clase *TokenizadorClase* y el código de algunos métodos *Tokenizar*.

```
class TokenizadorClase {
    friend ostream& operator<<(ostream&, const TokenizadorClase&);
    // cout << "DELIMITADORES: " << delimiters;

public:
    TokenizadorClase (const TokenizadorClase&);

    TokenizadorClase (const string& delimitadoresPalabra);
    // Inicializa variable privada delimiters a delimitadoresPalabra

    TokenizadorClase ();
    // Inicializa delimiters=",;:-/+*\ ' \" {} [] () <> ! ? & # = \t \n \r @"

    ~TokenizadorClase (); // Pone delimiters=""

    TokenizadorClase& operator= (const TokenizadorClase&);

    void Tokenizar (const string& str, list<string>& tokens) const;
    // Tokeniza str devolviendo el resultado en tokens. La lista tokens se
    vaciará antes de almacenar el resultado de la tokenización.
    // El código de esta función tal y como se ha visto en clase se muestra
en este enunciado a continuación del prototipo de la clase
TokenizadorClase

    bool Tokenizar (const string& i, const string& f) const;
    // Tokeniza el fichero i guardando la salida en el fichero f (una
    palabra en cada línea del fichero). Devolverá true si se realiza la
    tokenización de forma correcta enviando a cerr el mensaje
    correspondiente (p.ej. que no exista el archivo i)

    bool Tokenizar (const string & i) const;
    // Tokeniza el fichero i guardando la salida en un fichero de nombre i
añadiéndole extensión .tk (sin eliminar previamente la extensión de i
por ejemplo, del archivo pp.txt se generaría el resultado en pp.txt.tk),
    y que contendrá una palabra en cada línea del fichero. Devolverá true si
    se realiza la tokenización de forma correcta enviando a cerr el mensaje
    correspondiente (p.ej. que no exista el archivo i)

    bool TokenizarListaFicheros (const string& i) const;
    // Tokeniza el fichero i que contiene un nombre de fichero por línea
    guardando la salida en ficheros (uno por cada línea de i) cuyo nombre
    será el leído en i añadiéndole extensión .tk, y que contendrá una
    palabra en cada línea del fichero leído en i. Devolverá true si se
```

realiza la tokenización de forma correcta de todos los archivos que contiene *i*; devolverá false en caso contrario enviando a cerr el mensaje correspondiente (p.ej. que no exista el archivo *i*, o bien enviando a "cerr" los archivos de *i* que no existan, luego no se ha de interrumpir la ejecución si hay algún archivo en *i* que no exista)

```
bool TokenizarDirectorio (const string& i) const;
// Tokeniza todos los archivos que contenga el directorio i, incluyendo
// los de los subdirectorios, guardando la salida en ficheros cuyo nombre
// será el de entrada añadiéndole extensión .tk, y que contendrá una
// palabra en cada línea del fichero. Devolverá true si se realiza la
// tokenización de forma correcta de todos los archivos; devolverá false en
// caso contrario enviando a cerr el mensaje correspondiente (p.ej. que no
// exista el directorio i, o los ficheros que no se hayan podido tokenizar)

void DelimitadoresPalabra(const string& nuevoDelimiters);
// Cambia "delimiters" por "nuevoDelimiters"

void AnyadirDelimitadoresPalabra(const string& nuevoDelimiters); //
// Añade al final de "delimiters" los nuevos delimitadores que aparezcan
// en "nuevoDelimiters" (no se almacenarán caracteres repetidos)

string DelimitadoresPalabra() const;
// Devuelve "delimiters"

private:
    string delimiters;
    // Delimitadores de términos. Aunque se modifique la forma de
    // almacenamiento interna para mejorar la eficiencia, este campo debe
    // permanecer para indicar el orden en que se introdujeron los
    // delimitadores
};

// Versión del tokenizador vista en clase
void
TokenizadorClase::Tokenizar (const string& str, list<string>& tokens) {
    string::size_type lastPos = str.find_first_not_of(delimiters,0);
    string::size_type pos = str.find_first_of(delimiters,lastPos);

    while(string::npos != pos || string::npos != lastPos)
    {
        tokens.push_back(str.substr(lastPos, pos - lastPos));
        lastPos = str.find_first_not_of(delimiters, pos);
        pos = str.find_first_of(delimiters, lastPos);
    }
}
```

