Progetto di Linguaggi e Compilatori – *PARZIALE* Parte 2
A.A. 2024-25

Gruppo 24-11

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

Si ricordi che questa è una prova d'esame, pertanto **si deve produrre un contributo originale, sviluppato in autonomia**. Si può utilizzare solo il materiale didattico fornito durante il corso ed eventualmente il materiale didattico fornito pubblicamente da altri docenti in altri corsi, compresi libri di testo (a meno che non sia in contraddizione con il materiale del corso). In particolare **NON È PERMESSO** utilizzare (anche solo parti di) soluzioni di progetti realizzati da altri studenti, di qualunque corso di studi, in qualunque anno, di qualsiasi università.

Non è altresì **permesso diffondere il presente testo** in qualsiasi forma (sia fisica che elettronica) nè tantomeno distribuirlo online.

Importante: Si deve concordare il ritiro della versione definitiva del testo entro il tempo a disposizione per l'elaborazione di questa versione parziale. Non verrà consegnato il testo definitivo oltre la scadenza e conseguentemente verrà valutato in sostituzione l'elaborato parziale, sempre che questo sia stato consegnato entro la sua scadenza. Ovviamente questa soluzione parziale verrà considerata come se fosse la soluzione del testo definitivo che sarebbe stato consegnato, quindi potrebbero facilmente mancare parti richieste o esserci parti in contrasto con specifiche non presenti nel testo parziale (subendo la conseguente penalizzazione). Nel caso neanche l'elaborato parziale sia consegnato entro la sua scadenza la Parte 2 (definitiva) verrà valutata 0.

Almeno **5 giorni lavorativi** prima del ritiro del testo definitivo si *deve* inviare via email un unico file compresso in formato ZIP — il cui nome sia "`ProgettoLC parte2parz Gruppo 24-11.zip`" — contenente tutti i **sorgenti** sviluppati. 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 parte2parz Gruppo 24-11 Relazione.pdf`".

Attenzione i nomi di suddetti files **devono** essere esattamente come specificati, carattere per carattere!

La relazione può contenere immagini passate a scanner di grafici o figure fatte a mano, il che **non** include fotografie digitali. Evitare scansioni di pagine quadrettate e di scritti a matita. Assicurarsi comunque che si abbia un buon contrasto. Analogamente si possono inserire grafici/figure fatte con tavolette grafiche o analoghi strumenti digitali, sempre assicurandosi di avere una chiarissima leggibilità del risultato.

- Si richiede una descrizione dettagliata di tutte le tecniche **non**-standard impiegate. Al contrario le tecniche standard e tutto quanto sia stato mostrato a lezione (o disponibile su e-learning) non va descritto, li si deve solo utilizzare correttamente.

  Non si deve 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 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).

Esercizio

Per questo esercizio

- Si richiede una descrizione sintetica generale della soluzione realizzata.
- Si richiede di commentare (molto) sinteticamente ma adeguatamente il codice prodotto.
- Si richiede di fornire opportuno Makefile (conforme 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`) che mostri i test case più significativi.

Progetto di Linguaggi e Compilatori – *PARZIALE* Parte 2
A.A. 2024-25

- La soluzione *deve* essere implementata in Haskell, utilizzando Happy/Alex. Si suggerisce caldamente 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).

- **IMPORTANTE**
  - All'inizio della relazione si scrivano le versioni degli strumenti di generazione usati (BNFC, Happy, Alex e GHC).
  - Si utilizzi GHC $\geq$ 9.2.8 e BNFC $\geq$ 2.9.1.
  - È vietato utilizzare qualunque forma di variabili modificabili, ottenibili mediante librerie come Data.IORef o Data.STRef.

    In caso non si ottemperi a queste direttive la prova verrà valutata 0

  .

Si consideri un semplice linguaggio imperativo, con scoping statico e tipi espliciti, costruito a *partire* dalla stessa *sintassi concreta* del linguaggio **Go** (http://golang.org).

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

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

Si hanno le operazioni aritmetiche, booleane e relazionali standard ed i tipi ammessi siano

- i tipi base: interi, booleani, float, caratteri, stringhe, e
- i tipi composti: array (di tipi qualsiasi) e puntatori (a tipi qualsiasi),

con i costruttori e/o le operazioni di selezione relativi.

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

Tutte le modalità di passaggio dei parametri supportate dal linguaggio devono rispettare la loro semantica canonica. In particolare, se una funzione altera il contenuto di un parametro formale passato per valore (di qualsiasi tipo, inclusi gli array), il corrispondente parametro attuale del chiamante non deve subire modifiche.

Il linguaggio deve prevedere—con la sintassi concreta di Go qualora sia prevista—gli usuali comandi per il controllo di sequenza (condizionali semplici, iterazione *indeterminata*).

Per detto linguaggio (non necessariamente in ordine sequenziale):

- Si progetti (e implementi) una opportuna sintassi astratta e si implementi la corrispondente funzione di serializzazione che trasformi un albero di sintassi astratta in sintassi concreta *legale* con un minimo di ordine, ad esempio andando a capo in corrispondenza dei blocchi).
- Si implementi un lexer con Alex, un parser con Happy. Si suggerisce caldamente 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).
- 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 il type-checker. 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.
- Dopo aver definito un opportuno data-type con cui codificare il three-address code (non una stringa di char!), si implementi il generatore di three-address code per il linguaggio considerato (non Go), tenendo presente che:
  - gli assegnamenti devono valutare l-value prima di r-value;
  - l'ordine delle espressioni e della valutazione degli argomenti si può scegliere a piacere (motivandolo);

– le espressioni booleane nelle guardie devono essere valutate con short-cut. In altri contesti si può scegliere a piacere (motivandolo).

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, il pretty-print del sorgente ed (infine) generazione e pretty-print del three-address code. Nel pretty-print del three-address code si serializzino gli identificatori che vengono dal programma aggiungendo l'informazione relativa al punto di dichiarazione nel sorgente originale (ad esempio `t37 = x_12 + 5` se `x` è stata dichiarata alla linea 12).

Importante: si tenga presente che del linguaggio Go si devono utilizzare **solo** le scelte di sintassi concreta per i costrutti (canonici) previsti dal testo precedente. Tutto il resto, sia la sintassi concreta di costrutti non richiesti sia la semantica di funzionamento, è irrilevante ai fini della soluzione e probabilmente anche in contrasto con quanto si richiede di fare. Argomentazioni del tipo "ma in Go queste cose non esistono" o "in Go ci sono questi costrutti che funzionano così" non saranno accettate.