Linguaggi e Traduttori II

Progetto d'esame

Scopo del progetto è la costruzione di un compilatore per il linguaggio sorgente descritto di seguito.

Variabili e tipi

Il linguaggio prevede variabili di tipo intero e array di interi. Un identificatore di variabile è una sequenza di lettere e cifre che inizia con una lettera. Le variabili devono essere dichiarate *all'inizio del programma* precedute dal nome del tipo (`int` o `int[]`), utilizzando uno stile simile a quello di C e di Java. Esempi:

```
int pippo, x, y;
int[] numeri, z;
```

Le variabili `pippo, x, y` sono di tipo intero. Per esse il compilatore dovrà prevedere uno spazio in memoria durante l'esecuzione. Le variabili `numeri` e `z` sono, come nel linguaggio Java, *riferimenti* ad array di interi. Gli array potranno essere costruiti successivamente utilizzando l'operatore `new`. Ad esempio, l'istruzione

```
z = new[10];
```

crea un nuovo array di 10 interi e ne assegna il riferimento alla variabile `z`. Le posizioni negli array sono numerate a partire da zero, pertanto gli elementi dell'array così creato saranno accessibili come `z[0]`, `z[1]`, ..., `z[9]`. L'assegnamento

```
numeri = z;
```

copia in `numeri` il riferimento contenuto in `z`. Pertanto, `numeri` e `z` dopo l'assegnamento si riferiranno allo *stesso* array.

Come in Java è previsto il letterale `null`, utile per indicare che una variabile riferimento non si riferisce a nulla. Ad esempio, l'assegnamento

```
z = null;
```

farà in modo che `z` non si riferisca più all'array creato in precedenza (si noti che, nella sequenza di istruzioni fornite sopra, `numeri` continuerà a riferirsi ad esso).

Espressioni e condizioni

Espressioni intere

Il linguaggio prevede le usuali espressioni aritmetiche su interi, costruite a partire da variabili semplici[1] e costanti, con le operazioni di somma, sottrazione, moltiplicazione, divisione intera e resto della divisione (indicate rispettivamente con i simboli `+, -, *, /, %`), parentesi tonde. Ad esempio, un'espressione corretta secondo le dichiarazioni precedenti è `x+z[z[x-1]*2] /(pippo+y)`.

Espressioni riferimento

Il nome di una variabile array è un'espressione riferimento (come in Java). Inoltre la costante `null` è un'espressione riferimento.

[1] Una *variabile semplice* è una variabile di tipo intero o un elemento di un array di interi

Condizioni

Una espressione intera può essere interpretata come *condizione*, secondo quanto previsto dal linguaggio C: un valore diverso da zero indica "vero", mentre un valore uguale a zero indica "falso". Anche le espressioni riferimento possono essere interpretate come *condizioni*: il valore `null` indica "falso", gli altri valori indicano "vero". Ad esempio, utilizzando la variabile `z` dichiarata sopra si può scrivere `if (z) ....`

Le condizioni possono essere combinate tra loro, mediante gli operatori logici di congiunzione (`&&`), disgiunzione (`||`) e negazione (`!`), valutati utilizzando la *lazy evaluation*.

Per precedenze e associatività tra gli operatori del linguaggio si faccia riferimento alle regole di Java.

Istruzioni

- Assegnamento:

  ```
  <variabile> = <espressione>
  ```

  Ci deve essere compatibilità tra la variabile e l'espressione. Pertanto, nel caso di espressione intera la variabile deve essere semplice, nel caso di espressione riferimento la variabile deve essere array (vedere esempi precedenti).

- Selezione:

  ```
  if (<condizione>) <istruzione> else <istruzione>
  if (<condizione>) <istruzione>
  ```

- Iterazione:

  ```
  while (<condizione>) <istruzione>
  ```

  L'esecuzione dei cicli termina quando la condizione è falsa.

- Iterazione sugli array:

  ```
  for (<identificatore> : <variabile_array>)
    <istruzione>
  ```

  `<identificatore>` è un nome di variabile che viene dichiarata implicitamente ed è utilizzabile solo all'interno del ciclo; il ciclo viene eseguito assegnando alla variabile via via i valori contenuti nell'array. Ad esempio, alla fine del seguente ciclo la variabile `somma` (precedentemente dichiarata) conterrà la somma dei numeri presenti nell'array riferito da `numeri` (se l'array è vuoto, cioè di zero elementi, la variabile conterrà zero; se numeri contiene `null` dovrà essere fornito un messaggio d'errore e interrotta l'esecuzione):

  ```
  somma = 0;
  for (alfa: numeri)
    somma = somma + alfa;
  ```

- Istruzioni di lettura e scrittura:

  ```
  leggi <variabile_semplice>
  scrivi <variabile_semplice>
  scrivimsg <costante_stringa>
  scriviln <costante_stringa>
  ```

La prima istruzione legge da input un intero e lo assegna alla variabile indicata, la seconda scrive in output il valore contenuto nella variabile indicata, la terza e la quarta scrivono in output la `<costante_stringa>` indicata. La costante è una sequenza di caratteri tra virgolette. L'ultima istruzione di scrittura riporta il cursore a capo, le altre lasciano il cursore sulla stessa riga.