Método que tokeniza ficheros

A continuación se muestra una posible versión del **método que tokeniza ficheros**. Esta versión se sugiere como ejemplo de uso de ficheros y STL, pero se aconseja mejorar su eficiencia:

```
bool
TokenizadorClase::Tokenizar (const string& NomFichEntr, const string&
NomFichSal) {
    ifstream i;
    ofstream f;
    string cadena;
    list<string> tokens;

    i.open(NomFichEntr.c_str());
    if(!i) {
        cerr << "ERROR: No existe el archivo: " << NomFichEntr << endl;
        return false;
    }
    else
```

```

{
    while(!i.eof())
    {
        cadena="";
        getline(i, cadena);
        if(cadena.length()!=0)
        {
            Tokenizar(cadena, tokens);
        }
    }
    i.close();

    f.open(NomFichSal.c_str());
    list<string>::iterator itS;
    for(itS= tokens.begin();itS!= tokens.end();itS++)
    {
        f << (*itS) << endl;
    }
    f.close();
    return true;
}

```

Cómo realizar el acceso a un directorio

Ahora se muestra un posible ejemplo de cómo realizar el **acceso a un directorio** (esta versión se sugiere como ejemplo, pero se aconseja mejorar su eficiencia y funcionalidad, por ejemplo para detectar directorios en el método *TokenizarListaFicheros*):

```

#include <iostream>
#include <sys/types.h>
#include <sys/stat.h>
#include <cstdlib>

bool TokenizarDirectorio (const string& dirAIndexar) const {
    struct stat dir;
    // Compruebo la existencia del directorio
    int err=stat(dirAIndexar.c_str(), &dir);
    if(err==-1 || !S_ISDIR(dir.st_mode))
        return false;
    else {
        // Hago una lista en un fichero con find>fich
        string cmd="find "+dirAIndexar+" -follow |sort > .lista_fich";
        system(cmd.c_str());
        return TokenizarListaFicheros(".lista_fich");
    }
}

```

Prototipo de la clase *Tokenizador*

La clase *Tokenizador* tendrá el mismo prototipo y formato de salida que *TokenizadorClase*, salvo que se añadirá en su parte privada los campos *casosEspeciales* y *pasarAminuscSinAcentos*, que se inicializarán en los constructores tal y como se especifica seguidamente. **Asimismo, se podrá modificar el carácter const de los métodos/argumentos aquí descritos siempre que se mantengan las características de funcionamiento previstas:**

```

class Tokenizador {
    friend ostream& operator<<(ostream&, const Tokenizador&);
    // cout << "DELIMITADORES: " << delimiters << " TRATA CASOS ESPECIALES:
    " << casosEspeciales << " PASAR A MINUSCULAS Y SIN ACENTOS: " <<
    pasarAminuscSinAcentos;

```

```

// Aunque se modifique el almacenamiento de los delimitadores por temas
de eficiencia, el campo delimiters se imprimirá con el string con el que
se inicializó el tokenizador

public:
    Tokenizador (const string& delimitadoresPalabra, const bool&
kcasosEspeciales, const bool& minuscSinAcentos);
    // Inicializa delimiters a delimitadoresPalabra; casosEspeciales a
kcasosEspeciales; pasarAminuscSinAcentos a minuscSinAcentos

    Tokenizador ();
    // Inicializa delimiters=" , ; : . - / + * \ ' \" { } [ ] ( ) < > ¡ ¢ £ $ % & ' = \t \n \r \0 ";
casosEspeciales a true; pasarAminuscSinAcentos a false

    void CasosEspeciales (const bool& nuevoCasosEspeciales);
    // Cambia la variable privada "casosEspeciales"

    bool CasosEspeciales ();
    // Devuelve el contenido de la variable privada "casosEspeciales"

    void PasarAminuscSinAcentos (const bool& nuevoPasarAminuscSinAcentos);
    // Cambia la variable privada "pasarAminuscSinAcentos". Atención al
formato de codificación del corpus (comando "file" de Linux). Para la
corrección de la práctica se utilizará el formato actual (ISO-8859).

    bool PasarAminuscSinAcentos ();
    // Devuelve el contenido de la variable privada "pasarAminuscSinAcentos"
private:
    string delimiters; // Delimitadores de términos. Aunque se
modifique la forma de almacenamiento interna para mejorar la eficiencia, este
campo debe permanecer para indicar el orden en que se introdujeron los
delimitadores

    bool casosEspeciales;
    // Si true detectará palabras compuestas y casos especiales

    bool pasarAminuscSinAcentos;
    // Si true pasará el token a minúsculas y quitará acentos, antes de
realizar la tokenización
};

