Linguaggi Formali e Compilatori
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Prova scritta[1]: Domanda relativa alle esercitazioni
24/09/2008

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 il seguente costrutto condizionale* con la sintassi (C-like).

```
int a;
int b;
...
b = a<7 ? 5 : 3;
...
```

Il significato semantico del costrutto precedentemente illustrato è : se `a` ha valore inferiore a 7 , allora assegna a `b` il valore 5, altrimenti assegna il valore 3. Più in generale, il costrutto valuta la condizione che precede l' operatore `?` e, nel caso abbia valore diverso da 0 propaga il valore dell' espressione immediatamente a sinistra dell' operatore `:`, altrimenti propaga il valore dell' espressione a destra dell' operatore `:`.

I vincoli implementativi sono :

- Il costrutto va gestito come un' *espressione* e non come un generico statement del linguaggio
- Gli operatori `?` e `:` hanno priorità minima , i.e. inferiore a qualsiasi altro operatore
- Gli operatori `?` e `:` sono associativi a destra
- La condizione valutata (la parte prima dell' operatore `?`) può fare utilizzo di qualsiasi tipo di operatore aritmetico/logico tranne l' operatore di assegnamento

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

- Ogni elemento del costrutto non può essere uno statement generico ma solo un'espressione.

Esempi:

```
c = (a==7) ? 20 : a + c;                    /* corretto */
c = a+b*c ? 37*v : 5;                       /* corretto */
c = a+b ? d=5 : e;                          /* statement generico non consentito */
c = (a=b+c) ? h : i ;                       /* assegnamento di a non consentito */
c = (if (a==3){c=2} else {c=1}) ? 7 : m;    /* statement generico non consentito */
c = ((2*b) || ! (2*b)) ? t : f ;            /* corretto */
```

Le modifiche devono mettere il compilatore `Acse` in condizione di analizzare la correttezza sintattica del costrutto sopra descritto e di generare una traduzione corretta nel linguaggio assembler della macchina `Mace`.

Per risolvere il problema si ricordano le seguenti funzioni di `axe_engine.[h,c]`:

- `getNewRegister` : restituisce il numero di un registro (del banco di registri infinito) non ancora assegnato;

```
/* get a register still not used. This function returns
 * the ID of the register found */
int getNewRegister(t_program_infos *program);
```

- `reserveLabel` : riserva una nuova etichetta assembly

```
 /* reserve a new label identifier and return the identifier to the caller */
extern t_axe_label * reserveLabel(t_program_infos *program);
```

- `fixLabel` : assegna un' etichetta alla prossima istruzione assembly

```
/* assign the given label identifier to the next instruction. Returns
 * the label assigned; otherwise (an error occurred) LABEL_UNSPECIFIED */
extern t_axe_label * fixLabel(t_program_infos *program, t_axe_label *label);
```

- `assignNewLabel` : esegue nell' ordine una `reserveLabel` e una `fixLabel`

```
/* reserve and fix a new label. It returns either the label assigned or the
 * value LABEL_UNSPECIFIED if an error occurred */
extern t_axe_label * assignNewLabel(t_program_infos *program);
```

Si ricorda inoltre che :

- La macro `REG_0` identifica il registro R0 che contiene sempre il valore 0
- I possibili modi di indirizzamento per le operazioni aritmetiche a 3 registri sono :
  - `CG_DIRECT_ALL` : tutti i registri usati non contengono indirizzi

- `CG_INDIRECT_ALL` : il registro destinazione e il secondo registro sorgente contengono indirizzi
- `CG_INDIRECT_DEST` : il registro destinazione contiene un indirizzo
- `CG_INDIRECT_SRC` : il secondo registro sorgente contiene un indirizzo

1. Definire (16 punti):

   - I token (e le relative dichiarazioni in Acse.lex e Acse.y) aggiuntivi, *solo se necessari*, per implementare il nuovo costrutto.
   - Le regole sintattiche necessarie per implementare il nuovo costrutto.
   - Le modifiche alle strutture dati (*solo se necessarie*) per supportate il nuovo costrutto.
   - Definire le azioni semantiche necessarie per supportate il nuovo costrutto.

