Corso di Linguaggi e Traduttori
Anno Accademico 2015–2016
Prof. Giovanni Pighizzini

Progetto d'esame

Sviluppate a scelta uno tra i due progetti specificati nel seguito:

1. compilatore di straight-line program,
2. compilatore di un semplice linguaggio di programmazione.

Alla fine del documento (pagina 10) sono riportate le richieste *comuni ai due progetti*.

1 Straight-line program

Un programma è una sequenza di istruzioni di assegnamento e di scrittura. Ogni istruzione termina a fine riga; tuttavia è possibile indicarne la prosecuzione sulla riga successiva, concludendo la riga corrente con il carattere `%`.

Le *istruzioni di assegnamento* hanno le seguenti forme

```
<identificatore> = <espressione>
<tipo> <identificatore> = <espressione>
```

Un'istruzione della prima forma è un usuale assegnamento, mentre un'istruzione della seconda forma è una dichiarazione di variabile con assegnamento. Ogni variabile deve essere dichiarata prima di essere utilizzata. Non è possibile dichiarare due variabili con lo stesso nome.

- `<identificatore>` è una sequenza di lettere e cifre che inizia con una lettera e che non sia una delle parole riservate del linguaggio (indicate man mano nel seguito).
- `<tipo>` è una delle due parole riservate `int` e `real`, utilizzate per indicare, rispettivamente, numeri interi e numeri in virgola mobile.
- `<espressione>` è un'espressione costruita utilizzando gli operatori sottoindicati e le parentesi tonde, seguendo la semantica e le regole di precedenza e associatività del linguaggio Java:
  - Operatori di tipo boolean: `&&`, `||` e `!`
  - Operatori di tipo `int` e `real`: gli operatori binari `+`, `-`,`*`, `/`, e gli operatori unari `+` e `-`.
  - Operatori di confronto: `>`, `<`, `<=`, `>=`, `!=`, `==`. Permettono di confrontare due valori interi o reali, fornendo un risultato boolean.
  - Letterali: per il tipo `int` sono sequenze di cifre, per il tipo `real` sono sequenze di cifre contenenti un punto decimale.
  - Operatore condizionale (ternario) `?:` il primo argomento è di tipo boolean, gli altri due possono essere `int` o `real`.

– Operatore di lettura `input` (parole riservata). Questo operatore fornisce come risultato un valore di tipo `int` letto da input, che potrà essere assegnato a una variabile o utilizzato nel calcolo di un'espressione. L'operatore può essere seguito da una stringa che verrà visualizzata sul monitor prima della lettura del dato.

Sono possibili assegnamenti ed espressioni che coinvolgano sia interi che reali. In tal caso il valore intero viene promosso per il calcolo a reale. In ogni caso si ricordi che, come in Java, *i tipi devono essere stabiliti durante la compilazione* e che il tipo di operazione applicato e il tipo del risultato dipendono *esclusivamente* dai tipi degli operandi e non dal loro valore o da come il risultato viene utilizzato.

Ad esempio, in `a + b / c`, se `b` e `c` sono di tipo `int`, la divisione è tra `int`; nel caso `a` sia di tipo `real`, il risultato della divisione intera dovrà poi essere convertito al tipo `real`. Se a una variabile reale viene assegnato il risultato di un'espressione tra interi, la conversione da intero a reale va fatta al momento dell'assegnamento e non prima. Si noti che nel caso dell'operatore condizionale `?:` se il secondo operando è intero e il terzo reale (o viceversa), il risultato dovrà comunque essere di tipo reale, in quanto tutte le decisioni sui tipi vengono prese in compilazione e non dipendono dalla particolare esecuzione. *Non sono possibili* conversioni da reali a interi.

L'*istruzione di scrittura* ha le forme

```
output <stringa>
output <stringa> <espressione>
```

dove `output` è una parola riservata, `<stringa>` è una stringa di caratteri tra virgolette ed `<espressione>` è un'espressione di tipo `int` o `real`.

