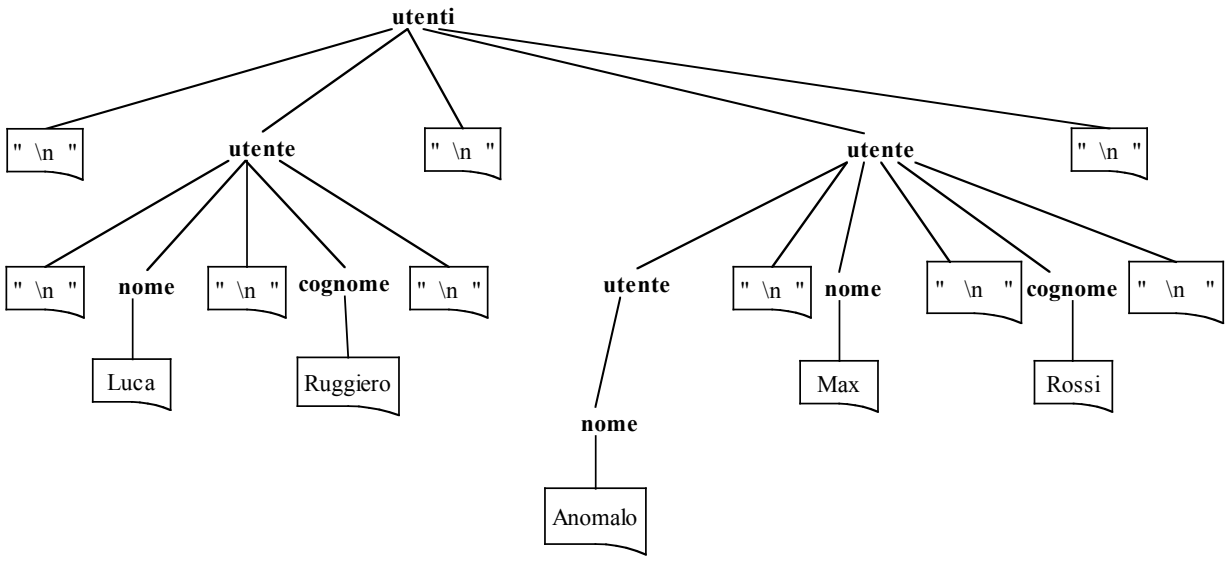
Laboratorio di Programmazione (A.A. 2006/07)
Modulo 4: xsMLQuery

L'obiettivo del quarto modulo è quello di implementare le funzioni per leggere, analizzare ed eseguire interrogazioni scritte nel mini-linguaggio xsMLQuery (si veda avanti).

Linguaggio xsMLQuery

Le interrogazioni del linguaggio xsMLQuery, fornite sotto forma di stringa, specificano il percorso da seguire per identificare i nodi da restituire. Ad esempio, dato il seguente file xsML (rappresentato già sotto forma di albero):



l'interrogazione:

/utenti/utente

deve restituire una lista dei due figli del tag <utenti> aventi nome "utente". Ancora, l'interrogazione /utenti/utente/nome deve restituire i due nodi aventi come figli i nodi di testo Luca e Max.

Se non si specifica lo slash iniziale /, la ricerca non deve necessariamente partire dalla radice dell'albero. Ad esempio, se si cerca:

utente/nome

si devono trovare tutte le istanze di tag <nome> all'interno dell'albero aventi come padre un tag <utente> (tre nell'esempio: quello con figlio Luca, quello con figlio Max e quello con figlio Anomalo).

Infine il linguaggio xsMLQuery permette di specificare dei vincoli sugli attributi dei nodi che compongono l'interrogazione. I vincoli sugli attributi di un nodo sono specificati tra parentesi quadre

subito dopo il nome del nodo. Ad esempio:

/utenti/utente[@lingua=”italiano”]/nome[@id=”5”]

richiede la lista di tag <nome> aventi id = “5” e aventi come padre <utente> (con attributo lingua=”italiano”) e come padre di <utente> il tag <utenti>. Notiamo che i nomi degli attributi sono sempre preceduti dal simbolo @. Se vengono specificati più vincoli per uno stesso nodo questi saranno separati da virgola. Ad esempio:

/utenti/utente[@lingua=”italiano”,@nazione=”svizzera”]/nome[@id=”5”]

richiede la lista di tag <nome> aventi id = “5” e aventi come padre <utente> (con attributi lingua=”italiano” **e** nazione=”svizzera”) e come padre di <utente> il tag <utenti>.