   Si assuma di avere a disposizione una funzione di liberia:

   ```
   void  propagate_expression (int regdest, t_axe_expression expr)
   ```

   Questa funzione genera automaticamente il codice necessario a propagare il valore dell' espressione `expr` (indipendentemente dal fatto che `expr` sia un immediato o un registro) nel registro `regdest`.

   In `Acse.lex` é necessario aggiungere un token identificativo del simbolo di domanda `'?'`:

   ```
   "?" { return QUESTIONMARK; }
   ```

   Non é necessario aggiungere ulteriori tokens in quanto il simbolo `':'` é giá gestito.

   In `axe_struct.h` aggiungiamo la seguente struttura dati:

   ```
   typedef struct t_cond_expr
   {
      t_axe_label *label_false;
      t_axe_label *label_end;
      int result_register;
   } t_cond_expr;
   ```

   In `Acse.y` si aggiunge la seguente dichiarazione di token:

   ```
   %token <cond_expr> QUESTIONMARK
   ```

   Si aggiunge inoltre la seguente regola di precedenza facendo in modo che sia la prima ad apparire nell'elenco delle regole di precedenza:

   ```
   %right COLON QUESTIONMARK
   ```

   La regola bison da aggiungere al file Acse.y é la seguente:

```
exp : exp QUESTIONMARK {
     $2.label_false = reserveLabel(program);
     $2.label_end = reserveLabel(program);
     $2.result_register = getNewRegister(program);
     gen_beq_instruction (program, $2.label_false, 0);
} exp COLON {
     propagate_expression ($2.result_register, $4);

     gen_bt_instruction (program, $2.label_end, 0);

     fixLabel(program, $2.label_false);
} exp {
     propagate_expression ($2.result_register, $7);

     fixLabel(program, $2.label_end);
     $$ = create_expression($2.result_register, REGISTER);
}
```

2. Modificando la grammatica ottenuta al punto precedente fare in modo che non sia possibile utilizzare operatori diversi da `==` e `!=` in qualsiasi punto del costrutto. Seguono alcuni esempi di codice. Per ciascuna delle nuove regole *non* é necessario specificare le azioni semantiche. (7 punti):

```
c = a+b ? d : e;                    /* non corretto */
c = a ? d+b : e;                    /* non corretto */
c = a ? d : e+b;                    /* non corretto */
c = a==b ? d : e;                   /* corretto */
c = (a!=b)+c ? t : f ;              /* non corretto */
c = (a!=b) ? t : f ;                /* corretto */
c = (a==b) || (a!=b) ? t : f ;      /* non corretto */
```

Introduciamo le seguenti regole sintattiche:

```
cond_expr : NUMBER
          | IDENTIFIER
          | cond_expr NOTEQ cond_expr
          | cond_expr EQ cond_expr
          | cond_expr QUESTIONMARK cond_expr COLON cond_expr
          | LPAR cond_expr RPAR
;
```

Modifichiamo infine la grammatica del costrutto ottenuto al punto precedente nel modo seguente:

```
exp : cond_expr QUESTIONMARK cond_expr COLON cond_expr
```

Sono state considerate corrette anche soluzioni che non prevedano la formulazione della seguente regola sintattica:

```
cond_expr : cond_expr QUESTIONMARK cond_expr COLON cond_expr
```

3. Dato il seguente codice sorgente :

```
int a = 10;
int b = 20;

do {
   a = b / a * a;
} while (a!=0)
```

scrivere l' albero sintattico generato dalla grammatica Bison definita in Acse.y *partendo dal nonterminale* `statements` (5 punti)

