SPECIFICHE DEL PROGETTO Sapy (Sapienza BASIC) v1.0
Metodologie di Programmazione A.A. 2012-2013
Corso di Laurea Triennale in Informatica
Proff. Roberto Navigli e Francesco Parisi Presicce

Il progetto consiste nell'implementazione in Java di un interprete di un dialetto BASIC chiamato Sapy v1.0 (o, per esteso, Sapienza BASIC v1.0). L'interprete, implementato mediante la classe **Sapy**, prende in ingresso un file di testo .sapy e lo interpreta eseguendone le istruzioni.

Il progetto può essere svolto in team da 1 o 2 studenti (la parte in giallo riguarda solo i team da 2 studenti e comprende la realizzazione di una interfaccia a finestre per l'utilizzo del sistema).

Sintassi di Sapy

--- Tipi di dati

Sapy prevede tre tipi di dati primitivi: intero, stringa e booleano. I letterali interi sono espressi come sequenze di cifre (ad es. 2, 42, 1024, ecc.), le stringhe sono sequenze di caratteri racchiusa da virgolette (ad es. "ciao", "12e4", ecc.), i letterali booleani possono essere solo i valori **true** e **false**.

--- Operatori aritmetici, booleani e di confronto

Sapy dispone dei seguenti operatori:

- operatori aritmetici binari tra interi di somma (+), sottrazione (-), prodotto (*), divisione (/) e resto (%) e dell'operatore unario di negazione (-)
- operatori booleani di AND, OR e NOT
- operatori di confronto: uguale (=), diverso (<>), minore (<), minore uguale (<=), maggiore (>) e maggiore uguale (>=)

--- Espressioni

Per chi aspira a **un voto per il progetto ≤ 28**, le espressioni possono seguire la seguente definizione non ricorsiva[1]:

```
bool_exp  = [ "NOT" ] bterm | bterm , ( "AND" | "OR" ) , bterm | bool_cmp ;
bool_cmp = term , ( "=" | "<>" | "<" | ">" | "<=" | ">=") , term ;
```

[1] La notazione utilizzata è EBNF (Extended Backus-Naur Form), una meta-sintassi per esprimere la grammatica di un linguaggio. L'operatore | indica l'alternativa. L'operatore [] indica un'opzione (ad esempio ["-"] signifca o nulla o "-"), l'operatore { } indica la ripetizione 0, 1 o più volte (ad esempio {"x"} indica nulla, "x" oppure "x" "x", "x" "x" "x" ecc. La virgola "," concatena, il punto e virgola ";" termina una regola.

```
bterm      = variabile | “TRUE” | “FALSE” ;
int_exp    = [“-”] , iterm | iterm , ( “+” | “-” | “*” | “/” | “%” ) , iterm ;
iterm      = variabile | numero_intero ;
string_exp = variabile | stringa ;
term       = variabile | numero_intero | stringa | “TRUE” | “FALSE” ;
```

La definizione di sopra ammette come espressioni booleane, ad esempio “NOT $x”, “NOT TRUE”, “NOT $x AND NOT $y”, ecc. Tuttavia la definizione di sopra non permette di utilizzare più operatori binari. Ad esempio, non permette di riconoscere l’espressione “x OR y AND z”. **Per poter raggiungere il voto massimo**, è richiesto di gestire espressioni matematiche e booleane ricorsive, implementandole secondo la seguente precedenza degli operatori e includendo anche le parentesi. Si deve cioè seguire la seguente definizione ricorsiva:

```
bool_exp  = [ “NOT” ] , bterm | bterm , { ( “AND” | “OR” ) , bterm } | bool_cmp ;
bool_cmp = term , ( “=” | “<>” ) , term | iterm , ( “<” | “>” | “<=” | “>=” ) , iterm ;
bterm      = variabile | “TRUE” | “FALSE” ;
int_exp    = mul_term , { ( “*” | “/” | “%” ) , mul_term}  ;
mul_term  = iterm , { ( “+” | “-” ) , iterm} ;
iterm      = variabile | numero_intero | “-” , iterm ;
string_exp = variabile | stringa ;
term       = variabile | numero_intero | stringa | “TRUE” | “FALSE” ;
```

Si consiglia di utilizzare un parser ricorsivo discendente per gestire la precedenza degli operatori. Si può trovare un esempio al seguente link:

http://stackoverflow.com/questions/2093138/what-is-the-algorithm-for-parsing-expressions-in-infix-notation

Controllo dei tipi: in fase di esecuzione, l’espressione sarà in grado di determinare il proprio valore. Se ciò non fosse possibile per via di tipi incompatibili (es. `“a”+ 5`, oppure `3 * 5 - true`) l’interprete dovrà emettere un errore appropriato.

