Formal Languages and Compilers
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Written exam[1]: laboratory question
06/02/2009

SURNAME: .....
NAME: ..... Student ID: .....
Course: ○ Laurea Specialistica ○ V. O. ○ Laurea Triennale ○ Other:.....
Instructor: ○ Prof. Breveglieri ○ Prof. Crespi ○ 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 analyzer (`flex` input) and the syntactic analyzer (`bison` input) and any other source file required to extend the `Lance` language with the ability to *handle simple macros* resembling `#define` construct for the C preprocessor :

```
define ANSWER 42;
define QUESTION 9;
int x;
read( x );
x = ANSWER * x;
write( x );
```

The first line of the sample code snippet defines the macro `ANSWER` as the integer value `42`. The expected behaviour of this program is to print the number provided in input multiplied by 42.

The solution needs to comply to the following specifications:

- The macros are parameter-free and each macro can only bind to a single *integer value*
- An arbitrary number of macros may be defined
- Macros may be employed *everywhere* an integer constant can be used in the original `Acse` language
- Macros cannot be modified by assignments

An **optimal solution** shouldn't generate any additional assembly code for a `Lance` program using macros with respect to one which isn't.

In case a double definition of the same macro is detected, implement a sensible strategy (either ignore or overwrite the former definition).

[1]Time 45'. Textbooks and notes can be used.
Pencil writing is allowed. Write your name on any additional sheet.

You may specify any further (sensible) assumption useful to complete the given specification list.

You may use the functions in `collections.h` if you are in need of a set of ready-made helpers for dealing with lists. If you prefer, you may also use these functions in order to handle lists of `DATA` typed elements. In the latter case, define the structure `DATA`.

```
void initList( t_list *list );
void addFirst( t_list *list, DATA *element );
void addLast( t_list *list, DATA *element );
DATA *getFirstElement( t_list *list );
DATA *getLastElement( t_list *list );
DATA *getElementAt( t_list *list, unsigned int position );
```

1. Define the tokens (and the related declarations in Acse.lex e Acse.y). (3 points)

2. Define the syntactic rules or the modifications required to the existing ones. (8 points)

3. Define the semantic actions needed to implement macros. (13 points for a working solution, 19 for the optimal one)

4. Bonus: Modify the existing solution in order to allow recursive definition of macros; for instance, the following program

```
define ULTIMATE_ANSWER  42;
define ANSWER  ULTIMATE_ANSWER;
int x;
x = ANSWER;
write( x );
```

prints “42”. (5 points)