```

Se intentará optimizar la eficiencia de la clase *Tokenizador* mediante cualquier modificación en la representación interna (parte privada de las clases) o el algoritmo utilizado. **No se estudiará la eficiencia de la clase *TokenizadorClase*.**

Heurísticas para la detección de palabras compuestas y casos especiales

Algoritmo general

La clase *Tokenizador* cuando *casosEspeciales* esté a true, detecta una serie de casos especiales de palabras (no se tokenizarán palabras vacías o longitud cero), los cuales se tratarán independientemente del conjunto de delimitadores con el que se configure el tokenizador, además de que **siempre se considerará el espacio en blanco como delimitador** aunque no se haya definido dentro de *delimiters*. Estos casos especiales se comprobarán y resolverán **en el siguiente orden**:

1. URL: los almacenaría como un solo término aunque esté definido como delimitador el `“:./?&=-”`
2. Números decimales: `.103 5,6 1,000,000` (los almacenaría como un solo término aunque esté definido como delimitador el `“.,”`)

3. E-mail: los almacenaría como un solo término aunque esté definido como delimitador el “._@”
4. Detección de acrónimos (U.S.A): los almacenaría como un solo término aunque esté definido como delimitador el “.”
5. Guiones: por ejemplo “MS-DOS” lo almacenaría como “MS-DOS” aunque esté definido como delimitador el “-”

El algoritmo general sería:

- Recorrer la cadena a tokenizar de izquierda a derecha hasta encontrar un delimitador (o un blanco).
- ¿El delimitador aparece entre las excepciones de los casos especiales? (los casos especiales se irán comprobando en el orden descrito anteriormente, quedándose con el primer caso especial que así lo cumpla)
 - En ese caso, el delimitador no habría de ser tenido en cuenta y se continuaría analizando según ese caso especial.
 - En caso contrario, se extraería del token a devolver (siempre que no sea una palabra vacía) sin incluir el delimitador.

A continuación, se describen las heurísticas a aplicar a cada tipo de caso especial.

Guiones

Para las palabras compuestas formadas por guiones, la heurística a implementar es el detectar un término que contenga **por el medio** el carácter “-” sin ningún espacio en blanco, aunque se haya definido “-” dentro de los delimitadores. Se detectará el inicio y final de la palabra compuesta cuando aparezca el blanco (aunque no aparezca entre los delimitadores) o cualquiera de los delimitadores definidos por el usuario (excepto el “-”). Por ejemplo, la siguiente secuencia de comandos:

```
Tokenizador a("-#", true, false);
list<string> tokens;

a.Tokenizar("MS-DOS p1 p2 UN-DOS-TRES", tokens);
// La lista de tokens a devolver debería contener: "MS-DOS, p1, p2, UN-DOS-TRES"

a.Tokenizar("pall -MS-DOS p1 p2", tokens);
// La lista de tokens a devolver debería contener: "pall, MS-DOS, p1, p2"

a.Tokenizar("pall MS-DOS#p3 p1 p2", tokens);
// La lista de tokens a devolver debería contener: "pall, MS-DOS, p3, p1, p2"

a.Tokenizar("pall#MS-DOS#p3 p1 p2", tokens);
// La lista de tokens a devolver debería contener: "pall, MS-DOS, p3, p1, p2"

a.DelimitadoresPalabra("/ ");
a.Tokenizar("MS-DOS p1 p2", tokens);
// La lista de tokens a devolver debería contener: "MS-DOS, p1, p2"

a.Tokenizar("pall -MS-DOS p1 p2", tokens);
// La lista de tokens a devolver debería contener (no quita el primer guion porque no está entre los delimitadores): "pall, -MS-DOS, p1, p2"

a.Tokenizar("pall MS-DOS#p3 p1 p2", tokens);
// La lista de tokens a devolver debería contener (no separa #p3 porque # no está entre los delimitadores): "pall, MS-DOS#p3, p1, p2"
```

```
a.Tokenizar("pal1#MS-DOS#p3 p1 p2", tokens);
// La lista de tokens a devolver debería contener: "pal1#MS-DOS#p3, p1, p2"
```