--- Variabili e operazione = di assegnazione

Le variabili in Sapy sono identificate da una sequenza alfanumerica (incluso l’underscore) **sempre** preceduta dal simbolo dollaro. L’assegnazione del valore di un’espressione a una variabile avviene mediante l’operatore =. Alcuni esempi:

```
$x = 5 + 3
$mia_stringa = “ciao ciao”
$bool = TRUE
$x = “stringa”
```

Al contrario di Java (che ne implementa solo qualche meccanismo), Sapy è un linguaggio a *tipaggio dinamico*, ovvero il tipo della variabile viene stabilito al momento dell'esecuzione sulla base di ciò che viene ad essa assegnato. La stessa variabile può essere utilizzata per memorizzare valori di tipi differenti (si veda l'esempio di $x mostrato sopra, prima valorizzato con un intero e poi con una stringa).

Si noti che l'operatore = è ambiguo: può essere sia un operatore di confronto (all'interno di un'espressione) sia un operatore di assegnazione. Tuttavia questo non creerà problemi in fase di analisi sintattica, poiché il token assume significato diverso sulla base della sua posizione.

--- Istruzioni I/O

Sapy dispone di due istruzioni fondamentali per l'input/output su e da console:

```
PRINT <espressione>
```

che stampa a video il valore dell'espressione. Ad esempio:

```
$x = 6
PRINT $x + 5
PRINT "ciao"
PRINT ( 5 * 3 ) / 2
```

stampano a video:

```
11
ciao
7
```

La seconda istruzione è:

```
INPUT <variabile>
```

che invece prende in input una stringa da tastiera e la memorizza all'interno della variabile specificata. Ad esempio:

```
INPUT $x
```

richiede un input da console e ne memorizza la stringa nella variabile $x.

--- Costrutti di controllo: IF, FOR, WHILE

Il costrutto **IF** esegue le istruzioni che seguono la parola chiave **THEN** se l'espressione booleana è vera:

```
IF espressione_booleana THEN istruzioni ENDIF
```

Se viene specificata la parola chiave **ELSE**, se l'espressione è falsa vengono eseguite le istruzioni che seguono tale parola chiave.

```
IF espressione_booleana THEN istruzioni ELSE istruzioni ENDIF
```

In entrambi i casi il costrutto si chiude con la parola chiave **ENDIF**.

Il costrutto **FOR** permette di iterare le istruzioni sul valore di una variabile $var incrementandone (o decrementandone) il valore da x a y:

```
FOR $var = x TO y DO istruzioni NEXT
```

dove x e y sono valori numerici (interi). Ad esempio:

```
FOR $x = 1 TO 3 DO PRINT $x NEXT
```

stampa:

```
1
2
3
```

mentre:

```
FOR $x = 3 TO 1 DO PRINT $x NEXT
```

stampa:

```
3
2
1
```

Se si vuole modificare il valore di $var con un passo diverso da 1 o -1 è necessario utilizzare la parola chiave **STEP**:

```
FOR $var = x TO y STEP z DO corpo_istruzioni NEXT
```

dove z è un valore numerico (intero) che specifica l'incremento (o il decremento) di $var a ogni iterazione. Ad esempio:

```
FOR $x = 5 TO 1 STEP -2 DO PRINT $x NEXT
```

stampa:

```
5
3
1
```

Il costrutto **WHILE** (analogo a quello in Java) ha la seguente sintassi:

```
WHILE condizione_booleana DO istruzioni NEXT
```

Commenti

I commenti (da non eseguire) vengono specificati mediante la parola chiave REM all'inizio della riga. Ad esempio:

```
REM il mio commento
```

