Progetto di Linguaggi e Compilatori 1
A.A. 2014-15

Gruppo 0

Attenzione: una volta letto il presente testo è *obbligatorio* consegnare alla scadenza quanto elaborato, indipendentemente dal fatto se lo si consideri adeguato o meno.

Ogni gruppo deve inviare via email entro il termine stabilito la relazione in formato PDF (dal nome "`ProgettoLC1 Gruppo 0 Relazione`") assieme a tutti i sorgenti sviluppati (in modo che sia possibile verificare il funzionamento di quanto realizzato) all'interno di un file compresso, il cui nome sia "`ProgettoLC1 Gruppo 0`". In tale file compresso *non* devono comparire file oggetto e/o binari. La relazione può contenere immagini passate a scanner di grafici o figure (oppure si può concordare, al momento del ritiro, per una consegna del cartaceo delle figure senza doverle includere nel file PDF). Si richiede:

Per tutti gli esercizi descrizione dettagliata di tutte le tecniche non-standard impiegate (le tecniche standard imparate a lezione non vanno descritte).

Per tutti gli esercizi descrizione delle assunzioni fatte riguardo alla specifica, sia relativamente a scelte non previste espressamente dalla specifica stessa, che a scelte in contrasto a quanto previsto (con relative motivazioni).

Per gli esercizi 5 e 6 descrizione sintetica generale della soluzione realizzata.

Per gli esercizi 5 e 6 commentare (molto) sinteticamente ma adeguatamente il codice prodotto.

Per gli esercizi 4, 5 e 6 fornire opportuni Makefile (compliant rispetto allo standard GNU make) per

- poter generare automaticamente dai sorgenti i files necessari all'esecuzione (lanciando `make`);
- far automaticamente una demo (lanciando `make demo`).

Per l'esercizio 3 aggiungere un file di testo con alcune query di test (da poter lanciare a terminale). Oppure fornire un Makefile come al punto precedente.

L'esercizio 5 *deve* essere implementato in C (o C++ o Java).

Gli esercizi 3, 4 e 6 *devono* essere implementati in Haskell.

1. Si considerino stringhe composte dal cognome e nome dei candidati, oltre che Comini Marco e Fabio Rossi, separati da virgola e spazi.
   (a) Si scriva un'espressione regolare e, con la sintassi di (f/a)lex, per riconoscere sequenze di tali stringhe, separate da punto e virgola (ad esempio "Comini, Marco; Rossi, Fabio").
   (b) Considerando un alfabeto Σ che includa almeno l'ASCII, si determini l'automa riconoscitore minimo per e (automa deterministico che non deve accettare tutte le sequenze che non sono istanza di e).

2. Si rappresenti il P-code relativo al seguente frammento di codice Pascal.

```
(* ONLINE *)
program esercizio;
  const IMAX = 9 ; UNKNOWN = ...;
  type harray = array[ 0..IMAX, 0..IMAX ] of integer; history = ^harray;
  ... ...
  var target, aim: integer;
      mainharray : harray;

  procedure init( i, j: integer; h: history; var z:integer );
    var u: integer;
  begin
      for u := 0 to min(IMAX, j) do
        begin
          p(u*i, z);
          h^[i,u] := z
        end
  end;

  function f( n: integer ) : integer;
```

```
  begin  ...   ...  end;

  procedure p( x: integer; var y: integer );
  begin
    y := f(f(x));
    while (y = 0) and (x > 0) do
      p(y,y)
  end;

  function min(x,y:integer): integer;
  begin  ...   ...  end;

begin
  ...;
  p(target,aim);
  init( 20, 30, @mainharray, target );
  ...
  end.
```

3. Si implementino matrici $2^n \times 2^n$ utilizzando opportunamente QuadTrees "propri" (i cui nodi non hanno figli con colori omogenei).

   ```
   data Matrix a = Mat { nexp :: Int, mat :: QT a }

   data QT a = C a | Q (QT a) (QT a) (QT a) (QT a)
   ```

   Si scriva una funzione `colaltsums` che, data una matrice implementata con la struttura poc'anzi definita, calcola il vettore delle somme a segni alternati delle colonne della matrice. Detto $s_j = \sum_{i=1}^{n}(-1)^{i+1}a_{ij}$, `colaltsums`$\left(\begin{pmatrix} a_{11} & \dots & a_{1m} \\ \vdots & & \vdots \\ a_{n1} & \dots & a_{nm} \end{pmatrix}\right) = \begin{pmatrix} s_1 & \dots & s_m \end{pmatrix}$