URLs

Para detectar los límites de las URL se ha de implementar la heurística que detecta una palabra que comienza por espacio en blanco (o cualquiera de los delimitadores definidos por el usuario) y **SOLO** los indicadores de URL “http:” o “https:” o “ftp:” (en minúsculas) seguido por una secuencia de caracteres (incluidos “_./?&=#@” aunque estén definidos como delimitadores), sin ningún blanco por medio. Finalizará cuando se detecte un delimitador (excepto “_./?&=#@”) o un blanco (aunque no esté entre el conjunto de delimitadores). Por ejemplo, la siguiente secuencia de comandos (destacar que **no se aplica la detección de e-mails u otros casos especiales dentro de la URL ya que es el primer caso especial que se comprueba**):

```
Tokenizador a(", ", true, false);
list<string> tokens;
string s = "p0
http://intime.dlsi.ua.es:8080/dossierct/index.jsp?lang=es&status=probable&date
=22-01-2013&newspaper=catedraTelefonicaUA@iuii.ua.es p1 p2";

a.Tokenizar(s, tokens);
// La lista de tokens a devolver debería contener: " p0,
http://intime.dlsi.ua.es:8080/dossierct/index.jsp?lang=es&status=probable&date
=22-01-2013&newspaper=catedraTelefonicaUA@iuii.ua.es, p1, p2"

a.DelimitadoresPalabra("/ ");
a.Tokenizar(s, tokens);
// La lista de tokens a devolver debería contener: " p0,
http://intime.dlsi.ua.es:8080/dossierct/index.jsp?lang=es&status=probable&date
=22-01-2013&newspaper=catedraTelefonicaUA@iuii.ua.es, p1, p2"

a.DelimitadoresPalabra("/ &");
a.Tokenizar(s, tokens);
// La lista de tokens a devolver debería contener: " p0,
http://intime.dlsi.ua.es:8080/dossierct/index.jsp?lang=es&status=probable&date
=22-01-2013&newspaper=catedraTelefonicaUA@iuii.ua.es, p1, p2"

a.DelimitadoresPalabra("/&");
s = "p0
hhttp://intime.dlsi.ua.es:8080/dossierct/index.jsp?lang=es&status=probable&dat
e=22-01-2013 p1 p2";

a.Tokenizar(s, tokens);
// La lista de tokens a devolver debería contener: " p0, hhttp:,
intime.dlsi.ua.es:8080, dossierct, index.jsp?lang=es, status=probable,
date=22-01-2013, p1, p2"
```

e-mails

Para los e-mails, se detectará la presencia de un e-mail en un término que se inicia por un blanco o separador seguido de cualquier carácter, y que contiene **un solo “@”** por medio. Además podrá contener por el medio “.-_” (aunque estén definidos como delimitadores), sin ningún blanco por medio. Se detectará el final por la presencia de un espacio en blanco (aunque no esté definido como delimitador) o delimitador (excepto “.-_”). Por ejemplo:

```
a.DelimitadoresPalabra("@.&");
a.Tokenizar("catedraTelefonicaUA@iuii.ua.es p1 p2", tokens);
```



```
// La lista de tokens a devolver debería contener:
"catedraTelefonicaUA@iuii.ua.es, p1, p2"

a.Tokenizar("pall @iuii.ua.es p1 p2", tokens);
// La lista de tokens a devolver debería contener (se ha quitado el @ porque
es delimitador, pero no se ha detectado un email): "pall, iuii.ua.es, p1, p2"

a.Tokenizar("pall cat@iuii.ua.es@cd p1 p2", tokens);
// La lista de tokens a devolver debería contener (se separa por el primer @
porque al detectar el segundo @ comprueba que no es un email, y al ser el @ un
delimitador, detectaría un email en el segundo @): "pall, cat, iuii.ua.es@cd,
p1, p2"

a.DelimitadoresPalabra("&.");
a.Tokenizar("catedraTelefonicaUA@iuii.ua.es p1 p2", tokens);
// La lista de tokens a devolver debería contener:
"catedraTelefonicaUA@iuii.ua.es, p1, p2"

a.Tokenizar("pall @iuii.ua.es p1 p2", tokens);
// La lista de tokens a devolver debería contener (no se quita el @ porque no
es delimitador, pero no se ha detectado un email): "pall, @iuii.ua.es, p1, p2"

a.Tokenizar("pall&iuii.ua.es p1 p2", tokens);
// La lista de tokens a devolver debería contener: "pall, @iuii.ua.es, p1, p2"

a.Tokenizar("pall&catedra@iuii.ua.es p1 p2", tokens);
// La lista de tokens a devolver debería contener: "pall, catedra@iuii.ua.es,
p1, p2"

a.Tokenizar("pall cat@iuii.ua.es@cd p1 p2", tokens);
// La lista de tokens a devolver debería contener: "pall, cat@iuii.ua.es@cd,
p1, p2"
```