Organizzazione del codice

Le singole istruzioni (tranne per i costrutti iterativi e condizionali, la cui fine è determinata dalle apposite parole chiave **ENDIF** e **NEXT**) terminano con la fine di una riga. Ad esempio, non è possibile scrivere:

```
PRINT
$x
```

oppure:

```
PRINT 5 +
 3
```

Tuttavia è possibile specificare più istruzioni su una stessa riga separandole con il simbolo **:**. Ad esempio:

```
PRINT $x : INPUT $y : PRINT $y
FOR $x = 1 TO 10 DO PRINT $x : $y = $y - $x NEXT
FOR $x = 1 TO 10 DO
      PRINT $x : $y = $y - $x
NEXT
```

--- Fine del programma

Il programma termina quando si incontra l'istruzione **END** oppure quando si è eseguita l'ultima riga del programma stesso.

--- Array

E' possibile dichiarare un array con il comando **DIM** seguito dal nome dell'array seguito dalla dimensione dell'array tra parentesi tonde:

```
DIM $mio_array(100)
```

Un elemento dell'array può essere acceduto mediante la sintassi:

```
$nome_array(posizione)
```

dove posizione va da 1 alla dimensione dell'array. Ad esempio, il seguente codice salva i primi 10 elementi della successione di Fibonacci nell'array $a:

```
DIM $a(10)
FOR $x = 1 TO 10 DO
IF $x = 1 OR $x = 2
THEN $a($x) = $x
ELSE $a($x) = $a($x-1) + $a($x-2)
NEXT
```

Svolgimento del progetto

Il progetto deve essere realizzato in modo modulare. In particolare, saranno valutati come moduli separati:

Analizzatore lessicale (Lexer)

Progettare un analizzatore lessicale (detto: lexer) implementato mediante una classe **it.uniroma1.sapy.lexer.Lexer** che, dato in input il programma sorgente sotto forma di stringa, costruisca una lista di token. Ad esempio, dato il programma:

```
10 PRINT 5 + 3
20 FOR $x = 1 TO 3 DO $j = $x - 1 : PRINT $j NEXT
30 PRINT "4" : $k = TRUE
```

l'analizzatore otterrà una lista di token del tipo:

```
INTERO,PRINT,INTERO,PIU,INTERO,EOL,INTERO,FOR,VARIABILE,UGUALE,IN
TERO,TO,INTERO,DO,VARIABILE,UGUALE,VARIABILE,MENO,INTERO,DUEPUNTI
,PRINT,VARIABILE,NEXT,EOL,INTERO,PRINT,STRINGA,DUEPUNTI,VARIABILE
,UGUALE,BOOLEANO,EOL
```

La classe **Lexer** può essere eseguita da console mediante il comando:

```
java it.uniroma1.sapy.lexer.Lexer nome_file.sapy
```

e restituisce a video l'elenco di token presenti nel programma sorgente fornito in input. Ovviamente la classe fornisce anche un supporto programmatico (ovvero un metodo che restituisce la lista dei token, invece che stamparla a video) per il modulo successivo, ovvero l'analizzatore sintattico.

L'analizzatore lessicale deve essere eseguibile anche da finestra, premendo un pulsante che chiede di inserire il nome del file sorgente. L'elenco dei token presenti nel programma viene in questo caso visualizzato in una nuova finestra.

Potete assumere che il separatore dei token sia lo spazio. In alternativa **(+1 punto extra)** potete effettuare una tokenizzazione più sofisticata che permette anche di non separare con spazio gli operatori dagli operandi. Ad esempio, in questo caso PRINT 5+3 fornisce comunque la lista PRINT,INTERO,PIU,TRE, anche se non c'è uno spazio tra 5 e + e tra + e 3.
Sarà fornito l'elenco dei possibili token di Sapy e un certo numero di esempi su cui sarà testato il Lexer. **Se il programma non supera i test, il progetto non ottiene la sufficienza.** Altri test aggiuntivi saranno svolti durante la correzione, che tuttavia influiranno solo sul voto complessivo, non compromettendone irreparabilmente il successo.

