Progetto di Linguaggi e Compilatori – Parte 2
A.A. 2021-22

Gruppo 21-3

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

Entro il termine stabilito si *deve* inviare via email dentro un unico file compresso, il cui nome sia "`ProgettoLC parte2 Gruppo 3`", tutti (e soli) i sorgenti sviluppati, in modo che sia possibile verificare il funzionamento di quanto realizzato. In tale file compresso *non* devono comparire file oggetto e/o binari o qualsiasi altro file che viene generato a partire dai sorgenti. Sempre in detto file va inoltre inclusa una relazione in formato PDF dal nome "`ProgettoLC parte2 Gruppo 3 Relazione`". La relazione può contenere immagini passate a scanner di grafici o figure fatte a mano, il che **non** include fotografie digitali. Evitare scansioni di scritti a matita e assicurarsi comunque che si abbia un buon contrasto.

- Si richiede una descrizione dettagliata di tutte le tecniche **non**-standard impiegate (le tecniche standard imparate a lezione non vanno descritte).
- Si richiede una 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).

  Non è assolutamente utile perdere tempo per includere nella relazione il testo dei vari esercizi o un suo riassunto o una qualsiasi rielaborazione, incluso descrizioni del problema da risolvere. Ci si deve concentrare solo sulla descrizione della soluzione e delle eventuali variazioni rispetto a quanto richiesto.
- Si richiede di 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`).
- La soluzione *deve* essere implementata in C o C++ o Java, utilizzando Bison/Flex o CUP/JLex o strumenti analoghi che generino il sorgente di parser/lexer a partire da grammatica/espressioni regolari. Si suggerisce di utilizzare BNFC per costruire un primo prototipo iniziale da raffinare poi manualmente, ma non è obbligatorio (si includa anche il sorgente .cf in caso).

Esercizio

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, stringhe tra doppi apici o altre variabili (precedute da "`$`" ed un eventuale prefisso "*nomesezione*`.`" per variabili non locali).

Commenti (da ignorarsi) da "#" a fine linea.

Un esempio è il seguente:

```
[nomesez1]
# questo e' un commento
   var1 = 3
   var2 = "non dire gatto"
   var3 = $var1

[nomesez2]
   miao = true
   var1 = false
   var2 = $nomesez1.var1
```

Si definisca la grammatica di tale sintassi.

Si implementi un parser (con relativo lexer) che dato un file di configurazione valido, costruisca una struttura dati (ad esempio mediante una lista di liste che associ sezioni a liste etichetta-valore) contenente i valori associati ad ogni variabile di ogni sezione. 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.

Per la struttura dati scelta si devono implementare, oltre alle funzioni di costruzione necessarie al parser, anche le funzioni di cancellazione di una sezione e di un singolo binding (e opzionalmente funzioni di modifica).

La struttura dell'implementazione può essere scelta fra 2 alternative.

1. Si utilizza una struttura dati più semplice, che contiene solo legami con valori "base" (interi, booleani e stringhe), e si implementa un parser che "risolve" i riferimenti per determinare i valori base da associare.
2. Si utilizza una struttura dati più sofisticata, che mantiene in qualche forma i riferimenti, e conseguentemente si implementa un parser più semplice che non esegue fisicamente nessun rimpiazzamento o copiatura. Per tale struttura dati però
   - deve essere implementata una funzione che data una sezione e un nome restituisca il valore base associato per effetto di una catena di riferimenti;
   - si devono realizzare le cancellazioni evitando di lasciare dei riferimenti inesistenti.

Si implementi inoltre un "pretty-printer" della struttura dati scelta che la serializzi in sintassi corretta per il parser. Quindi invocando il parser su una serializzazione di una struttura, se non ci sono riferimenti, si deve riottenere la stessa struttura, altrimenti dipende dalla scelta implementativa. Questo "pretty-printer" deve inoltre essere realizzato in modo che, qualora

- si invocasse il parser su un file di configurazione e
- si eseguesse una qualsiasi modifica dell'output del parser,

la serializzazione di tale struttura modificata contenga, oltre ai valori attuali della struttura, anche gli eventuali commenti presenti nel file originario. Non è lecito comunque assumere che i valori attuali passati al pretty-printer siano solo quelli ottenuti da un parse. Tali valori devono poter risultare da una qualsiasi elaborazione (che includa anche cancellazioni). Ovviamente (visto che al parser non devono arrivare i commenti in alcuna forma), il pretty-printer dovrà recuparare i commenti da un'opportuna struttura dati salvata dal lexer.

Attenzione: La soluzione deve fare uso solo degli strumenti introdotti nella parte del corso a cui si riferisce l'esercizio.