Esempi

Il seguente programma legge due numeri e riscrive il maggiore.

```
int x = input "Inserisci il primo numero"
int y = input "Inserisci il secondo numero"
int max = x > y ? x : y
output "Numero maggiore: " max
```

Il seguente programma comunica il perimetro di un rettangolo senza utilizzare variabili:

```
output "Il perimetro e' " 2 * (input "Base? " + input "Altezza? ")
```

Ecco lo stesso programma nel quale l'unica istruzione è scritta su due righe:

```
output "Il perimetro e' " 2 * (input "Base? " %
+ input "Altezza? ")
```

È opportuno che il carattere `%` utilizzato per indicare la continuazione dell'istruzione sulla riga successiva venga trattato direttamente dall'analizzatore lessicale (rinviare il trattamento all'analisi sintattica complicherebbe notevolmente, e inutilmente, la grammatica del linguaggio).

Nel seguente esempio si notino le conversioni di tipo:

```
int x = input "Inserisci il primo numero"
int y = input "Inserisci il secondo numero"
int z = x + y
output "somma " z
z = x / y
output "quoziente " z
real k = x / y
output "quoziente " k
k = x / (y + 0.0)
output "quoziente " k
```

In particolare, solo l'ultima divisione è di tipo `real`.

Si ricordi che i tipi degli operatori, come in Java, dipendono solo dai tipi degli operandi e non dai loro valori. Pertanto il seguente esempio deve produrre un errore di tipo durante la compilazione:

```
int y = 0
int x = y > 0 ? 1 : 1.0
```

L'esecuzione delle istruzioni di lettura può dipendere dai dati in input. Ad esempio, in alcuni casi questo programma esegue una sola istruzione di lettura, in altri due:

```
int x = 0
int y = input "?"
int z = y > 0 ? y + input "??" : 3
output "" z
```

Parti opzionali

- L'implementazione del tipo `real` è facoltativa.[1]
- È possibile aggiungere l'istruzione `loop` con il seguente formato:[2]

  ```
  loop <espressione>
    <sequenza-istruzioni>
  endloop
  ```

  dove `loop` ed `endloop` sono parole riservate, `<sequenza-istruzioni>` è una sequenza di istruzioni di assegnamento, output o istruzioni `loop`, secondo le specifiche indicate sopra, ed `<espressione>` è un'espressione di tipo `boolean`. La semantica è analoga a quella del ciclo `while` del linguaggio Java.

  Si noti che, come nel linguaggio Java, le dichiarazioni seguono la struttura statica del programma e non il flusso di esecuzione, come nel seguente esempio:[3]

  ```
  int x = input "Primo numero? "
  int y = input "Secondo numero? "
  loop y != 0
    int resto = x / y
    x = y
    y = resto
  endloop
  output "mcd = " x
  ```

  Si osservi che, se il ciclo non viene mai eseguito, la variabile `resto` alla fine dell'esecuzione non risulta inizializzata. In questo caso ciò non è un problema, in quanto non si tenta di utilizzare il valore di `resto`. Stabilite una strategia per le variabili non inizializzate (esempi: inizializzazione automatica a un valore di default, o a un valore random, oppure segnalazione di errore in esecuzione o in compilazione).

[1] Il tipo rappresenta numeri in *virgola mobile*. Può essere implementato in maniera rudimentale utilizzando due interi, per rappresentare rispettivamente mantissa ed esponente.

[2] I programmi con questa istruzione non sono più "straight–line". Essi sono in grado di esprimere tutte le funzioni calcolabili negli usuali linguaggi di programmazione.

[3] Dunque una variabile dichiarata all'interno di un ciclo è dichiarata un'unica volta, indipendentemente da quante volte verrà eseguito il corpo del ciclo.

2 Un semplice linguaggio di programmazione

La grammatica del linguaggio descritto nel seguito è riportata a pagina 8.