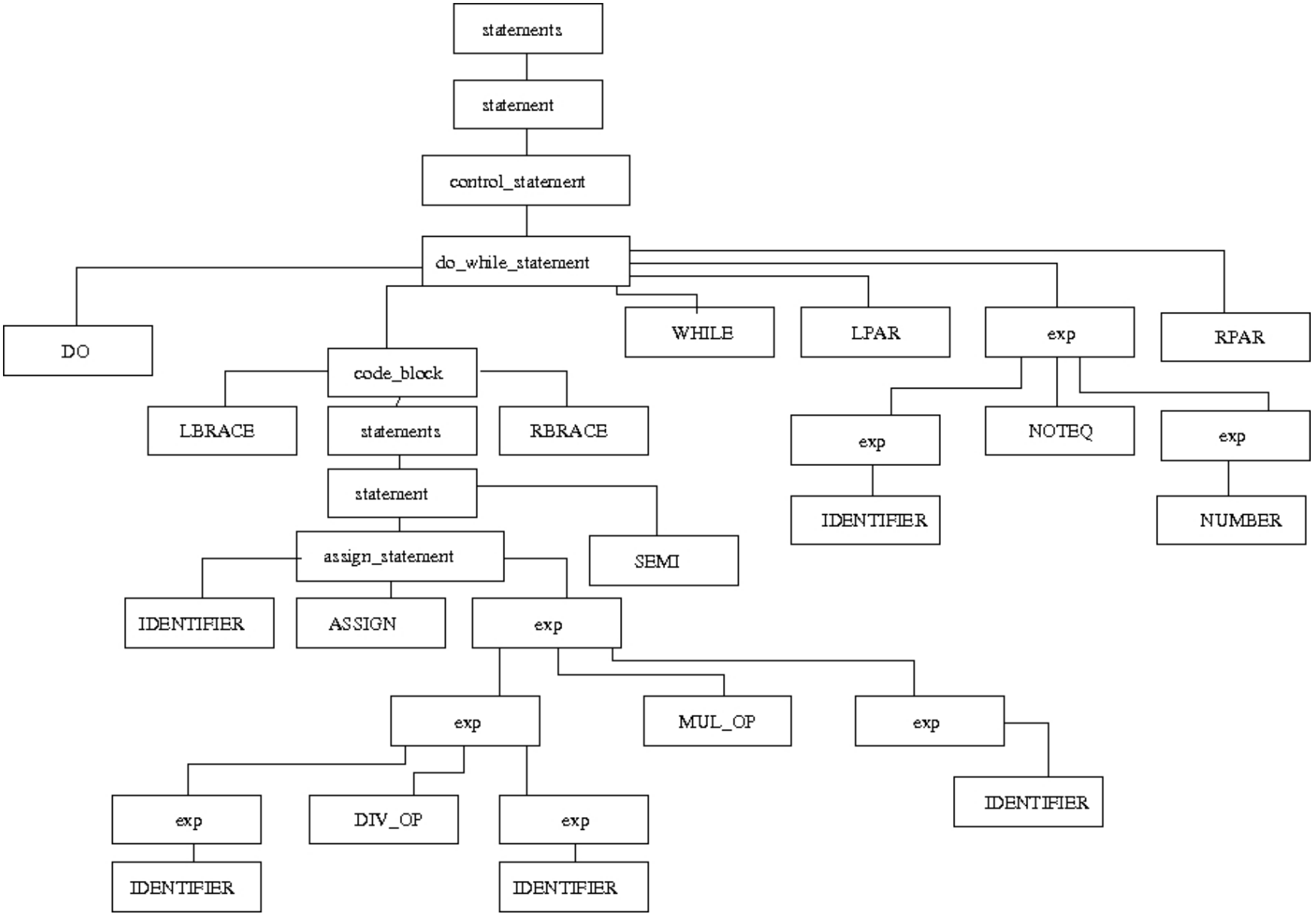

Figura 1: Albero sintattico

4. (*Facoltativo*) (5 punti) Dato il seguente codice sorgente :

   ```
   d = 5 < 7 ? a ? b : c ?  a : b : a;
   ```

   scrivere l' albero sintattico generato dalla grammatica Bison modificata al punto 2 *partendo dal nonterminale* ***assign_statement***