Compilers & Lab. Final Exam Mockup

(2022, Term 2)

NB)

1. **Turn off** mobile phones & PC (핸드폰 및 PC **전원 OFF**)

2. You will get 0 score and be asked to leave on cheating (부정행위시 0점처리 즉시퇴실)

3. Solve **in order**. (**순서대로** 답안 작성. 어길시 **감점 10%**)

1. Fill in the blanks

<Semantic Analysis>

1.1. Ensure that the program has a well-defined (meaning).

<Challenges in Semantic Analysis>

1.2. (Reject) the largest number of incorrect programs.

1.3. (Accept) the largest number of correct programs.

1.4. Do so (quickly).

<Other Goals of Semantic Analysis>

1.5. Gather useful (information) about program for later phases:

1.6. Determine what variables are meant by each (identifier).

1.7. Build an internal (representation) of inheritance hierarchies.

1.8. Count how many variables are in (scope) at each point.

2. Complete inference rules below

2.1. Assignments

$$\frac{S \vdash e_1 : T \quad S \vdash e_2 : T}{S \vdash e_1 = e_2 : T}$$

2.2. Arrays

$$\frac{S \vdash e_1 : T[] \quad (S \vdash e_2 : int)}{S \vdash e_1[e_2] : T}$$

3. Fill in the blanks.

<Runtime Environments>

3.1. A runtime environment is a set of data structures maintained at (runtime) to implement these high-level structures.

e.g. the stack, the heap, static area, virtual function tables, etc.

3.2. B. Strongly depends on the features of both the source and (target) language. (e.g compiler vs. cross-compiler)

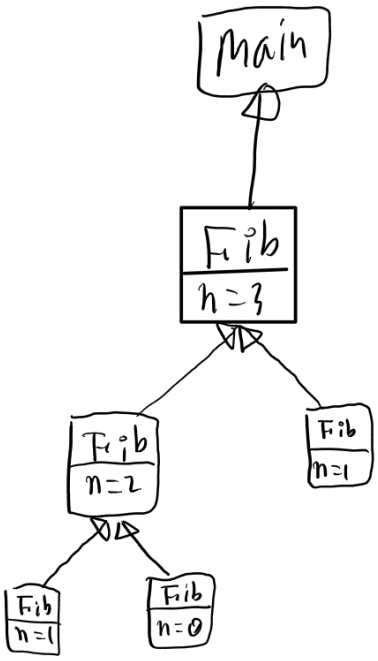
4. Draw an activation tree for the following codes.

```
int main() {
    Fib(3);
}

int Fib(int n) {
    if (n <= 1) return n;
    return Fib(n - 1) + Fib(n - 2);
}
```

```
int main() {
    Fib(3);
}

int Fib(int n) {
    if (n <= 1) return n;
    return Fib(n - 1) + Fib(n - 2);
}
```



5. Fill in the blanks.

< Stack Management in TAC>

5.1. The BeginFunc N; instruction only needs to reserve room for (local) variables and temporaries.

The EndFunc; instruction reclaims the room allocated with BeginFunc N;

5.2. A single parameter is pushed onto the (Stack) by the caller using the PushParam var instruction.

Space for parameters is reclaimed by the caller using the PopParams N; instruction.

- N is measured in bytes, not number of arguments.

6. Convert the following codes to TAC.

```
int x;
int y;
int z;

if (x < y)
    z = x;
else
    z = y;

z = z * z;
```

```
_t0 = x < y;
If z _t0 Goto _L0
z = x;
Goto _L1;
_L0:
    z = y;
_L1:
    z = z * z;
```

7. Optimize the codes given below. (아래 주어진 코드를 최적화 하시오.)

```
int x;
int y;
bool b1;
bool b2;
bool b3;

b1 = x + x < y
b2 = x + x == y
b3 = x + x > y
```

```
_t0 = x + x;
_t1 = y;
b1 = _t0 < _t1;

_t2 = x + x;
_t3 = y;
b2 = _t2 == _t3;

_t4 = x + x;
_t5 = y;
b3 = _t5 < _t4;
```

delete

delete

t1, t2

⇓

```
_t0 = x + x;
_t1 = y;
b1 = _t0 < _t1;

b2 = _t0 == _t1;

b3 = _t0 < _t1;
```

8. Fill in the blanks.

< Control-Flow Graphs >

1.1. A. control-flow graph (CFG) is a graph of the (basic) blocks in a function.

- The term CFG is overloaded – from here on out, we'll mean "control-flow graph" and not "context free grammar."

1.2. Each edge from one basic block to another indicates that (control) can flow from the end of the first block to the start of the second block.

1.3. There is a dedicated node for the start and (end) of a function.

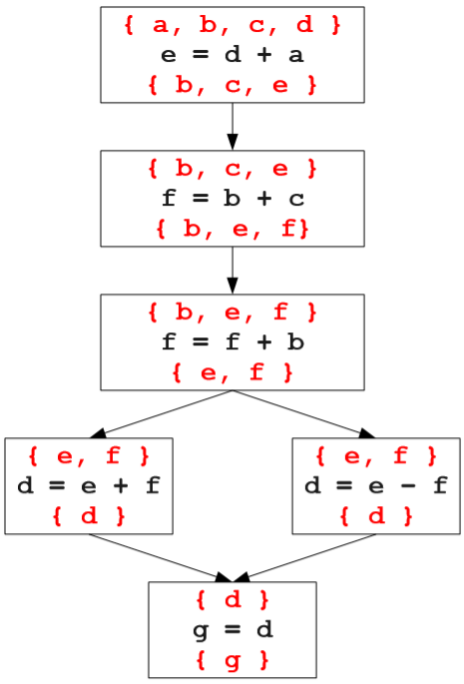
<Types of Optimizations >

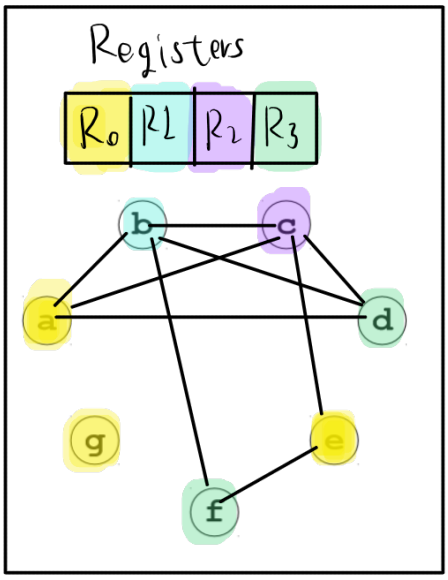
1.4. An optimization is (local) if it works on just a single basic block.

1.5. An optimization is (global) if it works on an entire control-flow graph.

1.6. An optimization is (interprocedural) if it works across the control-flow graphs of multiple functions.

9. Show the results of the connections between each variable for an RIG (register interference graph) of the codes below. (아래 주어진 코드에 대해 RIG를 위한 각 변수들간의 연결 관계를 보이시오.)





Register Allocation with Live Intervals