Object Oriented Software Design

Final Assignment

Giuseppe Lipari
`http://retis.sssup.it/~lipari`

Scuola Superiore Sant'Anna – Pisa

October 14, 2010

Final exam

- For IMCNE and MAPNET students:
  - the final exam will consist of two parts
  - The first part consists of developing a program of medium complexity
    - The program must be developed in a small group of maximum 3 persons
    - The objective of this assignment is to demonstrate your ability in designing and coding a program
    - The program must be coded in Java
  - The second part is an oral examination
    - The oral examination is strictly individual
    - The objective of the oral examination is to investigate your personal knowledge in Java, C++ and OO design

Specification of the assignment

- The assignment consists in writing a simple program to evaluate numerical expressions and function and perform calculations
- The assignment is divided in two successive steps
  - In the first step, the program performs only simple numerical operations
  - In the second step, the user can define functions of one or more variables
- For each step, there are mandatory things to be done, and optional things
  - It's fine to only do the mandatory parts;
  - groups that do the optional parts get an higher grade.

First Step

- The program is only text-based and input is read from the terminal
- The user can enter expressions like:

```
2.5 * 3
```

- the program will output the result of the multiplication (7.5 in this case) as soon as the user press <enter>
- These are simple numerical expressions
  - supported operations are $+$, $-$, $*$, $/$
  - it is possible to use parenthesis to enforce precedence, otherwise the normal operator precedence must be enforced

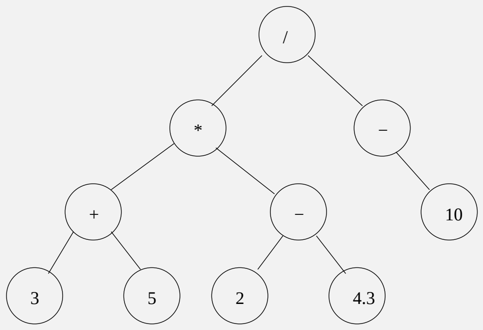
```
(1.5 + 1) * 3
>>> ans : 7.5
1.5 + 1 * 3
>>> ans : 4.5
```

- **Optionally:**
  - the program understands also standard math functions as exp(), log(), sin(), cos()

First step

- What's behind it?
  - A numerical expression can be represented as a tree

```
(3+5) * (2-4.3) / (-10)
```



- Going from a textual representation (a string in input) to a tree that represents the expression is called **parsing**
  - Your **parser** should be able to catch errors in the input (e.g. two numbers without operation, a multiplication with only one operand, etc.)
  - Once the tree is built, computing the result is easy

First step deadline

- The first step should be handed over before **November 26**
  - The earlier, the better
  - Delays will be taken into account in the final marks!

Second step

- In the second step, the program accepts *function definitions*
  - For a function definition, no result is given
  - the function can be used in future expressions

```
def add(x,y)=(x+y)
>>> ok
def inv(x)=1/x
>>> ok
inv(add(2,2))
>>> ans : 0.25
```

- Four additional commands:
  - **show** prints all definitions on screen
  - **save** *filename* saves all definitions in a file
  - **load** *filename* loads all definitions from a file
  - **delete** *funname* deletes the definition corresponding to the function named funname

Second step

- A function is a *labelled* tree
  - It is a tree with a name
  - It also contains *unbound* variables, i.e. the variables listed in the parameter list (and only those ones)
  - When a function is used, it is necessary to bound the variables to actual values, and then evaluate the tree as any other numerical expression
- **Optionally**:
  - Allow the definition of *projections*, i.e. partially bound functions, like these ones

```
def addtofive(x) = add(x,5)
def invsum(x,y)=inv(add(x,y))
def fracsum(x,y,z,w)=add(x,y)/add(z,w)
```

Second step deadline

- The second step has to be handed over one week before the oral examination
  - The earlier, the better
- The oral examination will include an analysis of your program, with specific questions on design choices, and parts of the code

When to do the assignment

- Part of it will be done during the labs
  - The rest you have to do by yourself
- Don't start to immediately write code!
- First, use paper and pencil to do the design
  - Divide the work into logically interacting modules (e.g. the tree implementation, the tree evaluator, the parser that builds the tree, the commands, etc.)
  - Establish the interfaces of the existing modules first (which functions each module export)
    - You may need to refine the initial design, modifying the interfaces, but do not worry too much about that now
  - Divide the implementation work among the members of the group
  - Test the modules individually
  - Integrate everything in one single program, and do integration testing