1 Purpose

The purpose of this project is to expose any weaknesses you have developing C++ applications in Linux. This project provides an early opportunity to address shortcomings before before we approach the complex tasks involving the language and operating system.

2 Task

Create a command line calculator named `calculate` capable of correctly evaluating infix notation expressions involving addition, subtraction, multiplication, and division. Correct evaluation means that order of operation is observed.

Input should be taken as arguments to the application, e.g.,
`./calculate 2 + 3 x 7`
Provide a makefile which generates an executable named
`calculate`
when given the command
`make`
And deletes any generated intermediary files when given the command
`make clean`

3 Deliverables

3.1 Submission

You must submit a zip archive and only a zip archive. Individual files and other archive types will not be accepted; 100% penalty.

Your archive must hold a topmost/root directory named `proj1`[1] containing only documentation and files necessary for compilation of the application, i.e. any source, header, or dependency files necessary for compilation; 10% penalty.

Your project must adhere to basic C++/C best practices. You must have the following:

- `README.md`
  A non-RTF text file, though you are free to use the standard markdown language of this file type if you choose. The file must describe the purpose of each included file (other than itself).
- `main.cc/.c`
  A C++/C source file containing only your application's entry point. The only code allowable in this file is your `main` function and any helper functions used to process input.
- Any number of source and header files needed to provide correct program behavior. Note that, though you may have more, you must have one `.cc`/`.c` source file and one `.h` header file, e.g., `calculate.cc/.h` or `str_array_to_equation.cc/.h`.
- A makefile to build the project. The makefile must build the project completely when given the
  `make`
  command, creating the executable `calculate`. It must clean any intermediary files when given the
  `make clean`
  command, leaving only the described files and the executable in the submission directory.

[1] This name is case-sensitive.

All source code must be styled. Use the Google style guidelines for your code; 15% penalty. Assuming you have access to Python 3 and Pip 3, install cpplint for ease of checking your code's style. Code so poorly styled as to be unreadable will not be considered.

3.2 Behavior

Your application must accept any valid infix arithmetic expression of any length as a command line argument. It must correctly implement order of operation for +, -, /, and x. Notice multiply is an 'x' not an asterisk and both '(' and ')' are omitted.

Numerics may be of any of the following formats: 1, 1.0, +1.0, -1.0, etc.

Output should be on the standard output stream (STDOUT) and should be followed by one new line character ('\' or `std::endl`).

An example interaction would be as follows:

```
user@machine:~/311/proj1/$./calculate 3 + 2 x 7
17
```

or

```
user@machine:~/311/proj1/$./calculate 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 x 2
1024
```

4 Points

This section describes the points awarded for each correct project deliverable. There may be additional penalties assigned; see the Submission subsection for details.

- Documentation: 1 point
- Correct(compilable) entry file (main.cc/.c): 1 point[2]
- Correct(compilable) application specific libraries: 1 point[3]
- Correctly behaving makefile: 1 point
- Correct evaluation of described abridged infix arithmetic: 1 point

Some portions must be correct before others can be tested, e.g., unless your whole program compiles, your makefile cannot be tested.

[2]Code that compiles, but contains warnings will only receive 0.85/1.0 points.
[3]Code that compiles, but contains warnings will only receive 0.85/1.0 points.

The End.