Linguaggi Formali e Compilatori
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Prova scritta [1]: Domanda relativa alle esercitazioni 30/09/2011

COGNOME: .....
NOME: ..... Matricola: .....
Iscritto a: ○ Laurea Specialistica ○ V. O. ○ Laurea Triennale ○ Altro: ...
Sezione: ○ Prof. Breveglieri ○ Prof. Crespi ○ 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' esecuzione di codice pilotata da un bit della program status word (PSW) tramite il costrutto `on <flag> run`, dove `<flag>` $\in$ {`neg`, `zero`, `pos`, `notzero`}.

```
1 int a,b,c;
2 on zero run a=0;
3 c= 5 * a + 7;
4 on neg run {
5    while(c>0){
6       c = c-a;
7    }
8 }
```

(a) Codice

Flag	Bit della PSW
`nonzero`	Z clear
`zero`	Z set
`pos`	N clear
`neg`	N set

(b) riferimento Program Status Word

Figura 1: Esempio di uso del costrutto `on <flag> run`

Il costrutto `on <flag> run` consente l' esecuzione del blocco di codice dopo la parola chiave `run` solo se il corrispondente bit della program status word del processore è nello stato indicato. In particolare, la tabella di corrispondenza tra il valore di `<flag>` e lo stato del bit della program status word che ne consente l' esecuzione è riportata in Tabella 1(b). Il costrutto `on <flag> run` può essere arbitrariamente annidato in altri costrutti.

[1] Tempo 45'. Libri e appunti personali possono essere consultati.
È consentito scrivere a matita. Scrivere il proprio nome sugli eventuali fogli aggiuntivi.

Si espliciti ogni eventuale ulteriore assunzione che sia ritenuta necessaria a completare la specifica data.

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

   La soluzione è riportata nella patch allegata.

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

   La soluzione è riportata nella patch allegata.

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

L' implementazione del costrutto richiesto è sostanzialmente quella di un costrutto if molto semplificato.

In particolare, il costrutto decisionale in questione non ha la possibilità di fare salti condizionati rispetto a un' espressione, ma si basa solamente sul controllare il valore di un bit della parola di stato. Ciò è possibile direttamente utilizzando le istruzioni di salto condizionato offerte dall' architettura.

Inoltre, a differenza del comune costrutto `if` non vi è la possibilità di specificare un ramo `else`.

Di conseguenza l' implementazione del costrutto richiesto si riduce a generare un salto condizionato con la condizione corretta ad un etichetta che va poi assegnata alla fine del blocco di codice che eventualmente non va eseguito. Va curato il fatto che la condizione del salto deve essere quella opposta a quella specificata dalla flag, in quanto si tratta di un salto usato per evitare l' esecuzione del blocco di codice che segue.

La soluzione completa è riportata nella patch allegata.

4. Dato il codice di Figura 2:

```
1  c & a * b == d
```

Figura 2: Espressioni miste

Scrivere l'albero sintattico relativo partendo dalla grammatica Bison definita in `Acse.y` *iniziando dal non-terminale* `exp` a precedenza più bassa. (5 punti)

La soluzione è riportata in Figura 3.

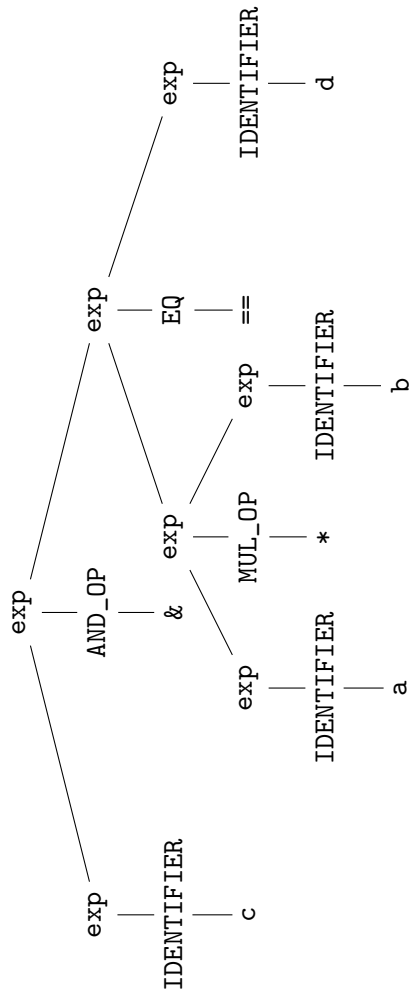

Figura 3: Albero sintattico del codice riportato in Figura 2

5. (Bonus) Si supponga di voler supportare un estensione del costrutto `on <flag> run` da realizzarsi come `on (<flag1>,<flag2>) run`. Questo costrutto consente l' esecuzione del codice solamente nel caso in cui le condizioni dettate da entrambe le `<flag>`.
   Quali modifiche andrebbero apportate al compilatore `Acse` per supportare questo costrutto esteso?
   Quale ottimizzazione sarebbe possibile implementare per gestire il caso `on (pos,neg) run`?

   É possibile implementare il costrutto semplicemente generando due branch consecutivi, uno per flag seguendo la stessa logica utilizzata per realizzare il costrutto base; i token da aggiungere non cambiano. Una possibile ottimizzazione per il caso `on (pos,neg) run` è realizzabile assegnando valori semantici ai token delle flag e, nel caso vi siano due condizioni logicamente opposte, generare semplicemente un' istruzione di salto incondizionato alla fine del blocco di codice (che viene comunque generato dall' espansione di `code_block`).

Applicare una patch

Sul sito del corso è disponibile una patch contenente la soluzione del tema d'esame per quanto riguarda la modifica della macchina `Acse`.

Per applicare la patch:

1. scaricare la macchina `Acse` versione 1.1.0
2. scaricare la patch `soluzione-30-09-11.diff`
3. scompattare l'archivio contenente la macchina `Acse`
4. usando il terminale, portarsi nella directory in cui è stata estratta la macchina `Acse`
5. copiare in tale cartella la patch
6. applicare la patch tramite il comando

```
patch -p1 < soluzione-30-09-11.diff
```

La patch è un normalissimo file di testo, contenente le differenze tra la versione di `Acse` con implementata la soluzione dell'esame e la versione 1.1.0.

Le righe che iniziano con il carattere `+` sono state aggiunte, mentre quelle con il carattere `-` sono state rimosse.