Variabili e tipi

- Il linguaggio prevede variabili di tipo intero e di tipo riferimento. Queste ultime si possono riferire ad array creati dinamicamente durante l'esecuzione, destinati a contenere numeri interi. Nel caso non si riferiscano a nulla, contengono un valore indicato dal letterale riferimento `null`.
- Un identificatore di variabile è una sequenza di lettere e cifre che inizia con una lettera.
- Le variabili devono essere *dichiarate esplicitamente* prima di essere utilizzate. La parola riservata `var` introduce la dichiarazione di una o più variabili di tipo intero, mentre la parola riservata `ref` introduce la dichiarazione di uno o più variabili riferimento. Le dichiarazioni si chiudono con un punto e virgola. Si utilizza la virgola per separare più variabili nella stessa dichiarazione. Ad esempio, si considerino le seguenti righe di codice:

  ```
  var x;
  ref a, z;
  var w, k, y;
  ```

  Nella prima e nella terza riga vengono dichiarate le variabili `x`, `w`, `k` e `y` di tipo intero, nella seconda i riferimenti `a` e `z`. Le variabili di tipo intero sono inizializzate automaticamente a `0`, quelle di tipo riferimento al valore `null`.

Espressioni

Sono espressioni di *tipo riferimento*:

- il letterale `null`,
- gli identificatori di array, come `a` e `z` nella precedente dichiarazione,
- l'espressione di *costruzione di array*, introdotta dalla parola riservata `new`, seguita da un'espressione intera, come ad esempio

  ```
  new x * 2
  ```

  L'espressione fornisce come risultato il riferimento a un nuovo array con capacità uguale al risultato dell'espressione che segue `new`. Ad esempio, se `x` contiene `5`, l'array sarà di `10` elementi. Per convenzione gli indici partono da `0`. Pertanto in questo caso l'indice massimo è `9`. Gli elementi di un array vengono inizializzati automaticamente a `0`.

Le espressioni di *tipo intero* sono definite in maniera usuale a partire dalle costanti intere, dalle variabili intere e dagli elementi di array. In particolare, un elemento di un array viene selezionato scrivendo, come in Java, il riferimento all'array seguito, tra parentesi quadre, dal *selettore* dell'elemento, cioè un'espressione di tipo intero come in

```
z[x - 1]
```

Le espressioni intere possono essere combinate per ottenere altre espressioni utilizzando gli operatori per la somma, sottrazione, moltiplicazione, divisione intera e resto della divisione (indicate rispettivamente con i simboli `+, -, *, /, %`), e le parentesi tonde. Un esempio di espressione corretta dal punto di vista sintattico e dei tipi è

```
x+z[z[x-1]*2] / (w+y)
```

Per precedenze e associatività si faccia riferimento alle regole del linguaggio Java.

Condizioni

Le condizioni sono ottenute mediante l'utilizzo degli usuali operatori di confronto `<`, `<=`, `>`, `>=`, `==` e `!=` applicati a due espressioni intere, e dei soli operatori `==` e `!=`, applicati a due espressioni riferimento.[4]

Istruzioni

- Assegnamento (operatore `=`):
  Il risultato di un'espressione intera può essere assegnato a una variabile intera o a un elemento di un array. Il risultato di un'espressione riferimento può essere assegnato a una variabile riferimento. Esempi:[5]

```
a[x + 2] = 3 + x;
z = a;
z = null;
z = new x * 2;
```

- Selezione:
  Istruzione `if` con parte `else` opzionale (come in Java)

- Iterazione:
  Istruzioni `while` e `do ... while` (sintassi e semantica come le analoghe istruzioni Java).
  Istruzione `for` (differente dal linguaggio Java!). L'istruzione ha in seguente formato:

```
for (<ident> = <espressione1>, <espressione2>)
  <istruzione>
```

  dove `<ident>` è l'identificatore di una variabile semplice, `<espressione1>` e `<espressione2>` sono due espressioni intere e `<istruzione>` indica l'istruzione da ripetere. Inizialmente vengono calcolati i valori x_1 e x_2 di `<espressione1>` e `<espressione2>`. Viene quindi eseguita ripetutamente l'istruzione assegnando alla variabile `i` i valori $x_1, x_1 + 1, \ldots, x_2$ (se $x_2 < x_1$ l'istruzione non viene eseguita). Ad esempio le istruzioni

```
x = 0;
for (i = 1, 10)
  x = x + i;
```

  sommano nella variabile `x` tutti i numeri da 1 a 10. (Le variabili devono essere state dichiarate in precedenza.) Si osservi che le espressioni vengono calcolate solo all'inizio dell'esecuzione del ciclo. Pertanto l'istruzione nel seguente ciclo viene ripetuta 10 volte, nonostante il valore della variabile `y` sia modificato nel corpo del ciclo.

```
x = 0;
y = 5;
for (i = 1, 2 * y) {
  x = x + i;
  y = y - 1;
}
```

[4] Come in Java, il confronto `a == z` tra riferimenti controlla l'uguaglianza tra i due riferimenti e non tra i contenuti degli array eventualmente associati ad essi.