Acrónimos

Para los acrónimos, se detectarán términos que contengan puntos por medio sin ningún blanco (aunque se haya definido el punto como separador). Los puntos han de estar separados por caracteres distinto del propio punto (p.ej. "U..S" no se detectaría como acrónimo). Se detectará el final por la presencia de un espacio en blanco o delimitador o varios puntos seguidos. **Los puntos del principio y final del término acrónimo se eliminarán**: p.ej. "U.S.." se almacenaría como "U.S"; o ".U.S" se almacenaría como "U.S". Por ejemplo:

```
a.DelimitadoresPalabra("@.&");
a.Tokenizar("U.S.A p1 e.g. p2. La", lt1);
// La lista de tokens a devolver debería contener: "U.S.A, p1, e.g, p2, La"

a.Tokenizar("U..S.A p1 e.g. p2. La", lt1);
// La lista de tokens a devolver debería contener: "U, S.A, p1, e, g, p2, La"

a.Tokenizar("...U.S.A p1 e.g. p2. La", lt1);
// La lista de tokens a devolver debería contener: "U.S.A, p1, e, g, p2, La"

a.Tokenizar("Hack.4.Good p1 ", lt1);
// La lista de tokens a devolver debería contener: "Hack.4.Good, p1"

a.DelimitadoresPalabra("");
a.Tokenizar("U.S.A .U.S.A .p1 p1 e.g. p2. La", lt1);
// La lista de tokens a devolver debería contener: "U.S.A, U.S.A, .p1, p1,
e.g, p2., La"

a.Tokenizar("U..S.A p1 e.g. p2. La", lt1);
// La lista de tokens a devolver debería contener (no lo detecta como
acrónimo, sino que el . no es delimitador): "U..S.A, p1, e.g., p2., La"
```

```

a.Tokenizar("...U.S.A p1 e.g. p2. La", lt1);
// La lista de tokens a devolver debería contener: "U.S.A, p1, e.g., p2., La"

a.Tokenizar("a&U.S.A p1 e.g. p2. La", lt1);
// La lista de tokens a devolver debería contener: "a&U.S.A, p1, e.g, p2., La"

a.DelimitadoresPalabra("&");
a.Tokenizar("a&U.S.A p1 e.g. p2. La", lt1);
// La lista de tokens a devolver debería contener (porque & es delimitador):
"a, U.S.A, p1, e.g, p2., La"

