CS4121 PJ Compiler Project 3: Implementing Arrays, Conditionals and Iteration
Due Date: Monday, March 17, 2025 at *11:59pm*

Purpose

The purpose of this project is to gain experience in giving meaning to a programming language by generating x86-64 for a subset of PJ. Specifically, you will be generating assembly for if-statements, while-loops, and one-dimensional or two-dimensional integer or boolean arrays.

Project Summary

In this project, you will add/update grammar rules and actions to the compiler developed in the previous project that will do the following

1. Add/update the grammar rules to handle any integer array bounds
2. Generate assembly to test the results of an expression controlling an if-statement or while-loop.
3. Generate assembly to branch through an if-then-else statement as determined by the controlling expression
4. Generate assembly to branch back to the beginning of and out of a while-loop as determined by the controlling expression.
5. Reserve space for an array of integers or booleans.
6. Generate address arithmetic for an array reference.

Follow the methods given in class for generating code for if-statements, while-loops and arrays. Note that an array is laid in memory from low address to high address, and variables, including both scalar and array variables, must be contiguous. The current grammar rules as they are can only handle non-negative array bounds. There will be a parse error when compiling test cases 15 and 16. You need to update the rules so the negative bounds are acceptable.

Requirements

Write all of your code in C or C++ . It will be tested on CS lab machines or MTU provided remote Linux systems and MUST work there. You will receive no special consideration for programs which "work" elsewhere.

Input. I have provided my solution to Project 2 as a Makefile project in `PJProject3.tgz`. Sample input is provided in the directory `PJProject3/input`. I will go over my code for those who wish to use it. To run your compiler, use the command

```
pjc <file>.pas
```

which will output to `<file>.s`.
To create an executable, run the following command on the assembly file

```
gcc -o <file> <file>.s
```

You may then directly run the executable.

Submission. Your code should be well-documented. You will submit all of your files, by tarring up the working directory using the command

```
tar -czf PJProject3.tgz PJProject3
```

Submit the file `PJProject3.tgz` via Canvas. Make sure you do 'make clean' of your working directory before executing the tar command. This will remove all of the '.o' files and make your tar file much smaller.

Additional Notes

The output for test case 25 is different from the one using `fpc` generated executable. The expected output for test case 25 is as follows:

```
b = -1
b = 4
c[4] = 4
c[5] = 5
c[6] = 6
c[7] = 7
c[8] = 8
c[9] = 9
c[10] = 10
c[11] = 11
c[12] = 12
c[13] = 13
c[14] = 14
c[15] = 15
c[16] = 100
```