CPEG 421/621 Project: A Calculator Compiler

The project is due 11:59PM May 20th, with a final presentation on May 17th. This is an individual project.

Objective: The objective of this project is to implement a calculator compiler including several optimizations.

Software tools: You need three open-source software for this project: bison, flex and gcc. They are readily available on almost all computer platforms.

- For this assignment, you can use any Linux machine that has lex and yacc (or flex and bison) installed.
- You may use a Windows 10 machine provided that you install "windows subsystem for linux", which is basically an Ubuntu Linux system.
- You may use a Mac OS X machine provided that you install Xcode.

Specification of the calculator language:

The language: The calculator compiler accepts a language that is very similar to the grammar you handled in the previous lab. Basically, a program written in the calculator language is a list of statement. The only data type allowed is integer. Each statement is an expression that consists of variables, integer constants and operators "+", "-", "*", "/", "!", "**" and "=", as well as parenthesis. Variables don't need to be declared before use, and variable names are case-insensitive. The value of an undeclared variable in its first use is "0". The operator "=" returns the value that is being assigned.

The language also supports conditional expression "?". Its format is "(cond_expression)?(expression)". The conditional expression is similar to the "?" operator in C. The semantic is that the conditional expression will return the value of "expression" if "cond_expression" is non-zero. Otherwise, the whole expression returns "0". An example is "(x=4)?(x=5)".

An example program might look like the following:

```
A = B+C*1
C = (B=D/2)
E=(D=4)?(D=5)
F=D+E
```

Task 1: Compiler frontend and backend

Your compiler frontend should translate any legal program written in the calculator language into an equivalent three-address code representation. The compiler backend should translate any valid three-address internal representation into a C program, treat any variables ***used without definition*** as user inputs, and print out the last value of all the variables appearing in the original program. Note that the backend doesn't print the temporary variables introduced during compilation. Also assign a label for every statement. For the above example program, a possible output will be:

```
main(){

        int A, B, C, D;
        int Tmp1;

        printf("B=");
        scanf("%d", &B);

        printf("C=");
        scanf("%d", &C);

        printf("D=");
        scanf("%d", &D);

        S1: Tmp1 = C*1;
        S2: A = B+Tmp1;
        S3: B = D/2;
        S4: C = B;

        println("A=%d\n", A);
        println("B=%d\n", B);
        println("C=%d\n", C);
        println("D=%d\n", D);

}
```

Task 2: Data dependence analysis

Analyze the data dependence based on the three-address internal representation. Print out the three types of data dependence for every statement. Following is an example output for the above program.

…
S2:
Flow dependence: S1
Anti-dependence: S3
Write-dependence: None
…

Task 3: Optimization

Implement the common subexpression elimination and the copy-statement elimination. Also write a process to iteratively apply either optimization, and be able to tell whether a fix point has been reached, i.e., not more optimization can be done. The example output after the copy-statement elimination is:

```
S1: Tmp1 = C*1;
S2: A = B+Tmp1;
S3: B = D/2;
S4: C = D/2;
```

Undergraduate students are only required to be able to apply the two transformations separately as following:

```
While(not fixed point){
        Do common subexpression elimination
}

or

While(not fixed point){
        Do copy-statement elimination
}
```

Graduate students are asked to develop reasonable heuristics for the two transformations, so that when they are applied together iteratively, a fixed point can still be achieved. Also report the performance of test benches before/after the transformation.

```
While(not fixed point){
        If(heuristic for CSE is true){
                Do common subexpression elimination
        }

        if(heuristic for copy elimination is true){
                Do copy-statement elimination
        }
}
```

What to Turn In

All project source files and report are due by 11:59PM May 20th.