[5] Come in Java, l'assegnamento tra i riferimenti copia solo i riferimenti, non gli array associati. Pertanto, dopo l'esecuzione del secondo assegnamento dell'esempio, `z` e `a` si riferiscono allo stesso array.

- Istruzioni di lettura e scrittura:
  L'istruzione `read` seguita dall'identificatore di una variabile intera o da un elemento di un array, legge da input un intero e lo assegna alla variabile o all'elemento indicato.

  L'istruzione `write` seguita da un'espressione intera visualizza il risultato dell'espressione (senza portare il cursore a capo).

  L'istruzione `writemsg` seguita da una stringa (racchiusa tra virgolette) visualizza la stringa (senza portare il cursore a capo).

  L'istruzione `writeln` porta il cursore all'inizio della riga successiva.

- Blocco:
  Istruzioni e dichiarazioni possono essere raggruppate tra parentesi graffe, formando un'*istruzione composta* o blocco.

  Una dichiarazione che si trovi all'interno di un blocco è visibile anche nei blocchi esterni e successivi, a partire dal punto in cui è scritta (si noti la differenza rispetto al linguaggio Java).

  Si presti attenzione al fatto che le dichiarazioni forniscono informazioni al compilatore e non dipendono in alcun modo dalla sequenza di esecuzione del programma. Pertanto, se la dichiarazione di una variabile si trova nel blocco associato a un ciclo, la variabile è dichiarata una sola volta *indipendentemente* dal numero di ripetizioni del blocco (inoltre essa è dichiarata anche se il blocco non venisse eseguito nel caso di condizione del ciclo immediatamente falsa).

  Analogamente, una dichiarazione che si trovi in uno dei due rami di una selezione, è valida anche nell'altro. Pertanto il seguente frammento di codice non è corretto (doppia dichiarazione del medesimo identificatore):

  ```
  if (...) {
    var x;
    x = 1;
  } else {
    var x;
    x = 2;
  }
  ```

- Istruzione vuota:
  Il punto e virgola chiude le istruzioni (eccetto il blocco, già delimitato dalla parentesi graffa chiusa). Inoltre il punto e virgola da solo costituisce un'istruzione vuota. Ad esempio

  ```
  while (x > 0);
  ```

  è un ciclo `while` infinito nel caso `x` contenga un valore positivo.

Commenti

È possibile introdurre commenti secondo lo stile di C e Java (da `//` a fine riga o racchiusi fra `/*` e `*/`).

Programma

Un programma è una sequenza di istruzioni e dichiarazioni. Alcuni esempi di programmi sono riportati a pagina 9.

Parti opzionali

È possibile omettere le seguenti parti:

- array e tipi riferimento,
- istruzione `for`,
- operatori di confronto: in questo caso le condizioni sono espressioni di tipo intero in cui il valore 0 indica *falso* e ogni altro valore indica *vero*. Esempi:

```
if (x) ...
if (x + z[x -1])...
if (1) ...
```

Si possono inoltre svolgere, in maniera facoltativa, alcune delle parti indicate di seguito.

1. *Operatori booleani* `&&`, `||` e `!`, con *lazy evaluation*, per combinare tra loro condizioni (riferirsi alle precedenze del linguaggio Java).
2. Introdurre un operatore unario `#` che, applicato a un riferimento, fornisca la capacità dell'array associato, come in `#a`. Si noti che l'operatore definisce una nuova forma di espressione intera, per cui è possibile scrivere istruzioni come

```
x = #a + #z;
a[#a - 1] = a[#a -1] * 2;  // indici da 0: #a - 1 e' l'ultima posizione!
k = 0;
while (k < #a) {
  a[k] = 1;
  k = k + 1;
}
```

3. Ciclo *for-each* (analogo a quello di Java) per scandire tutti gli elementi di un array, come in:

```
for (i: z)  //visualizza sulla stessa riga i valori presenti nell'array
  write i;
```

   Il ciclo viene eseguito assegnando alla variabile `i`, uno alla volta, i valori presenti in `z`. La variabile `i` deve essere stata dichiarata in precedenza.

