Formal Languages and Compilers
Proff. Breveglieri, Crespi Reghizzi, Morzenti
Written exam[1]: laboratory question - ACSE
05/03/2008

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 a new construct *switch* resembling the one in the following sample.

```
int a;
...
switch (a) {
      case 0:
            ...
            break;
      case 1:
            ...     /* Code block without any "break" statement */
      case 2:
            ...
            break;
      default:
            ...
}
...
```

The `switch` construct is built out of an arbitrary number of `case` blocks and may include an optional `default` block. The `switch` construct has the following semantics: the value of the variable between round brackets is checked for a match against the values indicated in the `case` statements. On a successful match, the corresponding code block is executed. The `break` statement brings the execution to the first instruction *after* the `switch` construct. In case a code block does not contain any `break` statement, the next `case` block (in declaration order) is executed.

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

If there are no more `case` blocks left, the code must exit the `switch` construct. If there is no match between the switch variable and the case values, the execution flow jumps to the `default` block (if present) or exits from the `switch` construct. At least a `case` block has to be declared. `Switch` blocks may be nested.

For instance: if we assign the value 0 to the variable `a` in the example, only the first code block (the one corresponding to `case 0`) will be ran. In case `a` evaluates as 1, the blocks related to both `case 1` and `case 2` will be executed. In case the value of `a` differs from $0, 1, 2$ the execution jumps to the `default` block.

Your modifications have to allow the `Acse` compiler to both correctly analyze the syntactical correctness of the aforementioned constructs and to generate a correct translation in the `Mace` assembly language.

1. Define the tokens and the `Acse.lex` and `Acse.y` declarations needed to achieve the required functionality. (3 points)

2. Define the syntactic rules needed to achieve the required functionality. (8 points)

3. Define the semantic actions needed to achieve the required functionality, without considering the `break` and `default` statements. (10 points)

4. Define the semantic actions needed to handle the optional `default` block. (6 points)

5. Define the semantic actions needed to handle the `break` construct. (6 points)