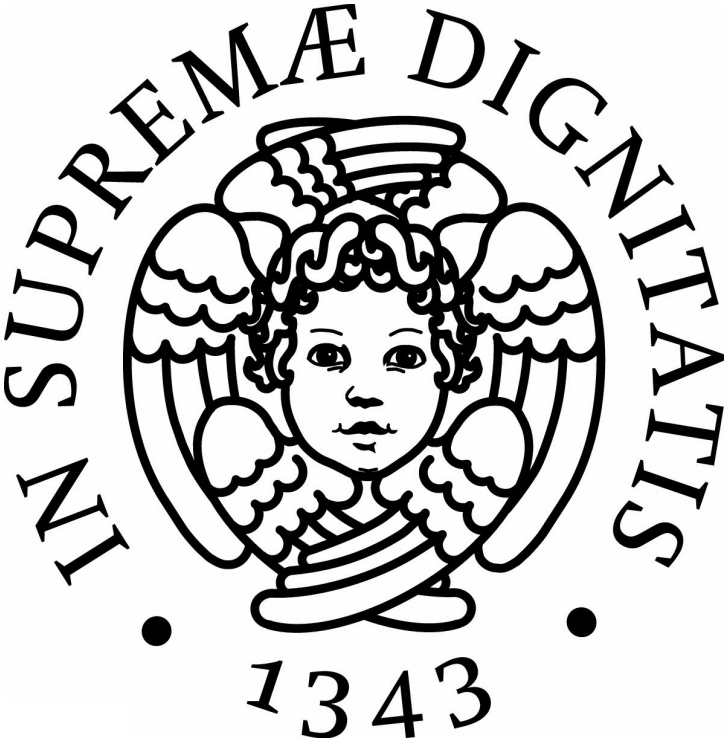

Università di Pisa
Facoltà di Scienze Matematiche Fisiche e Naturali
Corso di Laurea in Informatica

BBCode

Progetto finale di Laboratorio di Linguaggi di Sistema
a.a 2010/2011

Filippo Fontanelli

Prof. Vincenzo Gervasi

1. Introduzione

Il progetto svolto si occupa della realizzazione di un componente di sistema che permetta di leggere dei testi, la cui forma e' descritta tramite una serie di comandi BBCode, e di ottenere il medesimo testo in output in vari tipi di formati(ASCII, ANSI,..).

2. Organizzazione

Il componente è costituito da un unico comando denominato bbmft che accetterà da riga di comando i parametri che identificano il formato dell'output.

3. Principali scelte del progetto

Le scelte più rilevanti sono state:

a. Le strutture dati necessarie per l'implementazione:

Per la realizzazione ho deciso di avvalermi di una coda di token, dove il token rappresenta porzioni di testo con una formattazione specifica.

I token racchiudono in essi altre informazioni oltre alla semplice stringa, in particolare:

- *La matrice di formattazione*, relativa alla porzione di testo selezionata;
- *Il numero di spazi* , il quale potrà esser usato come parametro per la giustificazione del testo;
- *Link*, utilizzato per la memorizzazione dei link sia per le immagini che per l'hyperlink

La Coda contiene i riferimenti alla testa e alla coda, e altre informazioni di stato, come la lunghezza totale del testo dei token e il numero di elementi della coda.

b. Come ottenere la genericità e manutenibilità del componete

In particolare ho cercato di rendere il codice il più "flessibile" al variare dei parametri espressi da riga di comando.
Ciò è stato ottenuto in parte tramite la creazione di appositi file sorgente ed header per ogni tipo di formattazione di output possibile e tramite l'utilizzo di puntatori a funzione e typedef.
Per un ulteriore chiarimento prendendo in analisi la formattazione ASCII, possiamo notare che per l'integrazione con il main, e quindi con il comando bbfmt ,è caratterizzata da 2 funzioni cosi dichiarate nel file Ascii.h

```
/** Determina la formattazione in base alle necessita' della formattazione
ascii
 *  \param  t     token da analizzare
 *  \param  q     coda del token
 *  \param  pch   testo da analizzare
 *
 *  \retval 0       successo
 *  \retval ERROR in caso di errore
 */
```

```
extern int CheckAscii(token *t,queue *q, char *pch);

/** Determina la formattazione in base alle necessita' della formattazione ansi
 *  \param  q       coda del token
 *  \param  endline numero di caratteri gia' stampati
 *
 *  \retval n       numero di caratteri stampati
 */
extern int PrintAscii(queue *q, int *endline);
```

Le quali vengono definite nel main.h:

```
int CheckAscii(token *t,queue *q, char *pch);

int PrintAscii(queue *q, int *endline);
```

Procedendo con la definizione dei puntatori a funzione:

```
typedef int (*checkModeCallback)(token *t, queue *q, char *pch);

typedef int (*printModeCallback)(queue *q, int *endline);
```

E infine definendo:

```
checkModeCallback CheckFunc[O_MODE_MAX] =
{
     CheckAscii,
    CheckAnsi,
    CheckDefault
};
```

Tale strutturazione mi ha permesso di decidere a run time quale implementazione utilizzare in base ai parametri espressi.

In oltre per l'opzione -7 è stata prevista un'apposita funzione.