4. Istruzioni `break` e `continue` (senza etichetta), analoghe a quelle del linguaggio Java per forzare la terminazione del ciclo all'interno del quale si trovano o dell'iterazione corrente (al di fuori dei cicli queste istruzioni non hanno alcune effetto).

Osservazioni

- Attenzione a non rivalutare più volte una stessa espressione a fronte di una sola occorrenza nel sorgente. Ad esempio, nel calcolo di `a % b` le espressioni `a` e `b` devono essere valutate una volta sola.
- Attenzione a non mescolare espressioni di tipi differenti. Il linguaggio possiede due tipi: il tipo intero e il tipo riferimento. La sintassi delle espressioni non distingue tra i due tipi. Ad esempio, rispetto alla grammatica fornita è lecito scrivere `x * a` o `x == a`, anche se `x` è di tipo intero e `a` è un riferimento. È opportuno implementare un controllo sui tipi che individui queste situazioni anomale.

La grammatica

programma → *seqDichiarEIstr*

seqDichiarEIstr → ϵ
| *seqDichiarEIstr dichiarazione*
| *seqDichiarEIstr istruzione*

dichiarazione → `var` *seqIdentificatori* `;`
| `ref` *seqIdentificatori* `;`

seqIdentificatori → *identificatore*
| *seqIdentificatori* `,` *identificatore*

istruzione → *assegnamento*
| *selezione*
| *cicloWhile*
| *cicloDoWhile*
| *cicloFor*
| *lettura*
| *scrittura*
| *blocco*
| *vuota*

assegnamento → *variabile* `=` *espressione* `;`

selezione → `if (` *condizione* `)` *istruzione*
| `if (` *condizione* `)` *istruzione* `else` *istruzione*

cicloWhile → `while (` *condizione* `)` *istruzione*

cicloDoWhile → `do` *istruzione* `while (` *condizione* `)`

cicloFor → `for (` *variabile* `=` *espressione* `,` *espressione* `)` *istruzione*

lettura → `read` *variabile* `;`

scrittura → `write` *espressione* `;`
| `writemsg` *stringa* `;`
| `writeln ;`

blocco → `{` *seqDichiarEIstr* `}`

vuota → `;`

espressione → *numero*
| *variabile*
| `null`
| *espressione* `+` *espressione*
| *espressione* `-` *espressione*
| *espressione* `*` *espressione*
| *espressione* `/` *espressione*
| *espressione* `%` *espressione*
| `-` *espressione*
| `+` *espressione*
| `(` *espressione* `)`
| `new` *espressione*

condizione → *espressione* `<` *espressione*
| *espressione* `<=` *espressione*
| *espressione* `>` *espressione*
| *espressione* `>=` *espressione*
| *espressione* `==` *espressione*
| *espressione* `!=` *espressione*

variabile → *identificatore*
| *identificatore* `[` *espressione* `]`

identificatore → sequenza di lettere e cifre che inizia con una lettera

numero → sequenza non vuota di lettere e cifre

stringa → sequenza di caratteri racchiusa tra virgolette

Si ricordi inoltre che il linguaggio prevede i commenti.

Alcuni esempi di programmi

Esempio 1

```
/* Scarsa fantasia */
writemsg "Hello, world!";
writeln;
```

Esempio 2

```
/* Calcolo del massimo comun divisore
   mediante l'algorimo di Euclide */

var dividendo, divisore;

writemsg "Primo numero? ";
read dividendo;
writemsg "Secondo numero? ";
read divisore;

while (divisore != 0) {
   var resto;
   resto = dividendo % divisore;
   dividendo = divisore;
   divisore = resto;
}
writemsg "il massimo comun divisore e' ";
write dividendo;
writeln;
```

Esempio 3

```
/* Somma di un array di interi */

writemsg "quanti numeri desideri sommare? ";
var quanti;
read quanti;
ref numeri;
numeri = new quanti;

var i;
i = 0;
while (i < quanti) {
  writemsg "numero di posizione ";
  write i + 1;
  writemsg "? ";
  read numeri[i];
  i = i + 1;
}
var somma;
somma = 0;
i = 0;
while (i < quanti) {
  somma = somma + numeri[i];
  i = i + 1;
}

writemsg "La somma dei numeri letti e' ";
write somma;
writeln;
```

