\documentclass[12pt,a4paper]{article}

\usepackage[utf8]{inputenc}

\usepackage[english]{babel}

\usepackage[]{fullpage}

\usepackage{graphicx}

\title {Abacus \\ Interpreter for mathematical expressions in SML}

\author{

Vågbratt, Tommy

\and

Loberg, Micael

\and

Jin, Wenting}

\date{\today}

\begin{document}

\maketitle

\tableofcontents

\newpage

\section{Introduction}

\textnormal{A tool that takes mathematical expressions in text form may sound nothing special, but one may have encountered difficulties to find the right symbol on a typing-in calculator to express trigonometric related functions such as "sin", "arctan" etc. However, this program (Abacus) performs as an interpreter and functions just as a "normal" calculator, but can evaluate a whole mathematical expression that is just based on text! No more time wasting looking up symbols, with ability to declare variables, to do more advanced calculation. Interesting? Well, it's just a "calculator".}

\section{"Calculator" in SML}

\subsection{Design and Structure}

\textnormal{There are three major parts of the system: Input, Compiler and Evaluation which are combined into a REPL\footnote{Read-Evaluate-Print Loop, more about REPL, please refer to http://en.wikipedia.org/wiki/REPL}. All these parts are strictly sequential between each other. All steps inside a part are to build preparation work for next part, and all three parts form a successive execution. Executions can be achieved as many times as user need.}

\subsubsection{Structure Overview}

\begin{description}

\item [Input] \hfill \\ handles merely input text , checks if content other than whitespace is input and passes input onto Compiler

\item [Compiler] \hfill

\begin{itemize}

\item Tokenize: takes an expression represented as a string and split it into the tokens

\item Validate: performs grammatical validation on tokens, but careless of priority of the functions/operators

\item Translate: convert tokens from infix notation to postfix notation with priority considered

\end{itemize}

\item [Evaluation] \hfill \\ Compiled expression gets evaluated. A stack based "virtual Machine" evaluates expression as following rules:

\begin {itemize}

\item Numbers pushed onto the stack

\item Variables are looked up and get pushed onto the stack

\item Functions and operators takes item(s) off from the stack and push evaluated result back onto the stack

\end {itemize}

\end{description}

\iffalse

En mer detaljerad beskrivning (design)

- Vilka delar består systemet av? Hur samverkar de för att lösa problemet?

- Vilka datastrukturer används? Beskriv abstrakta datatyper (gränssnitt/interface)

\fi

\subsection{Algorithms and Alternatives}

\textnormal{ Data-types used in this program: Stack, Token, Environment.}

\begin{description}

\item [Stack] is a data-structure that only has two operations, pushing (adding) data to the top and the stack, and popping (removing and reading) data from the top of the stack.

\item[Token] is the data-structure used in the program to represent numbers, functions, variables, parentheses, operators and assignment (which is treated as a special operator).

\item[Enviroment] is a list of variables and their values.

\end{description}

\subsubsection{Tokenize}

\textnormal{Expressions consist of number, assignment, identifiers of variable, operator and function, open and close parenthesis and whitespace. Tokenize takes expression represented as a string and splits them into token characterized as Number, Variable, Assignment, Function, Operator, Open and Close parenthesis}

\begin{itemize}

\item Number:

\begin {enumerate}

\item \emph{Start}: take in digit 0-9.

\item if 0 encountered, either is a 0, \emph{or} followed by a decimal point with 0-9 combinations

\item if 1-9 encountered, either \emph{1)}:followed by 0-9 combinations \emph{or} \emph{2)}: 0-9 combinations followed by a dot(.) with 0-9 combinations \emph{or} \emph{3)}: followed by decimal point with 0-9 combinations

\end{enumerate}

\item Identifier:

\begin {enumerate}

\item \emph{Start}: take in alphabet a-z, A-Z until first digit 0-9 encountered

\item check if exist in function or operator list

\begin {itemize}

\item if exist recognized as an operator or a function, terminate and back to \emph{Start}

\item if not exist in list, take in the digit encountered and keep on reading a-z, A-Z, 0-9

\end{itemize}

\end{enumerate}

\item Symbolic Operator: take in symbolic operator (such as. \*,/,+,-, etc)

\item Parentheses

\begin{itemize}

\item take in left parenthesis (, terminate and back to \emph{Start}

\item take in right parenthesis ), terminate and back to \emph{Start}

\end{itemize}

\item Whitespace: whitespace “ “ encountered, terminate and back to \emph{Start}

\end{itemize}

\subsubsection{Validate}

\textnormal{Validate is achieved with recursive descent parser. \newline

Validate is implemented using a recursive descent parser algorithm.