- Istruzione composta:
  È una sequenza (anche vuota) di istruzioni, racchiuse tra parentesi graffe aperte e chiuse (come in Java):

```
{
   <istruzione>;
   <istruzione>;
   ...
   <istruzione>;
}
```

Ciascuna istruzione è terminata da un punto e virgola.

Commenti

È possibile introdurre commenti secondo lo stile di C e Java (da \\ a fine riga o tra * e *\).

Programma

Un programma è una sequenza di dichiarazioni seguita da un'istruzione. Ecco alcuni esempi di programmi[2]:

```
/* Esempio 1: Hello, world! */
scriviln "Hello, world!";


/* Esempio 2: calcolo del massimo
   comun divisore mediante l'algorimo di
   Euclide
*/
int dividendo, divisore;
int resto;
{
   scrivimsg "dividendo? ";
   leggi dividendo;
   scrivimsg "divisore? ";
   leggi divisore;

   while (divisore) {
      resto = dividendo % divisore;
      dividendo = divisore;
      divisore = resto;
   };
   scrivimsg "valore del massimo comun divisore: ";
   scrivi dividendo;
```

[2]Come avviene 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.

```
   scriviln "";
   scriviln "Fine programma";
};


/* Esempio 3: somma di un array di interi */
int quanti, i;
int[] numeri;
int somma;
{
   scrivimsg "quanti numeri vuoi sommare? ";
   leggi quanti;

   numeri = new[quanti];

   i = 0;
   while (quanti - i) {
     scrivi "numero? ";
     leggi numeri[i];
     i = i + 1;
   };

   scrivimsg "Numeri letti: ";
   for (beta: numeri) {
     scrivi beta;
     scrivi " ";
   };
   scriviln ""; //ritorno a capo

   somma = 0;
   for (gamma: numeri)
     somma = somma + gamma;

   scrivimsg "La somma vale ";
   scrivi somma;
   scriviln "";
};


/* Esempio 4: il crivello di Eratostene
   (calcolo numeri primi)
*/

int[] primi; //boolean[] primi;
int nMax;
int x, numero, multiplo;

{
  scrimsg "Numero massimo da considerare? ";
  leggi nMax;

  //creazione e inizializzazione array
```

```
  primi = new[nMax + 2];
  x = 2;
  while (nMax / x) {  //  x <= nMax
    primi[x] = 1111; //assegna true
    x = x + 1;
  };

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

  scrivimsg "Elenco primi: ";
  numero = 2;
  while (nMax / numero) {
    if (primi[numero]) {
      scrivi numero;
      scrivimsg " ";
    };
    numero = numero + 1;
  };
  scriviln "";
};
```

Parti facoltative

Operatori di confronto

Introdurre gli operatori di confronto `!=`, `==`, `<`, `<=`, ecc. tra interi e `!=`, `==` tra riferimenti.

Tipo boolean

Introdurre un tipo `boolean` con due letterali `true` e `false`. Deve essere possibile definire variabili e array di tipo `boolean`. Le condizioni sono espressioni di tipo `boolean`. Se si implementa questa parte, si adotti la semantica del linguaggio Java che, a differenza del C, *non* permette di interpretare interi e riferimenti come condizioni.

Cosa si richiede

- Scrivere una grammatica context–free che specifichi la sintassi del linguaggio.
- Scrivere un analizzatore lessicale e un analizzatore sintattico per il linguaggio, servendosi degli strumenti presentati a lezione.
- Scrivere una classe di prova per l'analizzatore lessicale che elenchi i token man mano inseriti.

- Scrivere 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.[3]
- Si suggerisce di ispirarsi all'esempio relativo alle espressioni mostrato a lezione. In particolare il parser (generato utilizzando CUP) 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 all'accesso agli array.
- Non è richiesta la garbage collection.
- In caso di errore in compilazione l'applicazione può terminare l'esecuzione (se possibile fornire un messaggio che indichi dove è stato riscontrato l'errore).

Si deve consegnare:

1. una descrizione della grammatica del linguaggio, delle classi utilizzate e dell'organizzazione della symbol table;
2. una stampa del file di specifica lessicale e del file di specifica sintattica;
3. una stampa dei sorgenti Java scritti per la risoluzione del problema;
4. una stampa di alcuni esempi di compilazione significativi (sorgente e codice generato);
5. i file sorgenti scritto per la risoluzione del problema, in forma elettronica, con un indice degli stessi.

Il progetto è valido sino a luglio 2007 e deve essere consegnato dieci giorni prima della data concordata per la prova orale[4].

[3] Anziché generare codive per `Macchina.java` è possibile generare bytecode per la Java Virtual Machine o altre forme di codice intermedio per le quali si disponga di un interprete.

[4] È necesssario iscriversi tramite SIFA all'appello di giugno o di luglio.