A continuación describiremos cómo hemos implementado los diferentes casos de prueba, así de cómo hemos implementado el módulo para pasar a minúsculas sin acentos y sus complejidades.

Módulo para pasar una cadena a letras minúsculas y sin acentos

```
string Tokenizador::aMinusSinAcentos(const string& str)const{
	string aux=str;
	for (unsigned int i=str.length()-1;i>=0 && i!=string::npos;--i){

		if(aux[i]==(char)192 || aux[i]==(char)193 || aux[i]==(char)194
|| aux[i]==(char)195 || aux[i]==(char)196 || aux[i]==(char)197 ||
aux[i]==(char)224 || aux[i]==(char)225 || aux[i]==(char)226 ||
aux[i]==(char)227 || aux[i]==(char)228 || aux[i]==(char)229 ){
			aux[i]='a';
		}
		else if(aux[i]==(char)200 || aux[i]==(char)201 ||
aux[i]==(char)202 || aux[i]==(char)203 || aux[i]==(char)232 ||
aux[i]==(char)233 || aux[i]==(char)234 || aux[i]==(char)235){
			aux[i]='e';
		}
		else if(aux[i]==(char)204 || aux[i]==(char)205 ||
aux[i]==(char)206 || aux[i]==(char)207 || aux[i]==(char)236 ||
aux[i]==(char)237 || aux[i]==(char)238 || aux[i]==(char)239){
			aux[i]='i';
		}
		else if(aux[i]==(char)210 || aux[i]==(char)211 ||
aux[i]==(char)212 || aux[i]==(char)213 || aux[i]==(char)214 ||
aux[i]==(char)242 || aux[i]==(char)243 || aux[i]==(char)244 ||
aux[i]==(char)245 || aux[i]==(char)246){
			aux[i]='o';
		}
		else if(aux[i]==(char)217 || aux[i]==(char)218 ||
aux[i]==(char)219 || aux[i]==(char)220 || aux[i]==(char)249 ||
aux[i]==(char)250 || aux[i]==(char)251 || aux[i]==(char)252){
			aux[i]='u';
		}
		else if(aux[i]==(char)221 || aux[i]==(char)253 ||
aux[i]==(char)255){
			aux[i]='y';
		}
		else if(aux[i]==(char)209){
			aux[i]='ñ';
		}
		else if(aux[i]==(char)231 || aux[i]==(char)199){
			aux[i]='ç';
		}

		else{
			//Pasar a minúsculas
			aux[i]=(char)tolower(aux[i]);
		}
	}
	return aux;
}
```

En este módulo, comprobamos cada uno de los caracteres del string pasado por referencia para pasarlo a minúsculas y sin acentos.

Sin importar si es mayúscula o no, el carácter será pasado a minúsculas y, en caso de ser uno de los casos no reconocidos por la función `tolower()` será comprobado directamente desde su representación numérica siguiendo la codificación ISO-8859-15.

La complejidad de este módulo es N, siendo N el tamaño del string, ya que ha de ser recorrido en su totalidad para completar la transformación.

Debido a lo anteriormente mencionado no existen caso mejor y peor en los que la complejidad de este módulo pueda ser minimizada.

CASOS ESPECIALES

Debe tenerse en cuenta que en todos los módulos siguientes se detecta, entre los delimitadores, el espacio en blanco.

URL

```
void Tokenizador::URL(string::size_type& pos, string::size_type&
lastPos,const string& aux, const string& delimitadores,list<string>& tokens)
const{
	string delimaux=delimitadores;
	string sinDelimURL; //Array sin delimitadores de URL
	for(unsigned int i=delimaux.length()-1;i>=0 && i!=string::npos;--i){
		switch(delimaux[i]){
			case ':':
				break;
			case '/':
				break;
			case '.':
				break;
			case '?':
				break;
			case '&':
				break;
			case '-':
				break;
			case '=':
				break;
			case '@':
				break;
			default:
				sinDelimURL+= delimaux[i];
				break;
		}
	}
	if(aux.find_first_of(sinDelimURL, lastPos)!=string::npos){
		pos = aux.find_first_of(sinDelimURL, lastPos); //Obtenemos el
espacio completo de la URL
	}
	tokens.push_back(aux.substr(lastPos, pos - lastPos)); //Metemos la URL
en el array de TOKENS
}
```

En el caso de detectar que se va a Tokenizar una URL, es decir, cuando el fragmento a tokenizar comienza por “http:”, “https:” o “ftp:”, creará un string auxiliar donde se guardarán todos los delimitadores que no sean parte de los reconocidos por una URL.

A partir de este string comprobamos que el puntero pos, que indica el final del token, no se salga del string, entonces, actualizamos el final del string y lo guardamos en la lista de tokens.

