Linguaggi Formali e Compilatori
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Prova scritta[1]: Domanda relativa alle esercitazioni
11/09/2008

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 gli operatori di referenziazione e dereferenziazione* con la seguente sintassi (C-like).

```
int *a;
int c;
...
a = &c;
c = *a;
...
```

Si ipotizzi che la grammatica sia già in grado di gestire

- il tipo puntatore ad intero
- dichiarazioni di variabili di tipo puntatore ad intero

Si implementino *soltanto* gli operatori unari `&` e `*` rispettando i seguenti vincoli:

- Gli operatori `&` e `*` sono associativi a destra
- Gli operatori `&` e `*` *NON* possono essere utilizzati per comporre il left-value di un assegnamento
- Gli operatori `&` e `*` *NON* possono essere associati ad array
- Gli operatori `&` e `*` possono soltanto essere associati ad identificativi di variabili
- *NON* deve essere possibile dichiarare puntatori multipli

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

Esempi:

```
a = &b;         /* corretto */
a = *b;         /* corretto */
a = *b + *c;    /* corretto */
a = *b + 10;    /* corretto */
a = *(b + 10); /* non corretto */
a = **b;        /* non corretto */
*a = b;         /* non corretto */
&a = b;         /* non corretto */
*(a + b) = c;  /* non corretto */
a = &c + 10;    /* corretto */
a = &b[5];      /* non corretto */
a = *b[2];      /* non corretto */
```

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

Per risolvere il problema si consiglia di utilizzare 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);
  ```

- `getLabelFromVariableID` : restituisce la "t_axe_label" che marca l'indirizzo di memoria di una variabile.

  ```
  /* retrieves the label associated to the ID which is passed as parameter*/
  t_axe_label * getLabelFromVariableID (t_program_infos *program, char *ID);
  ```

Si ricorda inoltre che :

- La macro `REG_0` identifica il registro R0 che contiene sempre il valore 0
- L' istruzione assembly `MOVA` può essere utilizzata per copiare un indirizzo in un registro
- 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 (9 punti):

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

   Non è necessario introdurre nuovi token lessicali.

   Non è necessario dichiarare ulteriori strutture dati a supporto del processo di traduzione.

   Bisogna introdurre la seguente regola alla grammatica Acse:

   ```
   exp : MUL_OP IDENTIFIER
   ```

   L'azione semantica è definita di seguito:

```
exp : MUL_OP IDENTIFIER {
    /* Il registro che memorizza il valore della
       variabile identificata da $3 (IDENTIFIER) */
    int regID;
    /* Il registro che conterrà il risultato
       dell'operazione di dereferenziazione */
    int regDest;

    /* recupera il registro in cui è memorizzato IDENTIFIER */
    regID = get_symbol_location (program, $2);

    /* ottiene un nuovo registro temporaneo */
    regDest = getNewRegister (program);

    /* Sposta nel registro 'regDest' in contenuto della
     * locazione di memoria puntata dall'indirizzo in 'regID' */
    gen_add_instruction (program, regDest, REG_0, regID, CG_INDIRECT_SRC);


    $$ = create_expression (regDest, REGISTER);
}
```

2. Definire (9 punti):

   - I token (e le relative dichiarazioni in Acse.lex e Acse.y) aggiuntivi, *solo se necessari*, per implementare l'operatore di referenziazione (`&`).
   - Le regole sintattiche necessarie per implementare l'operatore di referenziazione (`&`).
   - Le modifiche alle strutture dati (*solo se necessarie*) per supportate l'operatore di referenziazione (`&`).
   - Definire le azioni semantiche necessarie per supportate l'operatore di referenziazione (`&`).

   Non è necessario introdurre nuovi token lessicali.

   Non è necessario dichiarare ulteriori strutture dati a supporto del processo di traduzione.

   Bisogna introdurre la seguente regola alla grammatica Acse:

   ```
   exp : AND_OP IDENTIFIER
   ```

   L'azione semantica è definita di seguito:

   ```
   exp : AND_OP IDENTIFIER {
       t_axe_label *label;
       int movaReg;

       /* recupera la label associata alla variabile $2 */
       label = getLabelFromVariableID (program, $2);

       movaReg = getNewRegister (program);

       /* sposta in 'movaReg' l'indirizzo di
          memoria associato alla label 'label' */
       gen_mova_instruction (program, movaReg, label, 0);

       $$ = create_expression (movaReg, REGISTER);
   }
   ```

3. Si ipotizzi di avere a disposizione le seguenti funzioni di libreria:

   - `int is_address_expression ( t_axe_expression * exp )` : restituisce 1 se `exp` è un puntatore.
   - `void set_address_expression ( t_axe_expression * exp )` : marca l' espressione come puntatore.
   - `void invalid_operation_found()` : segnala l' uso errato di puntatori e blocca il processo di traduzione.

   Modificare le azioni semantiche delle seguenti regole Bison (5 punti)

```
exp :  exp MINUS exp  { ... }
   | exp DIV_OP exp  { ... }
```

   affinchè :

   - Non si possano dividere puntatori.
   - Gli operandi della sottrazione possano essere puntatori.
   - Le operazioni di sottrazione in cui soltanto uno degli operandi è un puntatore, diano come risultato un puntatore.
   - Le operazioni di sottrazione in cui entrambi gli operandi sono puntatori *NON* diano come risultato un puntatore.
   - Ogni infrazione delle regole sia segnalata e la traduzione interrotta.

   Esempio :