```

Números

Igualmente se aplicaría esta última heurística **para los números con o sin “.”** y con los componentes del término que sean solo dígitos numéricos (**no se reconocerán números en formato científico, p.ej. 1,23E+10**). **Varios puntos o comas seguidos no representarán a un número (p.ej. 3..2)**. Se detectará el inicio por un blanco; o separador; o “.” seguido de números en cuyo caso se le añadirá el 0 delante del término (p.ej. “.35” se almacenaría “0.35”, pero “.35” no cumpliría esa condición). Se detectará el final por la presencia de un espacio en blanco; o punto/coma seguido de blanco (los cuales no pertenecerían al término); o los **símbolos %\$€°a seguidos de blanco (los cuales se almacenarían en un término posterior)**; o delimitador; o lo anterior más delimitador. Por ejemplo:

```

a.DelimitadoresPalabra("@.,&");
a.Tokenizar("pall 10.000,34 10,000.34 10.000.123.456.789.009,34
10,000,123,456,789,009.34 20.03 40,03 2005 10. 20, 10.0 20,0 La
20,12.456,7.8.9,", lt1);
// La lista de tokens a devolver debería contener: "pall 10.000,34
10,000.34 10.000.123.456.789.009,34 10,000,123,456,789,009.34 20.03
40,03 2005 10 20 10.0 20,0 La 20,12.456,7.8.9"

a.Tokenizar(".34 ,56", lt1);
// La lista de tokens a devolver debería contener: "0.34 0,56"

a.Tokenizar("pall 10.35% 10,35% 23.000,3% 23,5€ 23.05€ 23,05€ 11.1$ 11.05$
3.4° 4,3ª", lt1);
// La lista de tokens a devolver debería contener: "pall 10.35 % 10,35
% 23.000,3 % 23,5 € 23.05 € 23,05 € 11.1
$ 11.05 $ 3.4 ° 4,3 ª"

a.Tokenizar("pall 10.00a 10.000.a.000 10/12/85 1,23E+10", lt1);
// La lista de tokens a devolver debería contener (no extraería números sino
acrónimos): "pall 10.00a 10.000.a.000 10/12/85 1 23E+10"

a.Tokenizar("pall&10.00@10.000&abc@10/12/85", lt1);
// La lista de tokens a devolver debería contener (extraería un email): "pall
10.00 10.000 abc@10/12/85"

a.Tokenizar(".34@@&,56", lt1);
// La lista de tokens a devolver debería contener: "0.34 0,56"

a.Tokenizar("...10.000.a.000 ,,23.05 10/12/85 1,23E+10", lt1);
// La lista de tokens a devolver debería contener (extraería un acrónimo):
"10.000.a.000 23.05 10/12/85 1 23E+10"

a.DelimitadoresPalabra("");
a.Tokenizar("...10.000.a.000 ,,23.05 10/12/85 1,23E+10", lt1);
// La lista de tokens a devolver debería contener (no quita las , ni los .
porque ahora no son delimitadores): "...10.000.a.000 ,,23.05 10/12/85
1,23E+10"

```

Aclaraciones

Evaluación de la práctica

La nota se calculará del siguiente modo:

- 50% por la corrección automática con los ficheros de prueba.
- 15% por el cálculo de la eficiencia temporal.
- 15% por el cálculo de la eficiencia espacial.
- 20% por la revisión presencial de la práctica.

Control de acentos y caracteres multi-byte

Para saber más de los diferentes formatos de codificación de ficheros, se recomienda la lectura del material de campus virtual: [Ex.Inf. Modulo 1. UTF-8 - Wikipedia, the free encyclopedia.pdf](#).

Es importante destacar que la evaluación de la práctica se realizará con *corpus.tgz*, dejado como material de campus virtual. Dicho corpus de ficheros y los ficheros de prueba están con codificación **ISO-8859** (es decir, **se utilizará un solo byte por cada carácter sea acentuado o no**; ver https://es.wikipedia.org/wiki/ISO/IEC_8859-1) Para comprobar la codificación de un archivo se utiliza el comando de Linux: **file archivo**.

Comprobación automática de los ficheros de prueba

A continuación, se muestra el método propuesto para la comprobación de los ficheros de prueba de las prácticas mediante el script "*corrigeAlumno*" disponible en campus virtual, el cual necesitaría del makefile que se muestra en la siguiente sección:

```
#####
# SE EJECUTA DESDE EL DIRECTORIO QUE CONTIENE src lib include (p.ej.
# ./corrigeAlumno.sh desde el directorio que contiene src lib include)
# DICH0 DIRECTORIO NO CONTENDRA NINGUN ARCHIVO $EJE (p.ej. ya que en este
# fichero EJE=practical, en ese directorio no podra haber ningun fichero que se
# llame "practical")
# DICH0 DIRECTORIO CONTENDRA UN FICHERO makefile QUE GENERE EL EJECUTABLE
# $EJE CUYO PROGRAMA PRINCIPAL SERA $MAIN (p.ej. ya que en este fichero
# EJE=practical, el makefile ha de crear un ejecutable de nombre "practical",
# como el makefile mostrado en la siguiente seccion)
# EL SUBDIRECTORIO src:
# * CONTENDRA LOS FICHEROS DE PRUEBA: *.cpp
# * SUS SALIDAS CORRESPONDIENTES *.cpp.sal
# * NO CONTENDRA NINGUN ARCHIVO $MAIN (el directorio src no debe contener
# ningun archivo con nombre "main.cpp" ya que MAIN=main.cpp)
#
# corrigeAlumno
# Fichero que contiene el main en el makefile
MAIN=main.cpp
# Nombre del ejecutable en el makefile
EJE=practical
```

