Linguaggi Formali e Compilatori
Proff. Breveglieri, Crespi Reghizzi, Sbattella
Prova scritta[1]: Domanda relativa alle esercitazioni
07/03/2006

COGNOME e NOME:..... Matricola:.....
Iscritto a: ∘ Laurea Specialistica ∘ Vecchio Ordinamento ∘ Altro:.....
Sezione: ∘ Prof. Breveglieri ∘ Prof. Crespi ∘ Prof.ssa Sbattella

Per la risoluzione della domanda relativa alle esercitazioni si deve utilizzare l'implementazione del compilatore `Simple` 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 linguaggio `Simple` con la possibilità di usare un tipo puntatore simile a quello del linguaggio `C`:

```
declarations
 integer * <var_puntatore>, a.
 ...
begin
 ...
 a = *<var_puntatore> + 3;
 ...
 <var_puntatore> = &a;
 ...
 *(<var_puntatore>+1) = 5 * a;
 ...
end
```

Le modifiche devono mettere il compilatore `Simple` in condizione di analizzare la correttezza sintattica dei costrutti sopra descritti e di generare una traduzione corretta nel linguaggio assembler della macchina `SimpleVM`, estesa con i seguenti opcode:

- STORE_STACK: salva all'indirizzo indicato dal top dello stack il valore immediatamente sottostante, consumando entrambi i valori.
- LOAD_STACK: carica sulla cima dello stack il valore all'indirizzo contenuto nel top dello stack.
- SWAP: scambia i primi due valori in cima allo stack.

[1] Tempo 30'. Libri e appunti personali possono essere consultati.
È consentito scrivere a matita. Scrivere il proprio nome sugli eventuali fogli aggiuntivi.

1. Definire i token necessari, le regole sintattiche relative alla dichiarazione di variabili di tipo puntatore e alle operazioni di dereferenziazione e calcolo dell'indirizzo. Si noti che nella variante proposta del linguaggio Simple, nella parte sinistra dell'assegnamento può comparire una espressione.

   Soluzione:
   In Simple.y aggiungere le seguenti produzioni:

```
id_seq
  : /* empty */
  | id_seq IDENTIFIER ','
  | id_seq '*' IDENTIFIER ','

exp
  : ....
  | '&' IDENTIFIER
  | '*' '(' exp ')'
  | '*' IDENTIFIER

commands
  : .....
  | '*' '(' exp ')' ASSGNOP exp
  | '*' IDENTIFIER ASSGNOP exp
```

   Occorre fare presente, con una breve nota, che modifiche analoghe influenzano anche le regole in cui compaiono le parole chiave `write` e `read` (Es: `write *mysymb`)

2. Definire le azioni semantiche relative alla dichiarazione di variabili di tipo puntatore e alle operazioni di dereferenziazione e calcolo dell'indirizzo. Si faccia attenzione alla corretta gestione dei tipi delle espressioni.

   Soluzione:
   Alla struttura `symrec` (file ST.h), che rappresenta una entry nella symbol table, si aggiunge un campo che memorizza il tipo del simbolo (per convenzione, 0=intero, 1=puntatore)

```
typedef struct _symrec{
......
int type;
......
} symrec;
```

   La funzione `insert` si suppone modificata opportunamente per gestire il nuovo campo: in particolare, prenderà in ingresso un parametro aggiuntivo identificante il tipo del simbolo trattato.
   Inoltre, bisogna tenere conto che ora le espressioni sono tipizzate (ossia il loro risultato può essere un intero o un puntatore. Anche in questo caso supponiamo che se il tipo di una espressione è 0, essa è di valore intero, se è 1 è un puntatore. Per tipizzare le espressioni si utilizza il costrutto `type` di Bison, come segue:

```
%type <intval> exp
```

   Veniamo quindi alle azioni semantiche:

```
id_seq
  : /* empty */
  | id_seq IDENTIFIER ','
    {insert($2,0);} /* Inserisce un simbolo intero nella ST */
  | id_seq '*' IDENTIFIER ','
    {insert($3,1);} /* Inserisce un simbolo puntatore nella ST */

exp
  : ....
  | '&' IDENTIFIER
    {context_check(LDINT,$1); /* Carica l'offset di $1 sulla cima dello stack */
     $$ = 1;       /* Tipizza l'espressione come puntatore */
    }
  | '*' '(' exp ')'
    {gencode(LOAD_STACK,0);
     $$ = 0;       /* Tipizza l'espressione come intera */
    }
  | '*' IDENTIFIER
    {gencode(LOAD_STACK,0);
```

```
    $$ = 0;       /* Tipizza l'espressione come intera */
  }
| exp + exp
  {$$ = $1 || $3; /* Un esempio di propagazione di tipo. Se $1 o $3 sono
       puntatori, allora lo \‘e anche $$ (basta un esempio) */
  }

commands
  : .....
  | ‘*’ ‘(’ exp ‘)’ ASSGNOP exp
    {gencode(SWAP,0);
     gencode(STORE_STACK,0);}
  | ‘*’ IDENTIFIER ASSGNOP exp
    {gencode(SWAP,0);
     gencode(STORE_STACK,0);}
```

Occorre fare presente, con una breve nota, che modifiche analoghe influenzano anche le regole in cui compaion o le parole chiave `write` e `read` (Es: `write *mysymb`). Occorre inoltre fare presente, con una breve nota, che le modifiche sull'assegnamento vanno introdotte anche nei punti in cui l'assegnamneto è riportato in modo esplicito (ad esempio, all'interno della regola relativa al `for`)

3. Introdurre un controllo semantico che impedisca di assegnare valori puntatore a variabili di tipo intero. Il contrario è invece permesso.

   Soluzione Anche in questo caso basta un esempio:

```
commands
  : .....
  | '*' '(' exp ')' ASSGNOP exp
     {if ($3==0 && $6==1) then{
         printf("Errore.")
         exit(1);
      }
      else{
         gencode(SWAP,0);
         gencode(STORE_STACK,0);
      }}
```

4. Implementare nell'interprete SimpleVM le istruzioni STORE_STACK, LOAD_STACK e SWAP.

   Soluzione: si tenga presente che questo esercizio, assieme al primo, è il più facile.

```
case STORE_STACK: stack[stack[top]] = stack[top-1]; top-=2;
case LOAD_STACK:  stack[top] = stack[stack[top]]
case SWAP:    int tmp=stack[top];
                  stack[top] = stack[top-1];
                  stack[top-1]=tmp;
```

5. **bonus**[2] L'uso di puntatori è principalmente utile in presenza di allocazione dinamica. Discutere quali modifiche sono richieste all'interprete SimpleVM e al linguaggio Simple per gestire l'allocazione e deallocazione dinamica della memoria (i.e., uno heap).
   Soluzione: La macchina virtuale Simple potrebbe riservare un banco di memoria da gestire come uno heap, nel quale vengono allocate le variabili puntatore. Le modifiche al linguaggio richiedono l'introduzione di eventuali parole chiavi che indichino la allocazione (es: new, alloc) e la deallocazione (es: delete, free, old).

[2]La domanda bonus verrà valutata solo se le parti precedenti del compito totalizzano almeno 18 trentesimi.