De nuevo, no existe un caso mejor ni peor para este módulo, de modo que su complejidad será N, siendo N, en este caso, la longitud del string de delimitadores.

Esto es así ya que debe recorrer todo el string para eliminar los elementos innecesarios para el token.

Implementando el módulo mediante un switch en lugar de un if con condiciones anidadas se reduce la complejidad temporal, aunque aumenta la espacial, debido a que switch crea una copia local de la variable a probar.

NÚMEROS

```
void Tokenizador::Numeros(string::size_type& pos, string::size_type&
lastPos,string::size_type& posAcron,const string& aux, const string&
delimitadores, list<string>& tokens)const{
	string delimaux=delimitadores;
	bool isNumber=true;
	bool decimal=false;

	//Calculamos el final del token
	string::size_type
nuevoini=aux.find_first_not_of(delimaux+".,",lastPos);
	string::size_type auxpos=pos;
	string atokenizar=aux.substr(nuevoini, auxpos-nuevoini);
	string especial="";//Especial es el caso con el simbolo del euro y el
espacio
	especial+=(char)164;

	//Mientras sea decimal
	while(aux.substr(pos, aux.find_first_not_of(delimaux+".,",pos)-
pos)=="." || aux.substr(pos, aux.find_first_not_of(delimaux+".,",pos)-
pos)==","){
		decimal=true;
		nuevoini = aux.find_first_not_of(delimaux+".,", pos);//Nueva
posición de inicio auxiliar
		auxpos=aux.find_first_of(delimaux+".,",nuevoini);
		if(auxpos==string::npos){//Si se va fuera del string
			auxpos=aux.length();
			if(aux.substr(pos,auxpos-pos)=="."
||aux.substr(pos,auxpos-pos)==","){
				break;
			}
		}
		if(nuevoini==string::npos){//Si se va fuera del string
			nuevoini=aux.length();
		}

		atokenizar=aux.substr(nuevoini, auxpos-nuevoini);
		if(esNumero(atokenizar)){
```

```
                        if(auxpos!=string::npos){
                                pos=auxpos;
                        }
                        else{
                                pos=aux.length();
                        }
                }
                else{
                        auxpos=aux.find_first_not_of(delimaux+".,",auxpos);
                        if(auxpos==string::npos){//Si se va fuera del string
                                auxpos=aux.length();
                                atokenizar=aux.substr(nuevoini, auxpos-nuevoini);
                                if((atokenizar.find("%")>0 &&
atokenizar.find("%")!= string::npos)|| (atokenizar.find("$")>0 &&
atokenizar.find("$")!= string::npos)|| (atokenizar.find("º")>0&&
atokenizar.find("º")!= string::npos) ||      (atokenizar.find("ª")>0 &&
atokenizar.find("ª")!= string::npos)|| (atokenizar.find(especial)>0 &&
atokenizar.find(especial)!= string::npos)){//Debemos asegurarnos de que no
coge a los npos

        pos=aux.find_first_of(delimaux+".,$%ºª"+(char)164,nuevoini);
                                }
                        }
                        atokenizar=aux.substr(nuevoini, auxpos-nuevoini);
                        //En el caso en el que terminen en "%$(euro)ºª" seguidos
de un espacio
                        if((atokenizar.find("% ")>0 && atokenizar.find("% ")!=
string::npos)|| (atokenizar.find("$ ")>0 && atokenizar.find("$ ")!=
string::npos)|| (atokenizar.find("º ")>0&& atokenizar.find("º ")!=
string::npos) ||    (atokenizar.find("ª ")>0 && atokenizar.find("ª ")!=
string::npos)|| (atokenizar.find(especial+" ")>0 &&
atokenizar.find(especial+" ")!= string::npos)){//Debemos asegurarnos de que
no coge a los npos

        pos=aux.find_first_of(delimaux+".,$%ºª"+(char)164,nuevoini);
                        }else{
                                isNumber=false;
                        }
                        break;
                }
        }

        if(!decimal){
                auxpos=aux.find_first_not_of(delimaux,auxpos);
                if(auxpos==string::npos){//Si se va fuera del string
                        auxpos=aux.length();
                        atokenizar=aux.substr(nuevoini, auxpos-nuevoini);
                        if((atokenizar.find("%")>0 && atokenizar.find("%")!=
string::npos)|| (atokenizar.find("$")>0 && atokenizar.find("$")!=
string::npos)|| (atokenizar.find("º")>0&& atokenizar.find("º")!=
string::npos) ||    (atokenizar.find("ª")>0 && atokenizar.find("ª")!=
string::npos)|| (atokenizar.find(especial)>0 && atokenizar.find(especial)!=
string::npos)){//Debemos asegurarnos de que no coge a los npos

        pos=aux.find_first_of(delimaux+".,$%ºª"+(char)164,nuevoini);
                        }
                }
                else{
                        atokenizar=aux.substr(nuevoini, auxpos-nuevoini);
```

```
                        //En el caso en el que terminen en "%$(euro)ºª" seguidos
de un espacio
                        if((atokenizar.find("% ")>0 && atokenizar.find("% ")!=
string::npos)|| (atokenizar.find("$ ")>0 && atokenizar.find("$ ")!=
string::npos)|| (atokenizar.find("º ")>0&& atokenizar.find("º ")!=
string::npos) ||   (atokenizar.find("ª ")>0 && atokenizar.find("ª ")!=
string::npos)|| (atokenizar.find(especial+" ")>0 &&
atokenizar.find(especial+" ")!= string::npos)){//Debemos asegurarnos de que
no coge a los npos
                pos=aux.find_first_of(delimaux+".,$%ºª"+(char)164,nuevoini);
                        }
                }
        }

        //Si no es un número se tratará como un Acronimo
        if(!isNumber){
                Acronimos(pos, lastPos, posAcron, aux, delimaux, tokens);
        }
        else{
                lastPos=aux.find_first_not_of(delimaux+".,",lastPos);//Nos
seguramos de que el primer token se obtenga sin el punto o la coma en el caso
de ser .5165
                if(lastPos>=1){
                        //si es un decimal .5815
                        if(aux.substr(lastPos-1, lastPos-(lastPos-1))=="." ||
(aux.substr(lastPos-1, lastPos-(lastPos-1))==","){
                                tokens.push_back("0"+aux.substr(lastPos-1,pos-
(lastPos-1)));//Decimal que empieza por "." o ","
                        }
                        //si es un decimal 6485165.684516541
                        else{
                                tokens.push_back(aux.substr(lastPos,pos-lastPos));
                        }
                }
                else{
                        tokens.push_back(aux.substr(lastPos,pos-lastPos));
                }
        }
}
```

