Programming Techniques Project, Spring 2016
Application 3: Interpreter for mathematical expressions

Abu-Ras Mohamed Ata Radu and Calina Constantin Alin

Computer Science English Section First Year Group 10105A

1 Problem Statement

Write an interpreter for a simple language that recognizes both neg-ative and positive integer constants, variable assignment and the basic mathematical operators: $+$, $-$, $*$, / and for exponentiation. Once read the expressionsshould be represented as binary trees which will be used to compute the result.The interpreter should present the user with a Read Eval Print Loop (REPL).

Here is the explanation of how the imput can be given:

1. You may enter any variable name followed by "=" and an integer to declare it
2. You may reinitialize a variable whenever you want
3. You may simply enter a mathematical expression using integers and basic operators
4. You may enter a mathematical expression using both integers and the variables previously declared

An example session in the interpreter would look like this:

> a = 6

- 6

> b = 2

- 2

> a2 + 2*b + 2

- 42

2 Pseudocode

3 Application design

3.1 The high level architectural overview of the application:

- The application uses 4 main functions to resolve the given task and a number of other auxiliary functions that are called by the main ones in order to reduce the ammount of written code and make them more organised and structed. Such auxiliary functions are : declarevar , push , priority , isoperator .

- The 4 main functions mentioned above are the following : infix2postfix , maketree , evaluatetree , read :
  - infix2postfix : this function takes the mathematical expression as given and translates it into a postfix form which is needed later . This form is also known as the polish form .
  - maketree : the function that creates a binary tree using the polish form created by infix2postfix . The tree's structure is based on a few characteristics :
    - wheter the value of a node in the tree is a mathematical operator or a variable(or simply a given integer)
    - the priority of the mathematical operators one to another
  - evaluatetree : this function is used to compute the expression and simply returns an integer
  - read : this is the function that uses the other ones and resolves the following tasks : reads the expression , removes excess spaces , transforms the expesion into a postfix order , makes a binary tree based off that order and then evaluates the expression .

3.2 The specification of the input:

The input may have one of the following 2 forms :

- a declaration of variables which may contain one variable and one value per line
- a mathematical expression using the basic mathematical operators stated in the hypothesis and either an integer or a variable that has been previously declared

3.3 The specification of the output:

The output , depending on the form of the input , may be :

- in case of a variable declaration the output is the value of the declared variable
- in case of a mathematical expression the output will be an integer which results after computing the given expression

3.4 The list of all the modules in the application and their description:

- list : contains functions that help to create and check different states of a list
- polish : is used to transform a given expression to its polish form

- tree : has functions that create a binary tree from a polish form of an expression and evaluates it
- inOut : in this module there are a few auxiliary functions used to read and print a variable or integer as well as the main functions which uses all the other modules to resolve the given task

3.5 The list of all the functions in the application, grouped by modules:

- list :
  - int isempty : a function that , given a stack , returns wheter it's emtpy or not
  - void emptystack : a function that , given a stack , sets its top to -1
  - void push : a function that adds a given item to a given stack
  - struct node* pop : a function that , given a stack , saves the data stored in the top node , deletes it from the stack and then returns the value
- polish :
  - int isoperator : a function that check if a given char represents a basic mathematical operator
  - int priority : a function that calculates the priority of a given operator and returns 1 if the operator is + , − or 2 if the operator is / , ∗ , ^
  - void infix2postfix : this function has 3 parameeters : an infix form of an expression (the default one), an array where the postfix form should be saved (the polish form) and a bool which tells wheter there's a space in the infix form . It goes throw the infix form and depending on the state of the read char (operator or digit) it creates the postfix which will be later used
- tree :
  - void maketree : a function that given a postfix form of an expression and the root creates a binary tree in which it stores integers and operators in a way that every 2 integers should have an operator as a common father
  - long long evaluatetree : a funtion that , given a node , checks if it is an operator firstly and then performs the operation defined by the operator between the sons of the node . If the given node is not an operator then the function returns the number stored in the node
- inOut :
  - void display : a print function

- void declare_var : a function that , given a line in which a declaration is written (contains a =) , it stores the value of the declared variable in an array
- void read : a function that , given an expression , check wheter it is a variable declaration or a mathematical expression . If it is a variable declaration then it stores the value as mentioned above and prints the value on the next line . If it is a mathematical expression then it performs the following steps:
  * changes its order from infix to postfix
  * creates a binary tree based off the postfix
  * prints an integer using the function "evaluatetree" which represents the result of the mathematical expression

4 Conclusions

5 References