Analizzatore sintattico (Parser)

Implementato tramite la classe **it.uniroma1.sapy.parsing.Parser** che, costruita con una lista di token (normalmente, ma non necessariamente, forniti dalla classe **Lexer**), fornisce funzionalità per l'interpretazione della sequenza e la sue trasformazione in un **it.uniroma1.sapy.runtime.ProgrammaEseguibile**, visto come una lista di istruzioni e che può essere eseguito mediante un metodo **esegui**. Ogni **it.uniroma1.sapy.runtime.istruzioni.Istruzione**, creata dal Parser, è a sua volta in grado di eseguirsi mediante un metodo **esegui**.

Il **ProgrammaEseguibile** mette a disposizione la funzionalità di serializzazione su file. Questo permette, dato un nome di file in input nome_file.sapy, di salvare un file nome_file.sapycode e ricaricarlo in seguito direttamente in memoria, pronto per l'esecuzione.

Interprete

L'interprete **it.uniroma1.sapy.Interprete**, dato in input un sorgente sotto forma di stringa, richiama il **Parser**, ottiene la lista di istruzioni e le esegue una per una.
Anche per l'interprete e per il Sapy (vedi paragrafo successivo), l'interazione deve avvenire anche tramite finestra.

Sapy

La classe **it.uniroma1.sapy.Sapy**, infine, è il punto di partenza del progetto completo, permettendo di effettuare le seguenti operazioni:

- Interpretazione di un file .sapy:

  ```
  java it.uniroma1.sapy.Sapy nome_file.sapy
  ```

- Parsing del file .sapy e serializzazione su file (senza interpretazione):

  ```
  java it.uniroma1.sapy.Sapy --save nome_file.sapy
  ```

- Interpretazione di un file .sapycode (ovvero, precedentemente serializzato mediante l'opzione --save):

  ```
  java it.uniroma1.sapy.Sapy nome_file.sapycode
  ```

La prima (per team di 1 studente) o tutte e tre (per team di 2 studenti) le operazioni saranno eseguite su un certo numero di programmi Sapy forniti agli studenti e su programmi aggiuntivi che non saranno resi disponibili.

Estensibilità (+1 punto extra se realizzata da 1 studente, obbligatoria per gruppi di 2)

Se questa funzionalità viene implementata, in Sapy è possibile aggiungere nuove istruzioni senza dover modificare il codice esistente. Ad esempio, per aggiungere la funzione INT (che converte un'espressione in intero) sarà sufficiente implementare la classe **it.uniroma1.sapy.runtime.istruzioni.Int**, che estenderà la classe **Istruzione** e aggiungere la classe nel classpath della JVM. A quel punto, senza modificare nient'altro, si potrà eseguire un programma del tipo:

```
INPUT $x
$x = INT $x + 5
PRINT $x
```

Analogamente per qualsiasi altra istruzione. Ad esempio, l'istruzione ABS calcola il valore assoluto di un'espressione intera:

```
PRINT ABS -3
```

Modalità di consegna del progetto

Per consegnare correttamente il progetto è necessario consegnare un file **.zip** (o **tar.gz, NON SARANNO ACCETTATI FILE .rar**) contenente i seguenti file:

- **sapy.jar**: questo file contiene i sorgenti .java, i file compilati .class e le librerie necessarie all'esecuzione del progetto mediante l'istruzione:

  ```
  java -jar sapy.jar
  ```

  IMPORTANTISSIMO: Molti di voi, nel generare il file .jar, consciamente o inconsciamente, non includono i sorgenti. Per essere certi che il file .jar contenga **TUTTI i sorgenti**, scompattatelo, per esempio con "unzip nome_vostro_progetto.jar". **NON SARANNO VALUTATI PROGETTI I CUI FILE .JAR NON CONTENGANO I SORGENTI.**

- **sapy.pdf**: breve relazione del progetto (3-4 pagine) in cui vengono esplicitate e giustificate le scelte progettuali e implementative. Tale relazione deve contenere un diagramma schematizzato delle classi (non è necessario includere tutte le istruzioni), senza la specifica di metodi e attributi;
- **javadoc**: il javadoc generato per le classi del progetto.

Plagio

Nel caso di plagio accertato, dal Web o da colleghi di corso, **non sarà consentito di consegnare nuovamente il progetto** in questo A.A., il che implicherà dover sostenere nuovamente anche la prova scritta nell'anno seguente. **Anche chi ha permesso il plagio (per esempio fornendo il proprio codice) è passibile di azioni disciplinari analoghe.**