En el caso de detectar que el string a Tokenizar comienza por un número, se llamará a este módulo.

Es, en comparación, el módulo más costoso espacialmente (muchas líneas de código), especialmente por el hecho de que debe realizar comprobaciones varias, como detectar si hay números que terminen en “&%ªº€” más espacio, debiendo Tokenizar dichos caracteres aparte, en caso de que no sean parte del conjunto de delimitadores, o números decimales que empiecen por “.” o “,” como, por ejemplo “.25” y ”,15” y tokenizarlos como “0.25” y “0,15”.

En el caso de encontrar una letra por medio de un decimal, los cuales pueden contener tantos puntos y comas como desee, en cualquier orden, mientras cumplan con las condiciones, se detectará que el token no era un número y se tokenizará de nuevo siguiendo las reglas de los Acrónimos, que se explican más adelante.

En cuanto a la complejidad del algoritmo, nos encontramos con varios casos:

- Primero: La complejidad máxima del algoritmo depende de si se detecta o no que, en vez de ser un número era un acrónimo. En caso de que fuera un acrónimo la complejidad sería N*D, siendo N la complejidad de Tokenizar un Acrónimo y D la complejidad de detectar que no era un número, que dependerá del número de puntos y comas existentes en el string hasta la sección crítica.
- Segundo: La complejidad máxima del algoritmo dependerá también de si el número detectado es un decimal o no, es decir, en el caso de que no se detecten puntos o comas la complejidad será mucho menor, a ser más concreto sería constante. Mientras que si se detectan puntos y comas la complejidad será M, siendo M dependiente del número de puntos y comas existentes en el decimal.

De este modo nos encontramos con 3 complejidades:

$O(N^2)$ $O(1)$ $O(N)$

Entre dichas complejidades podemos afirmar que la mejor es la complejidad constante, obtenida a partir del número NO DECIMAL, y la peor es la cuadrática obtenida del caso en el que se detecta un acrónimo.

EMAIL

```
void Tokenizador::Email(string::size_type& pos, string::size_type&
lastPos,const string& aux, const string& delimitadores, list<string>&
tokens)const{
	string delimaux=delimitadores;
	string sinDelimURL; //Array sin delimitadores de URL

	for(unsigned int i=delimaux.length();i>=0 && i!=string::npos ;--i){
		if(!(delimaux[i]=='.' || delimaux[i]=='_' || delimaux[i]=='@')){
			sinDelimURL+= delimaux[i];
		}
	}

	string comprueba=aux.substr(lastPos, aux.find_first_of(sinDelimURL,
lastPos) - lastPos);

	while(Count(comprueba,"@")>1){//Mientras haya más de una arroba irá
tokenizando las palabras hasta la arroba
		tokens.push_back(aux.substr(lastPos, pos - lastPos)); //Metemos
el e-mail en tokens
		lastPos = aux.find_first_not_of(delimaux, pos);
		pos = aux.find_first_of(delimaux, lastPos);
		comprueba=aux.substr(lastPos, aux.find_first_of(sinDelimURL,
lastPos) - lastPos);
	}

	pos = aux.find_first_of(sinDelimURL, lastPos); //Obtenemos el espacio
completo de la URL
	tokens.push_back(aux.substr(lastPos, pos - lastPos)); //Metemos el e-
mail en tokens


}
```

Llamaremos a este módulo cuando se detecte que en el string a Tokenizar hay un "@".

