University of Malta

CSM 201 – Compiling Techniques
Course Assignment 2001-2002

Department of Computer Science and A. I.
University of Malta.
Tutor: Kristian Guillaumier
Email: `kguil@cs.um.edu.mt`

An Evaluator for Arithmetic Expressions

- Design and implement an program that evaluate arithmetic expressions entered by the user. The arithmetic expressions are specified by the following BNF grammar:

```
expression    →    <primary> {<op> <expression>}
primary       →    <identifier>
                   | <integer>
                   | <floating>
                   | '(' <expression> ')'
<op>          →    + | - | * | / | \
```

- The operators allowed in the expressions are addition, subtraction, multiplication, floating point division, and integer division.
- Identifiers in the expression represent '*placeholders*' for integer or floating-point numbers. When interpreting the expression `pi*r*2`, the program must ask the user to enter the values for `pi` and `r` before proceeding to evaluate. The program must determine whether the value entered by the user for the identifiers are valid numbers and their type, i.e. integer or floating point.
- The interpreter will consist of a *lexical analyzer*, a *parser*, and an *evaluator*. The lexical analyser reads the arithmetic expression and outputs a list of tokens. The parser accepts as input the list of tokens and builds the *parse tree*. The parse tree is built using the *operator precedence* parsing algorithm described in class. The parse tree will then be used by the evaluator to derive the result of the expression. The student is responsible for the design of the evaluator.
- Operator precedence is defined as follows:

  Multiplication and division are higher than addition and subtraction.
- Brackets may be used to emphasise precedence.
- All operators are left-associative.
- The program should produce a trace file, which is a file describing all the intermediate steps during the interpreting and evaluation. An example of such a file could be:

```
Trace File
----------

Expression: (a+b)*2/1.2

Token List:
(     [TOKEN_OPENBKT ]
a     [TOKEN_IDENT   ]
+     [TOKEN_ADD     ]
b     [TOKEN_IDENT   ]
)     [TOKEN_CLOSEBKT]
*     [TOKEN_TIMES   ]
2     [TOKEN_INTEGER ]
/     [TOKEN_DIV     ]
1.2   [TOKEN_FLOATING]

User Variables:
a = 2.2 (Floating)
b = 3 (Integer)

Parse Tree:
   [NODE_DIV]
      [NODE_TIMES]
            [NODE_PLUS]
                  2.2 [NODE_FLOATING]
                        [NULL]
                        [NULL]
                  3   [NODE_INTEGER]
                        [NULL]
                        [NULL]
            2  [NODE_INTEGER]
                  [NULL]
                  [NULL]
      1.2  [NODE_FLOATING]
            [NULL]
            [NULL]

Evaluation: 8.6666
```

- The program must produce meaningful error messages if any occur.
- The program must be accompanied by a technical report describing any implementation details and techniques used to complete the assignment.
- Refer to the assignment instructions at **http://www.cs.um.edu.mt/~kguil/assignment.html**