E’ possibile anche porre come condizione il solo nome dell’attributo, a indicare che si richiede che quel nodo un attributo con quel nome (senza imporre vincoli sul suo valore). Ad esempio:

/utenti/utente[@lingua]/nome[@id=”5”]

richiede la lista di tag <nome> aventi id = “5” e aventi come padre <utente> (con attributo lingua) e come padre di <utente> il tag <utenti>.

Parsing delle query

La prima funzione da scrivere è la

```
int parse_query(char *str, list *qptr, int *rptr);
```

che riceve in `str` la stringa della query da analizzare e rappresenta sotto forma di lista di nodi, restituita nella locazione puntata da `qptr`, la sequenza di nomi di tag della query. Inoltre, la funzione restituisce nella locazione puntata da `rptr` un valore diverso da 0, se la query deve partire dalla radice del documento, o il valore 0 altrimenti.
Tutti i nodi della lista ottenuta dalla trasformazione della query saranno di tipo tag. L’eventuale lista di vincoli sui valori degli attributi di un dato nodo nella query sarà contenuta nella lista degli attributi del corrispondente nodo. Il campo della lista dei figli sarà indefinito per tutti i nodi. Ad esempio, la query

/utenti/utente[@lingua=”italiano”,@nazione=”svizzera”]/nome[@id=”5”]

sarà trasformata in una lista di tre nodi di tipo tag di nome, nell’ordime: utenti, utente e nome. Non essendoci vincoli sui valori degli attributi del primo nodo, la lista dei suoi attributi nella lista rappresentante la query sarà vuota. Invece, la lista degli attributi del secondo nodo conterrà due coppie attributo-valore (lingua=”italiano” e nazione=”svizzera”) e la lista del terzo nodo una sola coppia (id=”5”). Infine, per rappresentare il caso in cui nella query è richiesta la presenza di un dato attributo senza specificarne il valore, nella lista della query si assocerà il valore NULL al puntatore alla stringa del corrispondente attributo.

La funzione `parse_query` restituisce uno dei seguenti codici di errore:

```
typedef enum
{
      PARSE_QUERY_OK = 0,
      PARSE_QUERY_SYNTAX_ERROR,
      PARSE_QUERY_ERROR,
} parse_query_code;
```

rispettivamente con i seguenti significati:

codice	**Significato**
PARSE_QUERY_OK	Nel caso in cui la query sia sintatticamente corretta e la funzione abbia potuto portare a termine senza errori la trasformazione nella lista.
PARSE_QUERY_SYNTAX_ERROR	La query in ingresso contiene un errore di sintassi (es. manca la @ davanti al nome dell'attributo)
PARSE_QUERY_ERROR	Si è verificato un errore di esecuzione (es. non è stato possibile allocare della memoria)

Esecuzione della query

La seconda funzione da scrivere è la

```
int query(struct node *root, list q, int rooted, list *risultato);
```

che riceve in `root` il puntatore alla radice di un albero xsML, in `q` la lista di nodi che rappresenta una query, in `rooted` un valore diverso da 0 se la query parte dalla radice o un valore 0 altrimenti, e restituisce in `risultato` la lista dei nodi che soddisfano la query.

La funzione query restituisce uno dei seguenti codici di errore:

```
typedef enum
{
      QUERY_OK = 0,
      QUERY_FAIL,
      QUERY_ERROR,
} query_code;
```

rispettivamente con i seguenti significati:

codice	**Significato**
QUERY_OK	Almeno un nodo soddisfa l'interrogazione
QUERY_FAIL	Nessun nodo soddisfa l'interrogazione
QUERY_ ERROR	Si è verificato un errore di esecuzione o uno dei parametri ricevuti non era in un formato valido