Laboratorio di Programmazione – AA 2004-2005 – Andrea Sterbini

Modulo 1: Funzioni per tokenizzare una email

Si deve realizzare un modulo software, contenuto in un unico file "**modulo1.c**", che permetta la tokenizzazione del testo di una email e l'individuazione degli headers della email in modo da produrre un vettore di coppie <parola, contesto>.
A questo scopo si debbono implementare le funzioni che seguono:

int leggiFile(char * nomefile, int dimensione, char * testo);

Questa funzione riceve i tre parametri **nomefile, dimensione** e **testo**, che indicano rispettivamente:
- **nomefile**: il nome del file da aprire e leggere
- **dimensione**: il numero di bytes (meno 1 per lo '\0') che deve essere letto da tale file
- **testo**: il puntatore ad una zona di memoria grande **dimensione** caratteri in cui copiare il testo letto.

La funzione deve aprire il file e copiarne il contenuto nella zona di memoria **testo**.
La stringa che viene copiata nella variabile testo deve essere terminata dal carattere '\0'.
Se la dimensione della stringa testo è minore della dimensione del file, si copiano solo i primi caratteri.
Opportuni controlli debbono essere fatti sui parametri per assicurarsi che essi siano corretti.
Eventuali errori verranno comunicati dalla funzione tornando un codice intero che descrive l'errore.
Si veda il file "modulo1.h" per il valore ed il significato di tali codici di errore.
Se la funzione termina senza errori il codice da tornare è il valore **RESULT_OK** (zero).

int estraiHeaders(char * email, char * headers, int dimh);

Questa funzione riceve in ingresso un blocco di testo che contiene una **email** e ne copia il blocco delle intestazioni nella stringa **headers**. Gli argomenti sono:
- **email**: che contiene il testo della email, terminato dal carattere '\0'
- **headers**: è un blocco di memoria grande **dimh** caratteri in cui copiare le intestazioni della email,
- **dimh**: dimensione della stringa **headers** in caratteri

La funzione deve leggere dalla email contenuta nel parametro **email** gli headers e copiarne il contenuto nel blocco di memoria **headers**. La stringa headers dev'essere terminata dal carattere '\0'.
Si faccia attenzione a non copiare più caratteri di quanto spazio è disponibile. Se lo spazio dimh non è sufficiente si torni il giusto codice di errore.
Se la funzione termina senza errori il codice da tornare è il valore **RESULT_OK** (zero).
Come si distinguono gli headers dal corpo del messaggio? Gli headers sono linee di testo che **contengono almeno un carattere** prima di arrivare all'accapo, sono seguiti da una linea vuota, poi segue il corpo del messaggio. Esempio (in grassetto gli headers):

From: andrea@sterbini.org
To: babbo_natale@northpole.org
Subject: Cosa vorrei per Natale

Caro Babbo Natale,
quest'anno ho cercato di essere molto buono e di non bocciare nessuno, ma talvolta ho dovuto farlo.

Ti prego di portarmi in regalo un Lego Mindstorms per poter insegnare ai ragazzi del corso a programmare in microcodice Lego.

Ciao
Andrea

int primoHeader(char * headers, char * primo_header, int dim);

Questa funzione riceve in ingresso gli argomenti:
- **headers**: stringa che contiene tutti gli headers della email, terminata da '\0'
- **primo_header**: stringa grande **dim** caratteri in cui copiare il primo header.
- **dim**: dimensione in caratteri della stringa **primo_header**.