It is made from a number of mutually recursive states, the states being:

\newline A has two constructs, either Var = A, or E}

\newline E is an expression and has two constructs, N op E, and N

\newline N also has two constructs, either its T, or –T

\newline T the

\begin {itemize}

\item A \Rightarrow Var = A | E

\item E \Rightarrow N op E | N

\item N \Rightarrow T | -T

\item T \Rightarrow Num | Var | fun N | (E)

\end{itemize}

\subsubsection{Translate}

\textnormal{The method for converting an expression in infix notation to postfix notation makes use of two lists holding, the first one holding the input and the second one holding the output, and a stack that holds the operators, functions and parentheses \newline \newline The algorithm works as following:}

\begin {enumerate}

\item read the first element of the input list, if it is empty, pop items off the stack to the output list. Once both input list and the stack are empty the algorithm is done.

\item if first item on the input list is a number or a variable, add it to the output list

\item input is a left parenthesis, add it to the operator stack

\item if input is a function or an operator, check the priority of the top element of the operator stack.

\begin{itemize}

\item if the item of the stack has higher priority, push it to the output list, and go back to the previous step.

\item if priority of the input is higher, put it on the output list

\end{itemize}

\item if the first element on the input list is a right parenthesis, pop elements on the stack to the output list until a left parenthesis is found, then discard both the left and right parenthesis.

\end{enumerate}

Finally, reverse the output list.

\subsubsection{Evaluate}

\textnormal{Evaluate is implemented with an algorithm that makes use Enviroment, a stack and a list containing the input

The algorithm works like this:}

\begin{enumerate}

\item if a number or a variable is the first element in the input list then push the value of the number/variable to the stack and go back to step 1.

\item if the input is a function or operator, pop off as many arguments as the function needs from the stack, evaluate the function and push the result back to the stack and go back to \emph{step 1}.

\item if input is the empty list, then the original expression is evaluated to the top of the stack

\end{enumerate}

After the expression has been evaluated, the result is put in to the environment as the variable “ans”.

}

\subsubsection{Alternative algorithms}

\textnormal{There are optional methods to get implementation of Tokenize, that for instance replace the state check by multiple case options, or check backwards of the string (which can be more complicated to implement), but such different ways can all lead to the same result, the way implemented in this program is actually not of much difference in the result but might be clearly structured when different state is called. \newline \newline There are different methods to implement Validate and Translate. Instead of first validating the expression and then translating it to postfix, an abstract syntax tree could have been built in the Validate. So there is no need to convert infix notation to postfix notation since it easily can be evaluated by traversing the tree. \newline \newline The actual method implemented in this step (shunting-yard algorithm) is the grounding method of this program, all built up functions and development is based on this idea once this method is introduced during the PKD course\footnote{find it.uu.se}. While shunting-yard algorithm handles priority problem, thus it become easier for the step of Validate, that Validate does not need to handle priority issues.}

\iffalse

Smarta lösningar (algoritmer)

- Beskriv viktiga funktioner Designval

- Finns alternativa sätt att lösa problemet? Varför valde du det här sättet?

- Konsekvenser av dina val

\fi

\subsection{Implementation}

\textnormal{Logotype is presented and there is a user manual built into the program, by typing “help”, user can get user guide on how to use the program. \newline \newline Test cases are included for every source code file. \newline \newline It is easy to add more support for functions and operators because priority is not part of the validation but is handled in the translation part.}

\iffalse

- Intressanta detaljer

- Fungerar programmet? Hur vet du det?

- Finns det saker som inte fungerar, fall som inte hanteras?

- Ãr programmet effektivt?

\fi

\subsection{General Analysis}

\textnormal{In order to keep the main loop running when exceptions are raised a new instance of the main function are run on top of it.}

\iffalse

Analys/diskussion

- Vilka Ãr systemets styrkor/svagheter? -> Sustainability

- Blev det bra? Skulle du ha gjort något annorlunda om du skulle börja om? -> Sustainability

- Tänkbara vidareutvecklingar -> Furture Development

\fi

\subsubsection{Sustainability}

\textnormal{}

\subsubsection{Furture Development}

\textnormal{}

\section{Brief Conclusion}

\textnormal{}

\iffalse

- Avslutning: Sammanfattning, diskussion, slutsatser

\fi

\section{Simple Guide for Simple Calculator}

\textnormal{}

\iffalse

- Användarhandledning med exempel (use cases)

\fi

\begin{thebibliography}{breitestes Label}

\bibitem{}

% info...

\emph{}

\bibitem{}

\end{thebibliography}

\end{document}