pl98a: Project
Symbolic Calculator Revisited

February 17, 1998

The goal of this exercise is to write a program named plmath, which is a helper to the mathematician. The program reads its input from the standard input. The output is the standard output. Each line in the input is either empty, starts with a #-sign, and in this case plmath has to do nothing. Otherwise, it is a mathematical expression or an assignment. The program has to support treating functions. Each function is a function of x (only). Beside functions, plmath also supports handling constants. The name of a function or a constant is a letter followed by at most 8 alphabetic and underscores characters. A variable name is never ‘x’. The commands that plmath have to supports are:

\[
\text{func\_name}(x) := \text{expression}
\]

\[
\text{e.g.} \quad f(x) := x \cdot 12
\]

\[
\text{e.g.} \quad g(x) := f(x + 10) + f(0)
\]

\[
\text{e.g.} \quad g@1b(x) := 1 + x
\]

Define the function \( f(x) \) to be \( x \cdot 12 \), and \( g(x) \) to be \( (x + 10) \cdot 12 \). It also defines a function \( g@1b(x) \) to be \( 1 + x \). Notice, if \( f \) refers to other functions or constants, your program should substitute them into the function immediately. A function that does not depend on \( x \) is considered to be a constant, and might be referred to directly -

\[
\text{e.g.} \quad \pi := 3.1415
\]

\[
\text{e.g.} \quad \text{two\_pi} := \pi + \pi(x)
\]

Notice that accessing a function must be done using parenthesis. Thus, the following is invalid:

\[
\text{e.g.} \quad f := x \text{ or } f + 1.
\]

Notice that an identifier (a constant or a function name) might be defined only once. In particular, it is impossible to redefine a constant or a function.
**expression**

Read the given expression, and compute its value. If it is a constant expression, the program prints its value, else it prints the expression.

- **e.g.** $\pi$
  - **Output:** $3.1415$
- **e.g.** $f(x) + g(x)$
  - **Output:** $x^{12} + (x + 10)^{12}$
- **e.g.** $f(1)$
  - **Output:** $1$

You may assume that the input is given in the same nice and friendly format of exercise 2. However, `plmath` is required also to support the following additional operators/functions:

- **exp, ln** Your program should support the exp, ln functions, namely $\ln(x)$ is the natural logarithm of $x$, and $\exp(x) = e^x$. (Notice, that you can assume that the power of an expression is constant.

**derive(expr)** Compute the given expression derivative.

- **e.g.** $\text{derive}(g(x))$
  - **Output:** $12 \times (x + 10)^{11}$
- **e.g.** $h(x) := \text{derive}(\text{derive}(g(x)))$
  - Defines the function $h(x) = 132 \times (x + 10)^{10}$.
- **e.g.** $\text{derive}(x \cdot x)$
  - **Output:** $(x \cdot x) \times (\ln(x) + 1)$

**taylor(expr1, num)** Compute the polynomial which is the sum of the first num monomials in the Taylor expansion of expr1 around 0.

- **e.g.** $\text{taylor}(1/(1 - x), 3)$
  - **Output:** $1 + x + x^2$

**integral(expr, value1, value2, value3)** Compute the number which is the (numerically computed) integral of expr between value1 and value2.

Your project need to do the following

1. Prepare a c-program named `plint.c` which computes the value of the function `func_name` in the desired limits, and prints the result to the standard output.
2. Compile this program (using \
g++\) into an executable file named plint. Check the exit status to see if the compilation succeed, and if not, stop and notify.

3. execute this program using the command popen (see manual) read the result and return it as the value of the expression.

For computing the integral, divide the interval \([value_1, value_2]\) into \(value_3\) equally spaced subintervals, and approximate the region below the graph of \(expr\) in this subinterval as a trapezoid. See Figure 1.

e.g. \(r(x) := integral(x, 0, 10, 1000) + derive(x^2)\)
Define the function \(r(x) = 50 + 2 * x\).

All accesses to functions and constants should be done using an open address hash table containing their names and their associated values. When defining the table, your program should start with a table for 10 names. Whenever, the number of names in the table is more than twice its size, you should allocate a space for a twice-as-large table (i.e. the first time you resize the table it should have 20 entries), make a rehash into this table, and free the old one. See for example the book by T. H. Cormen, C. E. Leiserson and R. L. Rivest, “Introduction to Algorithms”, (The MIT Press, Cambridge, Mass. 1990).

**Function simplification**

Before printing an expression plmath should simplify the expression, according to the following rules (which should be applied recursively, until the expression can not be simplified any more):
0 * expr \iff 0 
expr * 0 \iff 0
1 * expr \iff expr
expr * 1 \iff expr
0 + expr \iff expr
expr + 0 \iff expr
0 - expr \iff -expr
expr - 0 \iff expr
exp(ln(expr)) \iff expr
ln(exp(expr)) \iff expr
expr/1 \iff expr
0/expr \iff 0
expr^1 \iff expr
expr^0 \iff 1

Moreover, your program should recursively identify subexpressions (i.e. subtree of the function) which are constant, and replace them by their exact value. For example -
expr*(1 + 2 + 3) \iff expr * 6.

Guidelines

- Submission dates

**Files and Printouts:** 1.4.97 at 23:00 on libra.

Any change you do make in the files later than that date will cost you points in your grade.

- All calculations in your project should be done using the type double.

- You might assume that no syntactic errors appear in the input. However, there might be tabs or spaces inside it.

- You are requested to open a directory named "/p197a/proj" (subdirectory "p197a/proj" under your home directory). Make sure that the files containing your project reside EXACTLY where you were asked to put them and with the EXACT names, and that the project you were requested to write have the EXACT interface as requested. If you do not do all that, then your grade will be severely affected. Remember that the installation of your work is a part of it.

Set the permission on your source files and the directories in the path leading to them, to be 705. That is,
chmod 705 "/"/p197a "/p197a/proj "/p197a/proj/*

- The project MUST be submitted both on the computer (compilable by g++ on libra, containing its documentation within the files) and on paper (printout of the files).
login name should be printed/written on the printouts. You must do this assignment on your own, that is, singleton submissions only.

- In addition to your source files, you are required to supply us with a makefile that compiles your project (with g++ -Wall). The make file should be ~/p197a/proj/Makefile. Running make should create the executable of your project named plmath.

- Your project must be compiled under g++ with full warning mode. Moreover, your source MUST compile with NO warnings what so ever.

- Add your name, id, and login in a remark in the top of each source file.

- Write clear and readable code. Use assertions to ensure your project correctness. In particular, your program should check (or assert) for errors returned by all the calls to library functions in your program, and respond accordingly. Moreover, give meaningful names to variables, constants, and functions in your project.

- Comment your project. But use your common sense! you will lose points for over commenting.

- You might add friendly messages to your program, any such message should start with a #-sign in the beginning of the line. In particular, plmath| plmath should work correctly for any valid input.

  We had implemented the project. You might play and test it. Your implementation should have the same input/output format.

  The exercise will not be checked if the above is not fulfilled. Furthermore, to get a full grade, your program should output EQUIVALENT results to these of the demo program.

- Any changes or updates concerning the exercise will appear in the FAQ.

  Good Luck!