Lab 2: A Calculator Compiler Back End---Instruction Scheduling and Register Allocation

Lab 2 is due Friday April 8. Your lab report and source code must be submitted by **12:20PM before the class**. The late policy applies to this lab project.

This lab is an individual project. Get started early! The required format for lab reports can be found on the resource page.

Objective:

The objective of this project is to implement a calculator compiler back end. You are going to practice the basic instruction scheduling and register allocation algorithm.

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: Revised compiler frontend

Revise your compiler frontend from the Lab 1 to (1) accept program from a file, not from command line; and (2) translate any legal program written in the calculator language into an equivalent three-address code representation. Note that you may introduce temporary variables when translating into three-address code. For the above example program, one of the correct outputs can be:

```
Tmp1 = C*1;
A = B+Tmp1;
B = D/2;
C = B;
D=4;
If(D){
        D=5;
        E=5;
} else {
        E=0;
}
F=D+E;
```

Task 2: Instruction scheduling or register allocation

The transformation should work on the three-address code the generated from the front end and output in three-address code.

Choose one of the instruction scheduling or register allocation problems.

(1) Implement the basic instruction scheduling algorithm based on the three-address code and manually (or using compiler to) measure the performance difference before/after the scheduling. Assume the operations having the following latencies:

Operations	Latency (cycles)
+, -	1
=, ?	2
*, /	4
**	8

(2) Implement the basic register allocation algorithm based on the three-address code. Assume we have only 4 registers (r1, r2, r3, r4). If a variable is being loaded into register, a loading assignment should be inserted ***before*** the variable's use. For example, if the expression is "a=b+c", and your algorithm choose to load "b" into register r2, the code should be transformed to:

```
r2 = b
a=r2+c
```

If a variable is being spilled from register, there are two possibilities:

a) during the period the variable is stored in the register, if the register has been modified, you should insert a storing assignment ***after*** the spilling of the register. For example, if your algorithm chooses to spill r2 after "d=r2+1",

```
r2 = b
a=r2+c
.
.
.
r2=x+y
d=r2+1
```

Should be transformed into:

```
r2 = b
a=r2+c
.
.
.
r2=x+y
d=r2+1
b=r2
```

b) during the period the variable is stored in the register, if the register has not been modified, you should do nothing.

Task 3: Backend Code Emission

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;
        S5:     D=4;
        S6:     If(D){
        S7:             D=5;
        S8:             E=5;
        S9:     } else {
        S10:            E=0;
                }
        S11:    F=D+E;


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

}
```

What to Turn In

All project source files, **your test files** and report should be submitted to CANVAS by the deadline.