PROGRAMMAZIONE II - Anno Accademico 2013-14 - Progetto Sessione Estiva – V.2

Il progetto ha l'obiettivo di applicare i concetti e le tecniche di programmazione esaminate durante il corso a casi specifici. Il progetto consiste nella progettazione e realizzazione di alcuni moduli software, ed è strutturato in due esercizi di programmazione.

Esercizio 1: Progettazione e sviluppo di un interprete in OCaml

Si consideri un semplice linguaggio funzionale per la manipolazione di liste di interi, la cui sintassi conocreta è definita dalla seguente grammatica:

(Identificatori)	Ide ::= <definizione standard>	
(Interi)	Int ::= <definizione standard>	
(Liste di interi)	intList ::= nil | cons(Int, intList)	
(Valori)	Val :: = Int	(Interi)
	| intList	(Liste di interi)
	| True | False	(Valori Booleani)
	| fun Ide -> E	(Funzioni con un solo parametro, non ricorsive)
(Espressioni)	E ::= Ide	(Identificatori)
	| Val	(Valori)
	| E and E | E or E | not E	(Espressioni Booleane)
	| OP(E,E)	(Espressioni su interi con OP ∈{"+", "-", "*", "=", "<="})
	| E <= E	(Operatore di confronto tra intList: "sottolista iniziale")
	| E == E	(Operatore di uguaglianza per intList)
	| isEmpty(E)	(Operatore logico su intList – controlla se la lista E è vuota)
	| E @ E	(Append di intList)
	| if E then E else E	(Condizionale)
	| let Ide = E in E	(Blocco let)
	| E(E)	(Applicazione Funzionale)
	| map E on E	(map E on E': Applica la funzione E a tutti gli elementi della lista E')

Gli operatori hanno il significato descritto tra parentesi, e in generale sono parziali: per esempio, '<=', '==' e '@' non sono definiti se gli argomenti non sono di tipo intList, 'map' può essere applicato solo a una funzione e a una intList, eccetera. Il confronto 'E1 <= E2' restituisce true se E1 è un prefisso di E2, cioè esiste una lista di interi E tale che E2 = E1 @ E.

1. Definire una sintassi astratta per il linguaggio, introducendo opportuni tipi di dati OCaml.
2. Definire in OCaml un interprete del linguaggio assumendo la regola di scoping statico.
3. Verificare la correttezza dell'interprete progettando ed eseguendo una quantità di casi di test sufficiente a testare tutti gli operatori.
4. Descrivere quali sono le modifiche da apportare all'interprete del linguaggio nel caso si voglia assumere una regola di scoping dinamico.

Esercizio 2: Progettazione e realizzazione di un modulo Java

Lo scopo del progetto è lo sviluppo di una componente software di supporto alla gestione di un semplice blog. Da un punto di vista astratto un blog può essere visto come una collezione di contenuti che vengono visualizzati in forma cronologica. Per semplicità assumiamo che i contenuti siano solamente in forma testuale e siano rappresentati da stringhe. Assumiamo, inoltre, che il blog sia gestito da un insieme di blogger. Solamente i blogger registrati possono inserire post nel blog mentre la visualizzazione del blog è aperta a tutti.

La struttura astratta del Blog è caratterizzata dalla Java interface definita di seguito.

```
interface SimpleBlog {
void addBlogger(Blogger bob, String pass) throws UnauthorizedAccessException;
        // Aggiunge il blogger bob all'insieme dei blogger attivi sul blog. Solleva una eccezione se pass non è corretta
void deleteBlogger(Blogger bob, String pass) throws UnauthorizedAccessException;
        // Elimina il blogger bob dall'insieme dei blogger attivi sul blog. Solleva una eccezione se pass non è corretta
int post(String message, Blogger bob) throws UnauthorizedBloggerException;
        // Inserisce nel blog un post del blogger bob. Restituisce un codice numerico del post.
        // Lancia un'eccezione se bob non è tra i blogger attivi sul blog
String readLast(Blogger bob) throws EmptyPostException;
        // Restituisce l'ultimo post inserito dal blogger bob. Solleva un'eccezione se non ci sono post di bob
String readLast() throws EmptyPostException;
        // Restituisce l'ultimo post inserito nel blog. Solleva un'eccezione se il blog è vuoto
List<String> readAll(Blogger bob);
        // Restituisce tutti i post inseriti dal blogger bob, nell'ordine di inserimento
List<String> readAll();
        // Restituisce tutti i post inseriti, nell'ordine di inserimento
void delete(int code) throws WrongCodePostException;
        // Cancella dal blog il post identificato da code. Solleva un'eccezione se non esiste un post con quel codice.
boolean emptyBlog();
        // Restituisce true se e solo se il blog è vuoto
}
```

1. Definire la classe **Blogger** (con un'unica variabile d'istanza **String nickname**) e le classi di eccezioni usate nell'interfaccia, in modo da compilare l'interfaccia con successo. Il tipo generico **List<E>** è quello fornito da Java.
2. Definire un'implementazione in Java dell'interface **SimpleBlog**, chiamata **Blog**. Il costruttore di **Blog** ha un argomento di tipo **String**, che serve per fornire una password (non modificabile) quando si crea un oggetto della classe. Non sono posti vincoli sulla struttura di implementazione prescelta.
   Attenzione: i metodi **read()** e **readAll()** restituiscono i post come stringhe nel formato **"nickname: message"**, dove il **nickname** è quello del blogger che ha fatto il post del **message**.
3. Descrivere la funzione di astrazione e l'invariante di rappresentazione. Realizzare una batteria di test per valutare il comportamento dell'implementazione proposta.

Modalità di consegna Progetto Sessione Estiva

- Il progetto deve essere svolto e discusso col docente individualmente. Il confronto con colleghi mirante a valutare soluzioni alternative durante la fase di progetto è incoraggiato,
- Per poter sostenere l'orale di un appello, il progetto deve essere consegnato entro il giorno dello scritto dello stesso appello, quindi entro (le ore 24 di) Giovedì 12 Giugno per il primo appello, e entro Giovedì 10 Luglio per il secondo appello. Un progetto sottomesso per il primo appello vale anche per il secondo se lo studente non supera la prova scritta o quella orale. Il progetto vale comunque solo per la sessione estiva.
- Il progetto deve essere costituito da:
    - I file sorgente contenenti il codice sviluppato e le corrispondenti batterie di test; tutto il codice deve essere adeguatamente commentato nei file sorgente.
    - Una relazione di massimo una pagina per esercizio, che descrive le principali scelte progettuali ed eventuali istruzioni per eseguire il codice.
- La consegna va fatta inviando per email tutti i file in un archivio. Per il corso A, inviare l'email al Prof. Ferrari con oggetto "[PR2A] Consegna progetto". Per il corso B, inviare l'email al Prof. Corradini con oggetto contenente la stringa "[PR2B] Consegna progetto".

Altre informazioni

- Per quanto riguarda il progetto, i docenti risponderanno solo a eventuali domande riguardanti l'interpretazione del testo, e non valuteranno o commenteranno soluzioni parziali prima della consegna.
- Per eseguire nell'interprete OCaml il codice contenuto nel file **prova.ml**, si può usare il comando dell'interprete **#use "prova.ml"** (il carattere '#' è necessario).