Esempio 4

```
/* Il crivello di Eratostene */

ref primi;
var nMax, x;

writemsg "Numero massimo da considerare? ";
read nMax;


//creazione e inizializzazione array
primi = new nMax + 2;
x = 2;
while (x <= nMax) {
  primi[x] = 1;  //1 per true
  x = x + 1;
}

//setaccio
var numero, multiplo;
numero = 2;
while (numero <= nMax) {
  if (primi[numero] == 1) {
    multiplo = numero * 2;
    while (multiplo <= nMax) {
      primi[multiplo] = 0; //assegna false
      multiplo = multiplo + numero;
    }
  }
  numero = numero + 1;
}

//comunicazione risultato
writemsg "Elenco primi: ";
numero = 2;
while (numero <= nMax) {
  if (primi[numero] == 1) {
    write numero;
    writemsg " ";
  }
  numero = numero + 1;
}
writeln;
```

Nota: Come nella maggior parte dei linguaggi di programmazione, l'indentazione e i ritorni a capo non hanno alcuna rilevanza dal punto di vista del compilatore e pertanto possono essere eliminati dall'analizzatore lessicale.

1-2 Cosa è richiesto (entrambi i progetti)

- Scrivete un analizzatore lessicale e un analizzatore sintattico per il linguaggio, servendovi degli strumenti presentati a lezione o di strumenti differenti *purché adeguatamente documentati e concordati preventivamente.*
- Scrivete una classe di prova per l'analizzatore lessicale che elenchi i token man mano inseriti.
- Scrivete un compilatore, dal linguaggio sorgente al linguaggio della macchina a stack presentato a lezione. Per generare il codice e per eseguirlo si utilizzino le classi `Codice.java` e `Macchina.java` che NON DEVONO essere modificate.
- Si suggerisce di ispirarsi all'esempio relativo alle espressioni mostrato a lezione. In particolare il parser dovrà costruire una rappresentazione del sorgente mediante un albero, con associata una symbol table. La generazione del codice avverrà a partire da tale albero.
- Non sono richiesti controlli in compilazione e in esecuzione relativamente ai range degli array.
- In caso di errore in compilazione l'applicazione può terminare l'esecuzione, fornendo un breve messaggio relativo al problema riscontrato. Un modo rudimentale per ottenere queste informazioni relative agli errori sintattici, consiste nell'inserire nel file di specifica sintattica di CUP il seguente codice:

```
parser code{:
  /* Ridefinizione del metodo che visualizza i messaggi di errore */
  public void unrecovered_syntax_error(Symbol cur_token)
                                       throws java.lang.Exception {
    Scanner sc = (Scanner) getScanner();  //riferimento all'analizzatore
                                          //lessicale in uso
    report_fatal_error("Errore di sintassi alla riga "  +
           sc.currentLineNumber() + " leggendo " + sc.yytext(), null);
    //numero della riga in esame e testo corrispondente al token corrente
  }
:}
```

  e nel file di specifica lessicale di JFlex (nella parte di opzioni e dichiarazioni) il seguente:

```
%{
  public int currentLineNumber() {
    return yyline + 1;
   }
%}
%line
```

Si deve consegnare:

1. una breve descrizione delle classi utilizzate e dell'organizzazione della symbol table;
2. tutti file sorgenti scritti per la risoluzione del problema (file di specifica lessicale, file di specifica sintattica, classi o altro codice scritto), in forma elettronica, con un indice degli stessi e con l'indicazione delle parti opzionali svolte;
3. alcuni esempi di compilazione *significativi* (sorgente e codice generato, *in forma leggibile*).

È possibile svolgere il progetto in gruppi di 2 o 3 persone: in tal caso sviluppate *almeno due* delle parti indicate come opzionali.

Il progetto è valido per l'anno accademico 2015/2016 e deve essere consegnato almeno dieci giorni prima della data concordata per la prova orale. (È necessario iscriversi tramite SIFA all'appello del mese in cui si presenta il progetto).