4. Si consideri una rappresentazione in sintassi concreta per alberi con un numero arbitrario di figli dove un albero non vuoto viene descritto con il valore in radice e, se non è un nodo foglia, immediatamente dopo abbiamo la sequenza dei figli delimitata con `<` e `>` usando come separatore `,`.
   (a) Per detti alberi, si progetti un opportuna sintassi astratta *polimorfa* e si realizzino due parsers (con relativo lexer) per alberi di numeri interi e alberi di numeri in virgola mobile, che utilizzino entrambi detta sintassi astratta.
   (b) Considerando il caso in cui $\Sigma = \{$`<`, `>`, `,`, `a`$\}$ e detta G la grammatica implementata nel parser, si determini $\mathcal{L}(G) \cap \{$`a<`w`>` $\mid w \in \Sigma^*, w = w^R\}$.
   (c) Si scriva una funzione `annotate` che, dato un albero codificato nella sintassi astratta proposta, costruisca un nuovo albero che in ogni nodo contenga, al posto del valore originale, una coppia composta dal medesimo valore e dall'altezza del nodo stesso.
   (d) Si combini quindi il parser per numeri interi con `annotate` in una funzione `test` per cui preparare dei test case significativi.

5. Si consideri un file di configurazione "alla Windows" composto da varie sezioni, identificate da una etichetta tra parentesi quadre; ogni sezione contiene assegnamenti di valori a variabili. Etichette e variabili sono stringhe alfanumeriche (inizianti con un carattere), mentre i valori possono essere numeri interi, booleani, o stringhe tra doppi apici. Commenti (da ignorarsi) da "#" a fine linea. Un esempio è il seguente:

   ```
   [nomesez1]
   # questo e' un commento
   var1=3;
   var2="non dire gatto";

   [nomesez2]
   miao=true;
   var1=false;
   ```

Si definisca la grammatica di tale sintassi.

Si implementi un parser che dato un file di configurazione valido, costruisca una struttura dati (ad esempio mediante liste etichetta-valore) contenente i valori associati ad ogni variabile. La definizione della stessa variabile in sezioni diverse è legittima, mentre la ridefinizione nella stessa sezione deve essere segnalata come warning.

In caso di ripetizione di una sezione invece deve essere segnalato errore.

Si implementi inoltre un "pretty-printer" che, avuta in input la struttura restituita dal parser, riscriva il file di configurazione con i valori aggiornati, mantenedo gli eventuali commenti presenti.

Si cerchi di fornire messaggi di errore che aiutino il più possibile a capire in cosa consiste l'errore, quali entità coinvolge e dove queste si collochino nel sorgente.

6. Si consideri un linguaggio costruito a *partire* dalla stessa *sintassi concreta* di **C**.

   In tale linguaggio (che non è C) le procedure son funzioni di tipo `void`. Si possono dichiarare funzioni, oltre che variabili, all'interno di qualsiasi blocco.

   Si hanno le operazioni aritmetiche, booleane e relazionali standard ed i tipi ammessi siano interi, booleani, real, caratteri, stringhe, array e puntatori (con i costruttori e/o le operazioni di selezione relativi).

   Non è necessario avere tipi di dato definiti dall'utente.

   Si hanno inoltre le funzioni predefinite `writeInt`, `writeFloat`, `writeChar`, `writeString`, `readInt`, `readFloat`, `readChar`, `readString`.

   Il linguaggio deve implementare—con la sintassi concreta di C qualora sia prevista—le modalità di passaggio dei parametri value e value-result (le altre a piacere), con i relativi vincoli di semantica statica.

   Il linguaggio deve ammettere le istruzioni `break` e `continue` (dentro il corpo dei cicli), con la sintassi concreta di C, qualora sia prevista.

   Per detto linguaggio (non necessariamente in ordine sequenziale):

   - Si progetti una opportuna sintassi astratta.
   - Si progetti un type-system (definendo le regole di tipo rispetto alla sintassi astratta generata dal parser, eventualmente semplificandone la rappresentazione senza però snaturarne il contenuto semantico).
   - Si implementi un lexer con Alex, un parser con Happy ed il corrispondente pretty printer (si suggerisce di utilizzare BNFC per costruire un primo prototipo iniziale da raffinare poi manualmente, ma non è obbligatorio).
   - Si implementi il type-checker e *tutti* gli altri opportuni controlli di semantica statica (ad esempio il rilevamento di utilizzo di r-expr illegali nel passaggio dei parametri). Si cerchi di fornire messaggi di errore che aiutino il più possibile a capire in cosa consiste l'errore, quali entità coinvolge e dove queste si collochino nel sorgente.

   Si predispongano dei test case significativi per una funzione che, dato un nome di file, esegua il parsing del contenuto, l'analisi di semantica statica e il pretty-print.

   Si tenga presente che del linguaggio C, da cui si trae ispirazione per la sintassi concreta, non si richiede di capire la semantica di funzionamento (irrilevante ai fini della soluzione) ma solo di utilizzare le stesse scelte di sintassi concreta relativamente ai costrutti (canonici) previsti dal testo.