Formal Languages and Compilers
Proff. Breveglieri, Morzenti
Written exam[1]: laboratory question
05/03/2014

SURNAME: .....
NAME: ..... Student ID: .....
Course: ○ Laurea Specialistica ○ V. O. ○ Laurea Triennale ○ Other: ...
Instructor: ○ Prof. Breveglieri ○ Prof Morzenti

The laboratory question must be answered taking into account the implementation of the `Acse` compiler given with the exam text.

Modify the specification of the lexical analyser (`flex` input) and the syntactic analyser (`bison` input) and any other source file required to extend the `Lance` language with the ability to handle the `map` and `reduce constructs`. Consider the following code snippet as an **illustrative example** of their use:

```
int vett[100];
int elem, t, sum;

map elem on vett as {
  t = elem * elem;
  t = t + 2 * elem;
  elem = t - 9;
}

read(t);
sum = 0;

reduce elem into sum
  as [[ sum + t * elem ]]
  on vett;

write(sum);
```

The `map` construct is used to apply in-place a transformation to the elements of an array (named `vett` in the example). For each element of the array, the code block representing the transformation, which is suffixed to the `as` keyword is executed and finally the processed array element is written back to its location. The `reduce` construct is used to apply a function reducing the elements of an array to a single scalar (e.g. computing their average). The result of the reduction construct, held in the support variable specified after the `into` keyword, is updated computing the reduction expression, enclosed between double square braces, for each element of the array.

[1] Time 60'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

```
int elem;
int vett[3];

// vett = {2, -10, 9}

map elem on vett as
  elem = elem * 2;

// now vett = {4, -20, 18}
```

Figura 1: Using `map` to double the elements of an array.

Figure 1 shows a simple example of the `map` construct which doubles every element of the array `vett`.

```
int elem, sum = 0;
int vett[3];

// vett = {2, 6, 13}

reduce elem into sum as
  [[ sum + elem ]] on vett;

// sum = 21
write(sum);
```

Figura 2: Using `reduce` to compute the sum of the elements of an array.

Figure 2 shows an example of the `reduce` construct which sums all the elements of the array `vett` into the scalar `sum`.

1. Define the tokens (and the related declarations in **Acse.lex** and **Acse.y**). (2 points)
2. Define the syntactic rules or the modifications required to the existing ones. (3 points)
3. Define the semantic actions needed to implement the `map` statement. (10 points)
4. Define the semantic actions needed to implement the `reduce` statement. (15 points)

5. (**Bonus**) Describe how it is possible to extend the `reduce` construct to allow a variant integrating a map construct instead of the array to be reduced. The reduction should be performed on the array after the `map` construct is run. An example of the syntax follows:

```
int vett[100];
int elem, t,  sum = 0;
reduce elem into sum
  as [[ sum + elem ]]
  on map vett as {
    t = elem * elem;
    elem = 2 + elem;
  }
```