In mancanza di parametri il comportamento di default è applicato da una terna di funzioni appositamente definite.

Per quanto riguarda invece la matrice di formattazione essa è formata da due colonne:

- Una relativa alla formattazione, la quale identifica la presenza o meno di tale formattazione.
- Una relativa alla stampa, la quale identifica se la relativa formattazione e' stata stampata.

ciò si è reso strettamente necessario per la formattazione ASCII in particolare per eseguire un controllo su eventuali formattazioni BBCode errate.

c. L'implementazione dei formati di output richiesti

i. Formato ASCII

Tale tipo di output è stato ottenuto procedendo con l'inserimento dei caratteri di formattazione contestualmente con l'individuazione della formattazione BBCode associati(utilizzando la matrice di formattazione).
Tale tipo d'implementazione si è rivelato poco adatta per le formattazioni di quote e code.

ii. Formato ANSI

Lo sviluppo di tale formattazione è stato proceduto da un accurato studio del medesimo formato e delle sue caratteristiche.
Esso è stato realizzato in due fasi:

i. La prima caratterizzata dall'individuazione delle formattazioni BBCode.
ii. La seconda caratterizzata dall'analisi delle informazioni raccolte al passo precedente e dalla stampa su stdout del testo correttamente formattato, dove la formattazione è stata determinata tramite la concatenazione di caratteri appartenenti ai codici ANSI.

d. La gestione dei parametri in ingresso da riga di comando

Tale funzionalità è stata realizzata mediante la funzione *getop*. Non ho previsto particolari controlli, tranne l'impossibilità di esprimere più di un parametro riguardante la formattazione di output, in particolare il comando non permette di esprimere *–a* e *–x* contemporaneamente.

4. Struttura del codice

Ho cercato di rendere il codice più chiaro possibile andando ad accumulare parti simili di codice e di suddividere il codice il moduli andando ad ottenere diversi file sorgente e header:

def.h: *contiene le definizioni principali del comando, e dei vari tipi di formattazione.*

main.c/.h: *realizzazione del ciclo di vita del comando, all'interno del quale vengono chiamate le funzioni di default o quelle relativi alla formattazione di output.*

queue.c/.h: *realizza la coda di token in cui troviamo le dichiarazioni delle principali funzioni di manipolazione/interrogazione delle strutture dati.*

Ansi.c/.h: *realizzazione di un package di funzioni necessarie per tale formattazione e per l'integrazione con comando realizzato.*

Ascii.c/.h: *realizzazione di un package di funzioni necessarie per tale formattazione e per l'integrazione con comando realizzato.*

5. Struttura dei programmi realizzati

E’ stato sviluppato un unico programma, ma la sua realizzazione si è sviluppata in modo sincrono con la realizzazione della funzione dei file Ansi e Ascii.
Per prima cosa ho cercato di ampliare la mia conoscenza per quanto riguarda ASCII e ANSI, ciò si è rivelato molto utile per iniziare a disegnare il problema nella sua complessità.

In seguito ho provveduto alla realizzazione delle strutture dati principali vedi coda e token, sulle quali ho basato il resto del mio lavoro.
La scelta di utilizzare tale tipo si struttura dati è nata da alcune considerazioni:

i. Potevamo salvare l’input in un file e successivamente muoverci in esso, ma ho ritenuto tale soluzione poco elegante dato che il testo in input potrebbe avere dimensioni rilevanti.
ii. Per le stesse motivazioni e per evitare problemi relativi all’allocazione di memoria ho ritenuto non auspicabile un salvataggio in memoria dell’intero testo in input.

Perciò raggiungendo un compromesso che ritengo accettabile, ho cercato di suddividere il lavoro in base ad alcuni criteri:

- Dal testo in input sono letti un numero di caratteri all’incirca parai alla larghezza della colonna.
- Per ogni step di lettura si ha la creazione di una coda con i relativi token.
- Ed infine dopo un processo di giustificazione il testo contenuto nella coda verrà stampata su stdout.
- Tale operazione è ripetuta fino alla fine del testo in input.

In seguito ho provveduto a occuparmi della formattazione ANSI, la quale non ha presentato particolari problematiche.

Infine mi sono concentrato sulla formattazione ASCII, la quale a sua volta ha richiesto alcuni accorgimenti come specificato in precedenza.

Il punto cruciale è stato la realizzazione della trasformazione da caratteri non ASCII a caratteri ASCII la quale si è rivelata problematica per differenti motivi:

- In base alle mie ricerche il modo più naturale avrebbe previsto una conversione ad Unicode ed una successiva riconversione tramite una matrice , oppure tramite l’utilizzo dei wchar_t e dalla relativa libreria.
- Purtroppo per mancanza di tempo e informazioni a riguardo non sono riuscito a implementare tale funzionalità.

6. Conclusioni

Le chiare specifiche iniziali nascondano la complessità di tutto il progetto.
Le caratteristiche di ogni singola formattazione, l’utilizzo di strutture dati apposite e gli innumerevoli casi da considerare discostano il progetto da una semplice banalità’. Ho cercato di realizzare il componente in tutti i suoi particolari, ma purtroppo alcuni di essi si sono rilevati di particolare difficoltà.