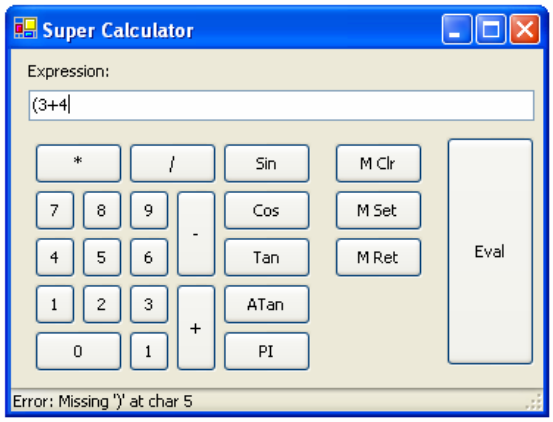
University of Malta

CSA 2010 – Compiling Techniques
Course Assignment 2004-2005

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

Creating an Expression Calculator

- The aim of this assignment is to create an application where expressions (that may include identifiers/variables) are evaluated.
- The design of the application must resemble a calculator:



- This assignment involves:
  - Creating a lexical analyzer for the expression language (tokens include identifiers, integer constants, floating point constants, arithmetic operators like + - * and /, keyword constants like PI and keyword operators like TAN, SIN and COS).
  - Creating and managing a symbol table (probably a HashTable implementation).
  - Creating a top-down predictive parser for the expression language. The BNF for the expression language should be an augmented version of that found in the lecture slides. The parser should build a binary tree.
  - Simple error reporting.
  - A user interface similar to the one shown above.
- An expression may include (and should support) identifiers. E.g. `3+counter-limit`. Whenever an identifier is encountered it should be placed in the symbol table if it is not already there (probably during scanning). The initial value of the identifier is *null*. When the expression has to be evaluated a popup input box will be displayed for each identifier having a *null* value for the user to supply the runtime value.

- The expression should be evaluated by calling a evaluation function passing the root note of the parse tree as an argument (as demonstrated in class). The function will need to be called recursively. Note that during evaluation if an identifier with a *null* value is encountered, the popup box must be displayed to acquire the actual value. Checking must be made to ensure that the value entered in the popup box is a valid integer or floating point constant.
- Operator precedence is informally defined as follows:

  Multiplication and division are higher than addition and subtraction.
- Brackets may be used to emphasize precedence. Brackets should be supported in the language.
- All operators are left-associative.
- Integer constants are 32-bit integers whilst floating point constants are 32-bit floating point numbers.
- The expression language is case-insensitive (for identifiers).
- Variable names/identifiers start with a letter or underscore symbol followed by any number of letters, underscore symbols or digits.
- The application must support a basic form of *memory clear, memory set* and *memory recall* functionality. Memory can be recalled from within the identifier value popup box.
- Whenever the value of the expression in the textbox changes the entered expression must be checked for errors and the error (if any) displayed in some sort of status area. This will involve capturing some sort of *TextChanged* event for the expression text box. Timers or threads may be used to improve responsiveness but their use is purely optional.
- The program must be accompanied by a technical report describing any implementation details and techniques used to complete the assignment.
- The technical report *must* include the BNF for the expression grammar as augmented from the lecture slides.
- (Important) Refer to the assignment instructions at **`http://webster.cs.um.edu.mt/kguil/assignment.html`**