Linguaggi Formali e Compilatori
Proff. Breveglieri, Morzenti
Prova scritta [1]: Domanda relativa alle esercitazioni
02/07/2014

COGNOME: .....
NOME: ..... Matricola: .....
Corso: ∘ Laurea Specialistica ∘ V. O. ∘ Laurea Triennale ∘ Altro: ...
Sezione: ∘ Prof. Breveglieri ∘ Prof.Morzenti

Per la risoluzione della domanda relativa alle esercitazioni si deve utilizzare l'implementazione del compilatore `Acse` che viene fornita insieme al compito. Si richiede di modificare la specifica dell'analizzatore lessicale da fornire a `flex`, quella dell'analizzatore sintattico da fornire a `bison` ed i file sorgenti per cui si ritengono necessarie delle modifiche in modo da estendere il compilatore `Acse` con la possibilità di gestire l'operatore ternario *splice* che, date due espressioni e_1 ed e_2 ed un' espressione costante e_c, è definito sintatticamente come e_1 \$ e_2 @ e_c. L'*espressione* risultante si ottiene combinando e_c bit più significativi di e_1 con $32 - e_c$ meno significativi di e_2. L'implementazione

```
int a, b, splice;

read(a);
read(b);

// a = 32768; b = 65536
// a = 00000000000000010000000000000000
// b = 00000000000000001000000000000000

splice = a $ b @ 16;

// splice = 98304
// splice = 00000000000000011000000000000000

write(splice);
```

Figura 1: Esempio

dell'operatore deve verificare che il terzo operando sia una costante e che il suo valore sia compreso tra 0 e 32. Se il valore di e_c è superiore a 32 allora lo splice viene realizzato su 32 bit.

[1] Tempo 60'. Libri e appunti personali possono essere consultati.
È consentito scrivere a matita. Scrivere il proprio nome sugli eventuali fogli aggiuntivi.

1. Definire i token (e le relative dichiarazioni in `Acse.lex` e `Acse.y`) necessari per ottenere la funzionalità richiesta. (3 punti)

2. Definire le regole sintattiche (o le modifiche a quelle esistenti) necessarie per ottenere la funzionalità richiesta. (4 punti)

3. Definire le azioni semantiche (o le modifiche a quelle esistenti) necessarie per ottenere la funzionalità richiesta. (18 punti)

4. Data il seguente snippet di codice `Lance`:

```
int a, b = 6, c;
read(a);
read(c);

write(-a * b + c > 12 -b | a);
```

scrivere l'albero sintattico dell'espressione che è argomento della funzione **write**, partendo dalla grammatica Bison definita in `Acse.y` ed *iniziando dal non-terminale* `exp`. (5 punti)

5. (**Bonus**) Descrivere l'estensione dell'operatore di splice che supporti l'utilizzo di un'espressione generica per caratterizzare l'operando e_c. Il seguente snippet di codice ne mostra un esempio.

```
int a, b, c, splice;

read(a);
read(b);
read(c);

splice = a $ b @ (c+a-1);
```

Figura 2: Splice generalizzato