```
 int *b;

 b-7; /* sottrazione valida */
 b/7; /* divisione non consentita */
```

   È necessario ridefinire l'azione semantica della regola

```
exp :  exp PLUS exp;
```

   nel modo seguente:

```
exp : exp MINUS exp
{
    $$ = perform_bin_numeric_op(program, $1, $3, SUB);

    if (is_address_expression ($1)
        ^^ is_address_expression ($3) )
    {
        set_address_expression ($$);
    }
};
```

È necessario ridefinire l'azione semantica della regola

```
exp :  exp DIV_OP exp;
```

nel modo seguente:

```
exp : exp DIV_OP exp
{

    if ( is_address_expression ($1)
       || is_address_expression ($3) )
    {
        invalid_operation_found ();
    }

    $$ = perform_bin_numeric_op(program, $1, $3, MUL);
};
```

4. Dato il seguente codice sorgente :

```
int b=0;

while(b < 36)
{
  b = b + 2 * b;
}
```

scrivere l' albero sintattico generato dalla grammatica Bison definita in Acse.y ignorando le dichiarazioni e *partendo dall' analisi del nonterminale* `statements` (5 punti)

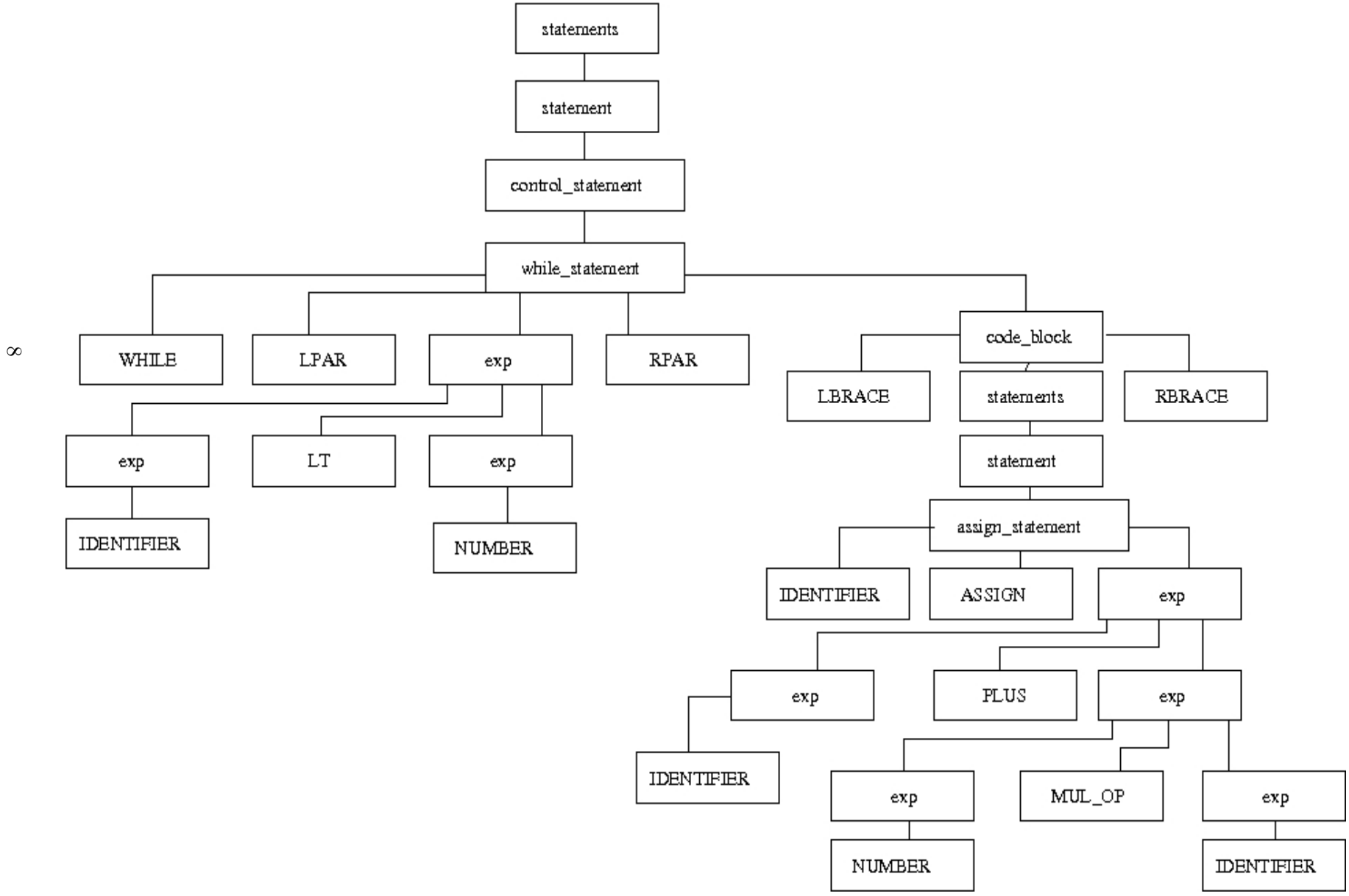

Figura 1: Albero sintattico

5. (*Facoltativo*) (5 punti) Implementare il supporto all' aritmetica dei puntatori C-like per la sottrazione : nel caso di sottrazione mista tra un operando di tipo puntatore e uno di tipo intero, l' intero va moltiplicato per 4 prima di sottrarlo (in quanto la dimensione della parola della macchina Mace è 4 bytes). Allo scopo è possibile modificare la regola di sottrazione definita al punto 3.