PROGRAMMAZIONE II - Anno Accademico 2013-14 - Progetto Sessione Invernale

Andrea Corradini & GianLuigi Ferrari

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 in cui un programma è costituito da una lista di dichiarazioni di funzioni (anche ricorsive) con un solo parametro, seguita da un'espressione. La sintassi concreta è data dalla seguente grammatica BNF:

(Programma)	**Program** Decl **=>** Exp	
(Dichiarazioni)	Decl ::= ε | **funz** Ide **(** Ide **) {** Exp **} ;** Decl	(NB: ε rappresenta la stringa vuota)
(Identificatori)	Ide ::= <definizione standard>	
(Interi)	Int ::= <definizione standard>	
(booleani)	Bool ::= <definizione standard>	
(Operatori)	Op ::= **+** | **-** | ***** | **=** | **<=**	
(Espressioni)	Exp ::= Ide | Int | Bool	
	| Exp **and** Exp | Exp **or** Exp | **not** Exp	(Espressioni Booleane)
	| Op **(** Exp **,** Exp **)**	(Espressioni su interi)
	| **if** Exp **then** Exp **else** Exp	(Condizionale)
	| Ide **(** Exp **)**	(Applicazione Funzionale)

Quindi una dichiarazione di funzione ha la forma **funz f (x) { expr }** dove **f** è il nome della funzione, **x** il parametro formale, ed **expr** un'espressione dove **x** può comparire libera. Si noti che, come per esempio in C, le dichiarazioni di funzioni possono essere sono al "top-level" e definiscono quindi un ambiente globale in cui deve essere valutata l'espressione finale del programma.

1. Definire una sintassi astratta per il linguaggio, introducendo opportuni tipi di dati OCaml.
2. Definire in OCaml un interprete del linguaggio che rispetti le seguenti specifiche:
   a. L'applicazione funzionale deve essere valutata con riferimento all'ambiente globale determinato dalle dichiarazioni.
   b. Tutte le altre operazioni hanno il significato visto a lezione.
3. Verificare la correttezza dell'interprete progettando ed eseguendo una quantità di casi di test sufficiente a testare tutti gli operatori.
4. Tradurre nella sintassi astratta proposta il seguente programma, e valutarlo con l'interprete: il risultato deve essere **13**.

```
Program  funz sub1 (n) { -(n,1)};
funz fib (n) { if =(n,0) or =(n,1) then n
                 else +( fib (sub1(n)), fib (sub2(n))) } ;
funz sub2 (m) { sub1(sub1(m)) }
=> fib (7)
```

5. Per quanto non specificato documentare e motivare le scelte fatte nella relazione.

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 una semplice agenda di eventi. Da un punto di vista astratto una agenda può essere vista come una collezione di eventi che vengono visualizzati in forma cronologica. Per semplicità, assumiamo che gli eventi siano oggetti di una opportuna classe di tipo record chiamata **Event**, definita come segue:

```
import java.util.Date;
public class Event {
      private String info;
      private Date inizio, fine;
      public Event (String info, Date inizio, Date fine){
            this.info = info;
            this.inizio = inizio;
            this.fine = fine;
      }
      public String getInfo(){ return info; }
      public Date getInizio(){ return inizio; }
      public Date getFine (){ return fine; }
}
```

La struttura astratta dell'agenda è caratterizzata dall'interfaccia Java definita di seguito.

```
import java.util.*;
interface Agenda {
/* Aggiunge l'evento all'agenda. Solleva un'eccezione se nell'agenda c'e' un
evento che si sovrappone temporalmente ad event. */
void addEvent(Event event) throws EventConflictException;
/* Elimina dall'agenda tutti gli eventi terminati prima di date */
void deleteOldEvents(Date date);
/* Restituisce l'evento attivo in date, se esiste, null altrimenti. */
Event readEvent(Date date);
/* Restituisce gli eventi che inizieranno dopo date, in ordine temporale. */
List<Event> readFutureEvents(Date date);
/* Restituisce true se e solo se l'agenda è vuota */
boolean emptyAgenda();
}
```

1. Aggiungere alla classe **Event** una overview che descrive eventuali proprietà che gli oggetti della classe devono rispettare, e modificare il costruttore in modo che lanci opportune eccezioni se tali proprietà non sono rispettate.
2. Completare la specifica del Tipo di Dati Astratti definito dall'interfaccia **Agenda**, aggiungendo un'overview e la specifica di ogni metodo.
3. Implementare l'interfaccia **Agenda** con la classe **SimpleAgenda**, in cui gli eventi sono memorizzati in una lista in ordine temporale crescente.
4. Definire l'invariante di rappresentazione e la funzione di astrazione per la classe **SimpleAgenda**.
5. Realizzare una batteria di test per valutare il comportamento dell'implementazione proposta.

Modalità di consegna Progetto Sessione Invernale

- 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 l'appello di gennaio il progetto deve essere consegnato entro le ore 24 di **Venerdì 23 gennaio 2015**: l'orare verrà fissato nella settimana successiva. Si può chiedere di sostenere l'orale nella settimana del 19 gennaio consegnando il progetto prima.
- Per l'appello di febbraio il progetto deve essere consegnato entro le ore 24 di **Giovedì 12 febbraio 2015**.
- 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 invernale.
- 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).