Modulo 1: tokenizzazione

Laboratorio di Programmazione

Canale E-O

Andrea Sterbini

Modulo 1: Tokenizzazione del testo

- Dobbiamo spezzare il testo in **parole**
- Riconoscere in quale parte della email (contesto) la parola si trova:
  - From, Subject, Body …
- Tornare l'elenco delle parole concatenate col loro contesto (a parte il corpo del messaggio)
  - Subject*Nigeria
  - From*yahoo
  - Viagra

Come sviluppare il progetto

- Per sviluppare potete usare:
  - **Linux** **(ideale)**
    - **Knoppix!!!** **(boot da CD)**
  - www.cygwin.com (almeno gcc, bash e gdb)
  - DevC++ con gcc o con Mingw
  - altri ambienti di sviluppo (meglio non VC++)
- **MA ATTENZIONE!** I test verranno eseguiti in ambiente **Linux** usando **gcc** e alcune librerie

Come consegnare i pezzi del progetto

- Vi ho dato 2 files con i prototipi delle funzioni:
  - modulo1.h funzioni da sviluppare
  - testsModulo1.h tests delle funzioni
- Dovrete sviluppare 2 files:
  - modulo1.c codice delle funzioni
  - testModulo1.c codice dei test
- Consegna su **twiki.di.uniroma1.it**
- Scadenza il **6 aprile (inamovibile)**

Quali librerie potete usare

- Sicuramente stdlib e la libreria matematica
  - #include <stdio.h> per printf/scanf
  - #include <stdlib.h> per malloc
  - #include <math.h> matematica
  - Opzione –lm in compilazione
  - #include <strings.h> per strtok
  - Opzione –lc (forse inutile?)

Funzioni di test (Unit testing)

- Per ogni funzione da implementare dovete realizzare anche una funzione che ne verifichi l'uso corretto
- Questo è un modo di sviluppare che permette di catturare molti errori sia in fase di sviluppo che in fase di manutenzione
- Ogni volta che si modifica il software si rieseguono i tests e si verifica se qualcosa ha smesso di funzionare da qualche parte
- C'è chi sviluppa PRIMA i test:
  - Quando i test passano lo sviluppo è FINITO! :-)

Elenco delle delle funzioni

- int leggiFile(char * nomefile, int dimensione, char * testo);
- int estraiHeaders(char * email, char * headers, int dimh);
- int primoHeader(char * headers, char * primo_header, int dim);
- int parseHeader(char * header, char * tipo, int dimt, char * valore, int dimv);
- int primaParola(char * testo, char * parola, int dim);
- int tokenizzaEmail(char * email, Token tokenArray[], int dimT);
- int printTokens(Token tokenArray[], int dimT);

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

- nomefile: il nome del file da aprire e leggere
- dimensione: il numero di bytes (incluso 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 stringa che viene copiata nella variabile testo deve essere terminata dal carattere '\0'.
  - Se **dimensione** è minore della dimensione del file, si copiano solo i primi caratteri.

Gestione degli errori

- Opportuni controlli debbono essere fatti sui parametri per assicurarsi che essi siano corretti.
- Gli errori vengono comunicati 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);

- Riceve un blocco di testo che contiene una email e ne copia il blocco delle intestazioni nella stringa headers.
  - 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

Come si distinguono gli headers dal corpo del messaggio?

- Sono linee di testo non vuote (con almeno un carattere prima dell'accapo), seguiti da una linea vuota, poi segue il corpo del messaggio.
  - **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);

- 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'.

Come si riconosce uno header?

- Inizia con una sequenza di caratteri senza spazi seguita opz. dal carattere ':', poi seguita da spazio, poi da testo fino ad un accapo.
- Può continuare sulle righe seguenti se ciascuna iniziano con spazio o tab.
  - **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);

- 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.
- “From: andrea@sterbini.org” deve fornire il tipo “**From:**” ed il valore “**andrea@sterbini.org**”.

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

- Deve estrarre dalla stringa **testo** (terminata dal carattere '\0') la prima parola e copiarla nella stringa **parola** di dimensioni **dim** caratteri.
  - **parola** dev'essere terminata dal carattere '\0'.
- Come si riconosce 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 una stringa contenente una email, la analizza e inserisce nel vettore tokenArray di dimensione dimT i token ricavati dallo header e dal corpo della email.
  - typedef struct a_token {
    char * parola;
    char * contesto; // NULL = body
    } Token;
- I token vanno inseriti nell'ordine in cui appaiono nella email.

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

- Stampa su stdout (una per ciascuna riga e nell'ordine di apparizione) le stringhe "**tipo*parola**" ottenute dai Token in tokenArray.
- Per i token estratti dal corpo del messaggio (quelli con tipo == NULL) va stampata, una per riga, solo la **parola**.

Esempio di stampa dei tokens

- 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

Funzioni di test

- Le funzioni di test debbono essere contenute tutte in "testModulo1.c".
- I prototipi sono in "testModulo1.h" assieme ai codici risultato che debbono tornare.
- 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);

Argomenti delle funzioni di test

- Le funzioni si comportano tutte nello stesso modo; ricevono i due parametri interi:
  - tipo_di_test: è il codice risultato che ci aspettiamo come dalla funzione da testare.
  - n_argomento: indica su quale argomento facciamo il test, partendo da 0 per il primo.

Risultato delle funzioni di test

- TEST_PASSED 0
  - il test è passato, 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

Stampa della spiegazione

- 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:
  - 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

Esempio di test

- Una funzione di test di nome test_XXX che riceva la coppia di valori:
  - tipo-di-test: NULL_POINTER
  - Argomento: 0
- Cerca di scoprire se XXX torna NULL_POINTER se gli si passa NULL come prino argomento
  - Se la funzione XXX torna NULL_POINTER il test è passato e test_XXX torna TEST_PASSED

Esempio di stampa in caso di errore

- Altrimenti test_XXX torna TEST_FAILED e stampa:
  - 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.

Test non validi o non implementati

- Se la funzione XXX 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**

FAQ (da twiki)

- Q: Una funzione potrebbe fallire per più errori contemporanei, quale codice di errore dobbiamo tornare?
  - Esaminate gli **argomenti nell'ordine**
  - Per ciascun argomento esaminate le condizioni di errore **nell'ordine in cui sono definiti i codici di errore**
- Q: cosa intende per stringa vuota?
  - Una stringa vuota è un (puntatore a) un vettore di 1 carattere contenente il solo carattere '\0'

FAQ (segue)

- Q: chi alloca i buffer in cui copiamo le risposte (headers, parole ...), noi nella funzione XXX o lei nel suo main?
  - Se ne occupa ciascuna funzione di test, quindi la risposta è: “sia io nei miei test che voi nei vostri”
- Q: come trovo dove inizia il secondo header?
  - Basta sapere dove finisce il primo