Fichero makefile para la corrección de la práctica

La práctica se corregirá utilizando el siguiente fichero makefile, en el que **no se añadirán opciones de optimización automática de código (tipo -O3)**:

```
.PHONY= clean
```

```
CC=g++
```

```

OPTIONS= -g -std=gnu++0x
DEBUG= #-D DEBUG
LIBDIR=lib
INCLUDEDIR=include
_OBJ= tokenizadorClase.o tokenizador.o
OBJ = $(patsubst %, $(LIBDIR)/%, $(OBJ))

all: practical

practical:      src/main.cpp $(OBJ)
               $(CC) $(OPTIONS) $(DEBUG) -I$(INCLUDEDIR) src/main.cpp $(OBJ) -o
practical

$(LIBDIR)/%.o : $(LIBDIR)/%.cpp $(INCLUDEDIR)/%.h
               $(CC) $(OPTIONS) $(DEBUG) -c -I$(INCLUDEDIR) -o $@ $<

clean:
      rm -f $(OBJ)

```

Control de tiempos de ejecución

Seguidamente se muestra el fichero *main.cpp*, con el que se realizará el estudio de tiempos de ejecución, el cual se utilizará para evaluar las prácticas:

```

#include <iostream>
#include <string>
#include <list>
#include <sys/resource.h>
#include "tokenizadorClase.h"

using namespace std;

double getcputime(void) {
    struct timeval tim;
    struct rusage ru;
    getrusage(RUSAGE_SELF, &ru);
    tim=ru.ru_utime;
    double t=(double)tim.tv_sec + (double)tim.tv_usec / 1000000.0;
    tim=ru.ru_stime;
    t+=(double)tim.tv_sec + (double)tim.tv_usec / 1000000.0;
    return t;
}

main() {
    long double aa=getcputime();

    Tokenizador a("\t , ; . - + / * _ ' { } [ ] ( ) ! ? & # \" \\ < > ", true, true);
    a.TokenizarListaFicheros("listaFicheros.txt"); // TODO EL CORPUS

    cout << "Ha tardado " << getcputime() - aa << " segundos" << endl;
}

```

Cálculo de eficiencia espacial

Igualmente, para la evaluación de la práctica se analizará la **eficiencia espacial**, para lo que se utilizará: **memory executable**. Este programa hace una estimación (aproximada) de la máxima cantidad de memoria usada por otro programa cualquiera. Esto se consigue haciendo un muestreo del uso de memoria unas 16 veces por segundo, quedándose con el máximo. Dicho programa se creará mediante la siguiente compilación:

```
g++ memory.cpp -o memory
```

Se mostrará la memoria utilizada por el programa en la línea en negrita que se muestra a continuación:

```
Memoria total usada: 10968 Kbytes
"   de datos: 10832 Kbytes
"   de pila: 136 Kbytes
```

Optimización de código de C++

Para la optimización de código de C++ se recomienda la lectura del material dejado en campus virtual (además de otros materiales disponibles en campus virtual y en la Web): [Optimizing C++ - Wikibooks.pdf](#).

Aquí se detallan muchas posibilidades de mejora de la eficiencia temporal de vuestros programas en C++ (p.ej. paso de parámetros por referencia, prefix vs. postfix, declaración variables dentro de un bucle, acceso a memoria secundaria, etc.).

Librería STL: listas y mapas

Se puede utilizar la **librería STL**, para lo que se puede consultar en <http://en.cppreference.com/w/> o en <http://www.cplusplus.com/>.

Por ejemplo, los tipos lista y mapa: *list* y *unordered_set*. En el código anterior se ha mostrado ejemplos de uso del tipo *list*, y a continuación se muestra un ejemplo de *unordered_set*:

```
unordered_set<string> stopWords;
string cadena = "elementoInsertado";
stopWords.insert(cadena);
if(stopWords.find(tokenExtraido)==stopWords.end())...
for (unordered_set<string>::iterator it=stopWords.begin();
it!=stopWords.end(); ++it)
    std::cout << (*it) << std::endl;
it= stopWords.find(nomfich);
if(it== stopWords.end())
...
cout << stopWords.size();
stopWords.clear();
```

Aclaraciones comunes a todas las prácticas

Se aconseja el uso de la herramienta **VALGRIND** para comprobar el manejo correcto de la memoria dinámica. Modo de uso:

```
valgrind - -tool=memcheck - -leak-check=full nombre_del_ejecutable
```

Todas las operaciones especificadas son obligatorias.

Si una clase hace uso de otra clase, en el código nunca se debe incluir el fichero .cpp, sólo el .h.

