Linguaggi Formali e Compilatori
Proff. Breveglieri, Crespi Reghizzi, Sbattella
Prova scritta[1]: Domanda relativa alle esercitazioni
10/09/2007

COGNOME: .....
NOME: ..... Matricola: .....
Iscritto a: ○ Laurea Specialistica ○ V. O. ○ Laurea Triennale ○ 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 compilatore `Simple` con la possibilità di gestire eccezioni sollevate dall'utente.

```
declarations
integer a, b, c.
exception ex1, ex2.
begin
     a := 1;
     b := 2;
try :
     read c;
     if c<a
      then throw ex2;
      else throw ex1;
catch ex1 do
     b := b + c ;
od
 write b;
end
```

Ogni volta che viene eseguito un `throw` all'interno di un blocco `try`, il controllo passa al successivo blocco `catch` che cattura l'eccezione sollevata. Una eccezione non gestita da alcun comando `catch` provoca la terminazione del programma. Per ogni `try` c'è esattamente un `catch`. Blocchi *try-catch* possono essere nidificati. `throw` e `catch` vogliono dei nomi di eccezioni come argomenti, che vanno definiti all'inizio

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

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`.

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

   NOTA ALLA SOLUZIONE: La seguente soluzione contiene anche una spiegazione – più o meno dettagliata – del suo "funzionamento". In sede di esame questo non è strettamente richiesto (seppur gradito). Quello che importa è ciò che nel seguito apparirà come codice. Inoltre, il grado di dettaglio qui presentato è ben al di sopra degli standard richiesti per il 30 e Lode.

   In Simple.lex aggiungere:

   ```
   try {return TRY;}
   catch {return CATCH;}
   throw {return THROW;}
   exception {return EXCEPTION;}
   ```

   In Simple.y aggiungere:

   ```
   %token TRY,CATCH,THROW,EXCEPTION
   ```

2. Definire le regole sintattiche necessarie per ottenere la funzionalità richiesta.

   Per quanto riguarda la dichiarazione di eccezioni, si può procedere come segue. Aggiungere in Simple.y:

   ```
   declarations: /* empty */
   | declaration declarations;
   declaration: INTEGER id_seq IDENTIFIER '.'
   | EXCEPTION id_seq IDENTIFIER '.'
   ```

   Per quanto riguarda i comandi, viene qui fornita la soluzione più semplice (e meno elegante). Una soluzione più elegante prevede di forzare – già a livello sintattico – la possibilità di effettuare `throw` solo all'interno di un blocco `try`.

   Si fa inoltre l'ipotesi che un `catch` possa catturare una e una sola eccezione.

   ```
   command: .....
   | THROW IDENTIFIER
   | TRY ':' commands CATCH IDENTIFIER DO commands OD
   ```

3. Definire le modifiche alle strutture dati per supportate la funzionalità richiesta.

   NOTA: questo è solo un esempio di soluzione. Ed è solo una tra le tante soluzioni possibili. Innanzi tutto occorre modificare la symbol table in modo tale da memorizzare variabili di tipo eccezione.

   ```
   typedef struct _symrec{
     char *name;
     int offset;
   ```

```
  int type; /* vale 0 se intero, 1 se eccezione */
  struct _symrec *next;
}symrec;
```

Inoltre va mantenuta una struttura lista (che verrà gestita come una pila) per poter tenere traccia dei blocchi `try-catch` annidati. Infatti, bisogna tener conto che un'eccezione sollevata all'interno di un blocco `try-catch` innestato potrebbe essere catturata da un `catch` appartenente ad un blocco più esterno, mentre non è vero il contrario.

Per ogni blocco `try-catch` il parser creerà una istanza di una opportuna struttura, che verrà impilata su una pila globale. Tale struttura terrà traccia (mediante una lista) di tutte le eccezioni sollevate all'interno del blocco `try`. Per ogni eccezione verrà memorizzato il punto – all'interno del codice – in cui essa è sollevata. La stessa eccezione può comparire con più di una istanza nella lista (nel caso sollevata in più punti differenti del programma). Se opportunamente gestita da un blocco catch, tutte le sue istanze verranno rimosse dalla lista.

Quando il parser esce dall'analisi di un blocco `try-catch`, la struttura dati ad esso relativa viene disimpilata. Se la lista delle eccezioni non è vuota, essa viene appesa alla lista delle eccezioni del blocco `try-catch` esterno. Se a pila vuota rimane una lista di eccezioni non catturate, per esse andranno emesse le istruzioni di `HALT`.