En este caso, se comprobará mediante una función que recorre el string contando cuantas veces aparece un carácter en dicho string, y cuyo coste es lineal si es el único "@" existente en el string.

En caso de no ser el único, se tokenizará el string incluyendo el "@" en los tokenizadores hasta que solamente quede 1.

Esto nos dice que, en el mejor caso, la complejidad del algoritmo es constante, ya que tokeniza directamente, mientras que, en el peor caso, la complejidad será lineal, dependiendo del número de "@" que existan dentro del string a Tokenizar.

ACRÓNIMOS

```
void Tokenizador::Acronimos(string::size_type& pos, string::size_type&
lastPos,string::size_type& posAcron,const string& aux, const string&
delimitadores, list<string>& tokens)const{
	string delimaux=delimitadores;
	string::size_type final=posAcron;
	string::size_type prepunto=
aux.find_first_of(delimaux+".",1+posAcron);//Si hay un punto antes del
acrónimo
	if(posAcron==lastPos){

	if(aux.substr(prepunto,aux.find_first_not_of(delimaux+".",prepunto)-
prepunto)=="."){
			lastPos = aux.find_first_not_of(delimaux+".", lastPos);
		}
	}
	//Cuerpo principal del acrónimo
	while(aux.substr(final, aux.find_first_not_of(delimaux+".",final)-
final)=="."){

	final=aux.find_first_of(delimaux+".",aux.find_first_not_of(delimaux+".
",final));
		if (string::npos == final && string::npos!=lastPos){//Al iniciar
lastPos será 0, pero si pos es npos dará error, por lo que lo cambiamos al
length del string
			final=aux.length();
		}
	}
	//Metemos en la lista de tokens el acrónimo
	tokens.push_back(aux.substr(lastPos, final - lastPos));//Introduce
palabra.palabra.palabra ..... mientras cumpla con la condición
	//quitamos los puntos finales
	final=aux.find_first_not_of(".",final);

	pos=final;
}
```

Este módulo es llamado en el único caso que se detecte que, en el string a Tokenizar existe un conjunto de letras entre las que existe un "." entre ellas, sin la presencia de otro delimitador y/o espacio.

Además de esto, se eliminan los puntos iniciales y finales, de modo que únicamente se tokenice el acrónimo.

En este módulo siempre se comprobará si existen o no puntos antes y después, de modo que esa parte es constante.

Podemos observar un mejor caso, en el que únicamente existe un punto dentro del acrónimo, en el cual la complejidad será constante, mientras que en el peor caso, se deberá recalcular el final del token tantas veces como puntos haya que cumplan la condición de acrónimo. En este peor caso la complejidad sería lineal.

GUIONES

```
void Tokenizador::Guiones(string::size_type& pos, string::size_type&
lastPos,const string& aux, const string& delimitadores, list<string>&
tokens)const{
	string delimaux=delimitadores;

	string::size_type final=pos;
	while(aux.substr(final, aux.find_first_not_of(delimaux+"-",final)-
final)=="-"){

	final=aux.find_first_of(delimaux,aux.find_first_not_of(delimaux+"-
",final));
		if (string::npos == final && string::npos!=lastPos){//Al iniciar
lastPos será 0, pero si pos es npos dará error, por lo que lo cambiamos al
length del string
			final=aux.length();
		}
	}
	tokens.push_back(aux.substr(lastPos, final - lastPos));//Introduce
palabra-palabra-palabra ..... mientras cumpla con la condición
	pos=final;
}
```

A este módulo se accede de forma parecida a cómo se hace en el apartado de los acrónimos, excepto por el hecho de que no se debe tener en cuenta a los guiones previos, ya que no se detectarán directamente.

Podemos observar que, al igual que en el caso de los acrónimos, existen un mejor y peor caso, ya que también se detecta la presencia de más de un guion dentro del token mientras se encuentre entre caracteres que no sean espacio o delimitador.

En el mejor caso, en el que únicamente existe un único guion dentro del token, la complejidad será constante, mientras que en el caso de que haya más de uno la complejidad será lineal, dependiendo del número de guiones que cumplan la condición de palabra con guiones.