En la parte PUBLIC no debe aparecer ninguna operación que haga referencia a la representación del tipo, sólo se pueden añadir operaciones de enriquecimiento de la clase.

En la parte PRIVATE de las clases se pueden añadir todos los atributos y métodos que sean necesarios para la implementación de los tipos.

Tratamiento de excepciones

Todos los métodos darán un **mensaje de error (en cerr)** cuando el alumno determine que se produzcan excepciones; para ello, se pueden añadir en la parte privada de la clase aquellas operaciones y variables auxiliares que se necesiten para controlar las excepciones.

Se considera excepción aquello que no permite la normal ejecución de un programa (por ejemplo, problemas al reservar memoria, problemas al abrir un fichero, etc.). NO se considera excepción aquellos errores de tipo lógico debidos a las especificidades de cada clase.

De cualquier modo, todos los métodos deben devolver siempre una variable del tipo que se espera.

Los mensajes de error se mostrarán siempre por la salida de error estándar (cerr). El formato será:

ERROR: mensaje_de_error (al final un salto de línea).

Forma de entrega

La práctica se programará en el Sistema Operativo Linux, y en el lenguaje C++. Deberá compilar con la versión instalada en los laboratorios de la Escuela Politécnica Superior.

La entrega de la práctica se realizará:

- En el SERVIDOR DE PRÁCTICAS en <http://pracdlsi.dlsi.ua.es/>
- A título INDIVIDUAL; por tanto requerirá del alumno que conozca su USUARIO y CONTRASEÑA en el Servidor de Prácticas.
- Se podrán realizar cuantas entregas quiera el alumno, corrigiéndose solo la última entrega realizada. Tras cada entrega el alumno recibirá un correo electrónico indicándole si se ha entregado en el formato correcto y el resultado de la ejecución.

Ficheros a entregar y formato de entrega

La práctica debe ir organizada en 3 subdirectorios:

DIRECTORIO 'include': contiene los ficheros (en MINÚSCULAS):

- "tokenizadorClase.h"
- "tokenizador.h"

DIRECTORIO 'lib': contiene los ficheros (NO deben entregarse los ficheros objeto ".o"):

- "tokenizadorClase.cpp"
- "tokenizador.cpp"

DIRECTORIO 'src':

- **TODOS los ficheros creados y utilizados por el alumno para comprobación de la práctica** (además de los que se dejen disponibles a tal efecto en el campus virtual). Por ejemplo: "main.cpp" y "main.cpp.sal".

- **“complejidad.pdf”** que contenga el cálculo de la complejidad temporal y espacial del algoritmo de tokenización de casos especiales.

Además, en el directorio raíz, deberá aparecer el fichero **“nombres.txt”**: fichero de texto con los datos del autor. El formato de este fichero es:

1_DNI: DNI1

1_NOMBRE: APELLIDO1.1 APELLIDO1.2, NOMBRE1

Cuando llegue el momento de la entrega, toda la estructura de directorios ya explicada (¡ATENCIÓN! excepto el MAKEFILE), debe estar comprimida en un fichero de forma que éste NO supere los 300 K. Ejemplo:

```
user@srv4:~$ ls
```

```
include
lib
nombres.txt
src
```

```
user@srv4:~$ tar cvzf PRACTICA.tgz *
```

Evaluación

La práctica se corregirá casi en su totalidad de un modo automático, por lo que los nombres de las clases, métodos, ficheros a entregar, ejecutables y formatos de salida descritos en el enunciado de la práctica SE HAN DE RESPETAR EN SU TOTALIDAD.

Uno de los objetivos de la práctica es que el alumno sea capaz de comprender un conjunto de instrucciones y sea capaz de llevarlas a cabo. Por tanto, es esencial ajustarse completamente a las especificaciones de la práctica.

Cuando se corrige la práctica, el corrector automático proporcionará ficheros de corrección llamados **“main.cpp”**. Este fichero utilizará la sintaxis definida para cada clase y los nombres de los ficheros asignados a cada una de ellas: únicamente contendrá una serie de instrucciones **#include** con los nombres de los ficheros **“.h”**.

Las prácticas no se pueden modificar una vez corregidas y evaluadas (no hay revisión del código). Por lo tanto, es esencial ajustarse a las condiciones de entrega establecidas en este enunciado.

En especial, se debe llevar cuidado con los nombres de los ficheros y el formato especificado para la salida.