La funzione esamina la stringa degli **headers,** e copia il primo header nella stringa **primo_header**.
La stringa deve essere terminata dal carattere '\0'. Dato che primo_header è grande **dim** caratteri, si torni un opportuno codice di errore (vedi "modulo1.h") se questi non sono sufficienti.
La funzione deve controllare i parametri in modo che siano corretti, e tornare un codice di errore se non lo sono (si veda il file "modulo1.h" per i codici di errore).
Se la funzione non ha avuto errori si torni il risultato **RESULT_OK** (zero).
Come si riconosce uno header?
Uno header inizia con una sequenza di caratteri senza spazi seguita opzionalmente dal carattere ':', poi seguita da spazio, poi da testo fino ad un accapo.
Lo header può continuare sulle righe seguenti se ciascuna riga inizia con un carattere spazio o tab.
Esempio (in grassetto il primo header):

> **From sterbini.di.uniroma1.it**
> **151.100.17.XXX**
> **fhghjndòkjgnhòksjh**
> From: andrea@sterbini.org
> To: babbo_natale@northpole.org
> Subject: Cosa vorrei per Natale

int parseHeader(char * header, char * tipo, int dimt, char * valore, int dimv);

La funzione deve analizzare un header e spezzarlo nelle due parti: **tipo** e **valore**.
Nell'esempio precedente lo header "From: andrea@sterbini.org" deve fornire il tipo "From:" ed il valore "andrea@sterbini.org".
I parametri della funzione sono:
- **header**: stringa terminata da '\0' che contiene tutto lo header,
- **tipo**: puntatore ad una stringa di dimensioni **dimt** in cui copiare il tipo dello header,
- **dimt**: dimensioni della stringa **tipo**,
- **valore**: puntatore ad una stringa di dimensioni **dimv** in cui copiare il valore dello header,
- **dimv**: dimensioni della stringa **valore.**

Sia la stringa copiata in **tipo** che quella copiata in **valore** debbono essere terminate dal carattere '\0'.
Si torni un opportuno codice di errore se il numero di caratteri dimt o dimv non è sufficiente a contenere la stringa da copiare. Si torni il valore **RESULT_OK** (zero) altrimenti.

int primaParola(char * testo, char * parola, int dim);

La funzione deve estrarre dalla stringa **testo** (terminata dal carattere '\0') la prima parola e copiarla nella stringa **parola** di dimensioni **dim** caratteri.
La parola dev'essere terminata dal carattere '\0'.
Se il numero di caratteri **dim** non è sufficiente si deve ritornare un opportuno codice di errore (si veda il file "modulo1.h").
Se la funzione termina correttamente si torni il codice **RESULT_OK** (zero).
Come si riconosce una parola?
Una parola è una sequenza di caratteri alfanumerici, ogni altro carattere va considerato separatore.
Una eventuale sequenza di separatori iniziale deve essere ignorata.
Esempio: la prima parola della stringa "$%$£&%**pippo96** $%£"$%" è "**pippo96**".

int tokenizzaEmail(char * email, Token tokenArray[], int dimT);

La funzione riceve come parametro una stringa contenente una email, ed usando le funzioni precedenti la analizza e inserisce nel vettore **tokenArray** di dimensione **dimT** i token ricavati dallo header e dal corpo della email. Se la email contiene più di dimT tokens si deve tornare il codice BUFFER_TOO_SMALL per indicare che altri token sono disponibili.
I token vanno inseriti nel vettore **tokenArray** nell'ordine in cui appaiono nella email.
La struttura dati **Token** (definita nel file "modulo1.h") è formata dai due campi **parola** e **contesto** (di tipo char *) in cui vanno inseriti il contesto e la parola estratta dalla email.
Se la parola è stata estratta da uno header il contesto sarà il **tipo** di header.
Se la parola è stata estratta dal corpo del messaggio il contesto sarà il valore NULL.

int printTokens(Token tokenArray[], int dimT);

La funzione stampa su **stdout** (una per ciascuna riga e nell'ordine di apparizione) le stringhe "**tipo*parola**" ottenute dai Token passati nel vettore **tokenArray** concatenando il tipo con la parola del token. Per i token estratti dal corpo del messaggio (quelli con tipo == NULL) va stampata, una per riga, solo la parola.