Quindi:

```
/* Lista di eccezioni */
typedef struct _exclist{
char *name;
int code_offset;
struct _exclist *next;
} exclist;

/* Blocco Try-Catch */
typedef struct _trycatchBlock{
struct _exclist *exceptionList;
struct _trycatchBlock *next;
} trycatchBlock;

/* Variabile Globale. Pila di Blocchi Try-Catch */
trycatchBlock *trycatchStack = NULL;
```

In più andranno modificate `insert`, `putsym` e affini, per gestire opportunamente la symbol table (prenderanno un parametro in più in ingresso). Inoltre, definiamo le seguenti funzioni:

```
/* Fai un push di un try-catch block sulla pila trycatchStack */
push(trycatchBlock *tb);

/* Fai un pop dalla pila trycatchStack e restituisci il puntatore
   all'elemento "poppato" */
trycatchBlock *pop();

/* Restituisci un puntatore al top dello stack */
trycatchBlock *topStack();

/* Lista dei punti in cui name e' sollevata. NULL se non sollevata */
/* all'interno del blocco try corrente */
exclist * whereExceptionIsThrown(char *name);

/* Rimuouve tutte le istanze di eccezione "name" dalla
   lista al top di trycatchStack */
removeException(char *name);

/* Crea e inserisce una "eccezione" dalla lista al top di trycatchStack.
   L'eccezione ha nome=name e code_offset=offset */
addException(char *name, int offset);

/* Appende una lista (l2) ad un'altra (l1) */
appendList(exclist *l1, exclist *l2);
```

4. Definire le azioni semantiche necessarie per ottenere la funzionalità richiesta. Per semplicità supponiamo che all'interno di un blocco `catch` non possano esserci altri `try-catch`. Sebbene la cosa sia permessa a livello sintattico, la si vieta a livello semantico per mezzo della variabile booleana `catchblock`, che vale 1 se si è interni ad un blocco `cacth`. In caso contrario una gestione più attenta degli scope andrebbe implementata, complicando sensibilmente la soluzione.

```
declarations: /* empty */
        | declaration declarations;
declaration: INTEGER {curType = 0;} id_seq IDENTIFIER '.'
             { insert($4,0); }
        | EXCEPTION {curType = 1;} id_seq IDENTIFIER '.'
             { insert($4,1);}
id_seq
   : /* empty */
   | id_seq IDENTIFIER ','
     { insert($2,curType);}
```

```
   ;

command: .....
        | THROW IDENTIFIER
        {
          { If (getsym($6,1) == 0){
                printf("Errore: eccezione mai dichiarata\n");
                exit(1);
          }
          if (trycatchStack==NULL){
             /* Eccezione fuori blocco */
             gen_code(HALT,0);
          }
          else
             addException($2,reserve_code_location());
        }

        | .....
        | TRY
{ if (catchblock) {printf("Vietato\n"); exit(1);}
  push((trycatchBlock *)malloc(sizeof(trycatchBlock)));
}
':' commands CATCH IDENTIFIER
{     exclist * el;
       trycatchBlock *tb,top;
       If (getsym($2,1) == 0){
           printf("Errore: eccezione mai dichiarata\n");
           exit(1);
      }
      /* Trova dove l'eccezione e' sollevata */
      el = whereExceptionIsThrown($6);
      /* Fai backpatching */
      while (el!=NULL){
           gen_back_code(el->code_offset,GOTO,current_code_location());
           el = el->next;
      }
      removeException($6);
      /* Disimpila il blocco try-catch e preleva il top */
      tb = pop();
      top = topStack();
      /* Se sei a pila vuota e hai ancora eccezioni, genera gli HALT,
          altrimenti linka le lista delle eccezioni */
      if (top==NULL){
             if (tb->exceptionList != NULL)
                    while(tb->exceptionList != NULL){
```

```
                    gen_back_code(tb->exceptionList->code_offset,HALT,0);
                     tb->exceptionList = tb->exceptionList->next;
            } /* while */
     }
     else /* top != NULL */
              appendList(top->exceptionList, tb->exceptionList);
}
DO {catchblock = 1;} commands OD {catchblock = 0;}
```

5. **Bonus** Implementare anche il costrutto *finally* come in Java. Si ricorda che il codice di un blocco *finally* viene eseguito ogni volta che si esce da un blocco *try-catch*, independentemente dal fatto che sia stata sollevata un'eccezione, e che questa sia stata catturata.