Middle East Technical University
Department of Computer Engineering

CENG 140

Spring 2019

Take Home Exam 2

Regulations

Due date: 30 April 2019, Tuesday *(Not subject to postpone)*

Submission: Electronically. You will be submitting your program source code written in a file which you will name as `the2.c` through the web system (follow instruction in Sec 3 group). Resubmission is allowed (till the last moment of the due date), The last will replace the previous.

Team: There is no teaming up. The take home exam has to be done/turned in individually.

Cheating: **<u>This is an exam:</u>** all parts involved (source(s) and receiver(s)) get zero+parts will be subject to disciplinary action.

I/O Specification: You are expected to write programs that %100 comply with the I/O specifications expressed in this worksheet. Do not beautify you I/O.

Introduction

In this homework we will be dealing with the differentiation of single variable mathematical expressions.

To ease the pain of parsing a restricted and simplified version of mathematical expression will be considered. Below you will find the BNF description of the syntax of the mathematical expressions we will be dealing with.

$$\begin{aligned}
\langle\text{expression}\rangle &::= \langle\text{expression}\rangle\,\langle\text{operator}\rangle\,\langle\text{expression}\rangle \mid \\
&\ \langle\text{expression}\rangle\ \texttt{\^{}}\ \langle\text{natural number}\rangle \mid \\
&\ \texttt{(}\ \langle\text{expression}\rangle\ \texttt{)} \mid \\
&\ \langle\text{atomic}\rangle \\
\langle\text{atomic}\rangle &::= \langle\text{natural number}\rangle \mid \texttt{X} \mid \texttt{sin} \mid \texttt{cos} \mid \texttt{tan} \mid \texttt{sh} \mid \texttt{ch} \mid \texttt{ln} \\
\langle\text{operator}\rangle &::= \texttt{+} \mid \texttt{-} \mid \star \mid \texttt{/} \\
\langle\text{natural number}\rangle &::= \texttt{0} \mid \texttt{1} \mid \texttt{...} \mid \texttt{100}
\end{aligned}$$

In addition to this an expression can contain any number of whitespace provided that does not split an $\langle atomic\rangle$. The semantics is as follows

- The single variable X is denoted by the letter `X`.
- `sin`, `cos`, `sh`, `ch`, `tan`, `ln` stand for $\sin(X)$, $\cos(X)$, $\sinh(X)$, $\cosh(X)$, $\tan(X)$ and $\ln(X)$ respectively.

- $\langle expression \rangle$^$n$ means $\langle expression \rangle^n$
- Precedence (greater value means performed first) and associativity of the operators are

Operator	Precedence	Associativity
`+ -`	1	left
`* /`	2	left
`^`	3	right

Problem

Your program will read a single expression and will produce the derivative expression (with respect to X) of this expression.

The resulting expression doesn't have to be in the *simplest* form. Furthermore it is allowed to have parenthesis which actually could be omitted. Though, having no more then 20% more then necessary parenthesis will be awarded with an additional 20 points. So, you have the chance to receive 120 points from this homework.

Here is an example,

Input:

```
(X^2-1)*tan - ln^2   /X
```

Output:

```
(X^2-1)*(tan^2+1)+2*X*tan+(2*ln*X-ln^2)/X^2
```

This is quite an elaborated output. Yours does not have to be that advanced. (But if you manage to do so you get bonus!)

```
(((((X^2)-1)*((tan^2)+1))+((2*X)*tan))+((((2*ln)*X)-(ln^2))/(X^2)))
```

is another alternative with some redundant parenthesis present.

Specifications

- You will be reading a single expression from the standard input which is the expression you will take the derivative of. The input may be scattered over lines, contain blanks (not corrupting any atomic) and will be terminated by EOF.
- Your output is the derivative expression on a single line containing no blanks.
- The count of operations in the input is ≤ 20.
- The following derivative table is provided for your convenience:

Expression	*Expression′*
N (*natural number*)	`0`
`X`	`1`
`sin`	`cos`
`sh`	`ch`
`ch`	`sh`
`ln`	`1/X`
$\square_1$ `+` $\square_2$	$\square'_1$ `+` $\square'_2$
$\square_1 \star \square_2$	$\square'_1 \star \square_2$ `+` $\square_1 \star \square'_2$
$\square_1$ `/` $\square_2$	`(`$\square'_1 \star \square_2$ `-` $\square_1 \star \square'_2$`)/`$\square_2^2$
$\square$ `^`N	$N\star\square$`^(`N`-1)`$\star\square'$

ARKADAŞLAR TARİHİMİZİN EN ÖNEMLİ MAÇINA ÇIKIYORUZ... İŞTE TAKTİĞİMİZ: GOL ATILACAK! TAMAM? NAAPIYORUZ? GOL ATIYORUZ. OK?
ULAN BÖYLE TAKTİK Mİ OLUR, NE BİÇİM HOCASIN SEN BE! NASIL ATICAZ GOLÜ ONU SÖYLESENE!
AYAANAN!
HAY ALLAH SENİN BELANI VERSİN!..
BURUYA GOL AT!

- Any exponent given in the input will be ≥ 2.
- Simplification is not required in this homework. This is so even for operations only involving numbers with one exception: the exponent. Any exponent in your output must be a reduced to a single number.
- No `cos` will be appear in the input. *(The reason for this is to keep the problem simple: we do not have unary minus!)*
- If you do some simplification, do not produce negative numbers, or introduce unary minus signs, this may mislead the evaluation process.
- The correctness of your output expressions will be tested by evaluating the expression at several numerical values of `X`. Therefore, the order and form of your expression is unimportant, since arithmetic obeys the Church-Russer property.
- You are guaranteed that the input is well formed and does not contain any error, complies with these specifications. Therefore, you do not have to bother with any error check.
- Please note that the variable `X` is in uppercase. Do not produce lowercase `x`'s accidentally.

How to Do's

- Implement Dijkstra's shunting yard algorithm (Phase 1).
- Implement evaluation algorithm (Phase 2), but push on this stack not the values but the reconstructed infix expressions. The best way to do this is pushing on the stack pointers pointing to dynamically created strings that are the infix representations of the sub expressions. When you pop two such subexpression strings from the stack with the intention to join them under an operation and push it back on the stack, do not forget to free the space allocated for the operants, after having done the 'joining'.
- **Golden hint:** It is very vise to run a parallel (shadow) stack during Phase 2, where the derivatives (also in infix form) are kept (generated) in parallel to the infix reconstruction process.