Esempio: stampando i tokens estratti dalla email vista sopra si produrrà l'output:

```
From:*andrea
From:*sterbini
From:*org
To:*babbo
To:*natale
To:*northpole
To:*org
Subject:*Cosa
Subject:*vorrei
Subject:*per
Subject:*Natale
Caro
Babbo
Natale
quest
anno
ho
cercato
di
essere
molto
buono
e
di
non
bocciare
nessuno
..... (segue)
```

Funzioni di test

Si sviluppi per ciascuna delle funzioni precedenti una funzione di test che verifica se il comportamento della funzione da testare è corretto. Le funzioni di test debbono essere contenute tutte nel file che si chiama "**testModulo1.c**". I prototipi delle funzioni di test sono i seguenti e sono contenuti nel file "**testModulo1.h**" allegato, assieme alla definizione dei codici da tornare come risultato dei test.

```
int test_leggiFile(         int tipo_di_test, int n_argomento);
int test_estraiHeaders(     int tipo_di_test, int n_argomento);
int test_primoHeader(       int tipo_di_test, int n_argomento);
int test_parseHeader(       int tipo_di_test, int n_argomento);
int test_primaParola(       int tipo_di_test, int n_argomento);
int test_tokenizzaEmail(    int tipo_di_test, int n_argomento);
int test_printTokens(       int tipo_di_test, int n_argomento);
```

Le funzioni si comportano tutte nello stesso modo; ricevono i due parametri interi:

- **tipo_di_test**: indica quale codice risultato ci aspettiamo come risposta dalla funzione da testare (si veda in "modulo1.h" per la definizione dei codici risultato delle funzioni da testare). Si può dire che ci suggerisca che genere di argomento dobbiamo passare alla funzione da testare per far uscire proprio quel codice di risultato.
- **n_argomento**: indica rispetto a quale argomento facciamo il test, partendo da 0 per il primo argomento.

Il risultato della funzione può essere uno dei valori interi seguenti (definiti in "testModulo1.h"):

- TEST_PASSED 0 il test è andato bene, la funzione si comporta come atteso
- TEST_FAILED 1 il test è fallito, la funzione non si comporta come atteso
- ARG_MISSING 2 la funzione da testare non ha questo argomento
- NOT_VALID 3 questo tipo di test non ha senso per questo argomento
- NOT_IMPLEMENTED 4 questo test per questo argomento non è stato implementato

Solo nel caso in cui la risposta sia TEST_FAILED, la funzione di test **deve stampare** un messaggio su stdout per spiegare perchè la funzione testata non si è comportata nella maniera corretta.
Ad esempio si può stampare:

- cosa si è passato alla funzione,
- cosa ci si aspettava di trovare,
- cosa si è ottenuto invece,
- quale si pensa sia il motivo dell'errore

Ad esempio, una funzione di test di nome **test_XXX** che riceva la coppia di valori:

- tipo-di-test: `NULL_POINTER`
- argomento: 0

Cercherà di scoprire se la corrispondente funzione XXX da testare torna il valore NULL_POINTER se gli viene passato NULL come primo argomento. Se questo accade vuol dire che XXX fa correttamente il suo lavoro, il test è passato e test_XXX torna il codice TEST_PASSED; altrimenti vuol dire che XXX non è ben fatta, si torna TEST_FAILED e si stampa su stdout un messaggio del tipo di:

```
Test fallito: funzione testata:  XXX(NULL,....)
              risultato atteso:  NULL_POINTER
              risultato ottenuto:RESULT_OK
              spiegazione:  la funzione XXX non controlla se il
                            primo argomento è un puntatore nullo.
```

Attenzione:

- Se la funzione non ha quell'argomento tornate il codice ARG_MISSING,
- Se quel codice di errore non ha senso con quell'argomento tornate NOT_VALID,
- Se non implementate qualcuna delle combinazioni codice/argomento tornate il codice NOT_IMPLEMENTED