# CMPSC461 Project 1

**Deadline**: March 17, 2019
NO extensions will be given

## Introduction

In this project, you will work as a group (max 2 students). You will be using **Flex** and **Bison**. The installation instructions are provided to be before hand in canvas. These tools are probably new to you, so it is **extremely important to get a familiarity with these tools beforehand.** We will provide you a couple of tutorials on these tools and we will do our best to help you with your projects. **But, we cannot help you if you don't at least read about these tool documentation and try them before coming to our office hours.**

### Flex

Flex is a lexical analyzer. It is basically a tool for generating tokens from the input given by matching the **regular expression** patterns. For example, simple flex rule for matching integers is:

$$[0-9]+ \ \{return \ \ T\_INT; \}$$

Flex uses regular expressions as patterns. You can think of flex as a terminal generator to use in your Context Free grammar. For example, assuming we have another rule in flex that returns **T_PLUS**. We could create a context free grammar production for addition on integers using these two tokens such as;

$$S \rightarrow \textbf{T\_INT} \ \ \textbf{T\_PLUS} \ \ \textbf{T\_INT}$$

### Bison

Now we have tokens generated by Flex, Bison is a parser generator, basically a tool to create CFG with attributes. For example, the production example above is written in Bison. You know what is a CFG grammar and what is an attribute grammar from the lectures. We will be using Bison to create an attribute grammar in this project.

Please checkout these tutorials and examples to understand how Flex and Bison works;

- https://www.gnu.org/software/bison/manual/bison.html
- http://aquamentus.com/flex_bison.html
- https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator/

### Finally

Since flex and bison will probably be new to almost all of you, their syntax and rules might seem hard to understand at first. Hence we recommend you to try a basic example first.

## Project Description

- You will create a parser that parses simple arithmetic operations on roman numerals.

  For example, you need to parse expressions like these;

  - **XXV** / **IV** $\rightarrow$ means 6.25
  - **CCCXCV** - **LV** $\rightarrow$ means 340
  - **CCCXCV** * **CXII** $\rightarrow$ means 44240
  - **IV** + **III** $\rightarrow$ means 7

- In this project, you will also parse "**floating roman numbers**" with roman numerals;

  **X.XX** $\rightarrow$ means 10.20

  **CC.CCX**$\rightarrow$ means 200.21

  **XXIV. III**$\rightarrow$ means 24.3

  It is actually straightforward, your input is a float if and only if it contains a dot **'.'**.

- You will define an attribute grammar in which you need to find a way to calculate the value $0.210$ from the input **.CCX** and add this value to the whole part of your number **CC**. So your attribute grammar for float expressions would look like

$$\text{Prodecure} = \text{WholeValue} + \text{Calculate(Fractional)}$$

### Notes

You can check the attribute grammar question in HW4 to see an example.

Floating roman numbers can also be operands of an expression, such as;

**XXV.V** + **IV** $\rightarrow$ means 29.5

These are all the expression types you need to parse;

- Roman numerals between [1 - 399] (That means you need to only parse strings containing roman numerals **I, V, X, L, C**)
- Floating roman numerals between (0 - 400) (You should be able to parse the expression **.XV** $\rightarrow 0.15$)
- Arithmetic operations / * - + on roman numerals.
- Roman numeral Chart for reference http://literacy.kent.edu/Minigrants/Cinci/romanchart.html

**We will test your implementation based on three components of this project** Your parser should be able to;

- Parse simple roman numerals.
- Parse floating point roman numerals.
- Parse arithmetic operations / * - + on roman numerals.
- Print out the result of the statement after parsing it. For example, after parsing the line **X.XX**, You should print this statements value, which is 10.20.

## Project Structure and Makefile

We have included a tarball which contains the starter files. You will not use any additional files for your project. You cannot change the basic structure of the files provided as well. We included a makefile in your project files. You can use this makefile to compile, run, test, and compress before submitting your implementation.

**Testing:** You have a test file you can test your implementation with, you can test your executable by `make test`. Keep in mind that, we won't be using these files in our tests and you should always test your implementation with additional tests. The out.txt file is for you to check your results against.

**Submitting:**

- The makefile in your project folder contains a target to create tarball of your **p1.flex** and **p1.y** files.
- The following command "***make tar***" It creates p1.tar.gz compressed file for your submissions. You need to submit this tarball that contains both flex and bison files.

**One additional note:** If you are parsing your expressions as `expression new_line`, using the test file might result in an error at the last line if the last line is not a blank new line. So the test file should always have an empty line at the end of it. We will also be testing your implementations with a test file as mentioned.

## Academic Integrity Policy

- Please be aware of the EECS Department's Academic Integrity Policy at http://www.eecs.psu.edu/students/resources/EECS-CSE-Academic-Integrity.aspx.

- Students are required to follow the university guidelines on academic conduct at all times. Students failing to meet these standards will automatically receive a 'F' grade for the course. The instructor carefully monitors for instances of offenses such as plagiarism and illegal collaboration, so it is very important that students use their best possible judgement in meeting this policy. The instructor will not entertain any discussion on the discovery of an offense, and will assign the 'F' grade and refer the student to the appropriate University bodies for possible further action.
- Note that students are explicitly forbidden to copy anything off the Internet (e.g., source code, text) for the purposes of completing an assignment or the final project. Also, students are forbidden from discussing or collaborating on any assignment except were explicitly allowed in writing by the instructor.