Sample Problems for CS 152 Final

Q1. Below is the specification of a control flow construct and its grammar:

Example:

```
If x < y then
        <otherstatements>
elseif a > b then
        <otherstatements>
else
        <otherstatements>
endif
```

Relevant Grammar Productions:

<S> → if <condt> then <otherstatements> <rest>

<rest> → elseif <condt> then <otherstatements> <rest>
| else <otherstatements> endif

<condt> → id relop id

Provide the semantic rules that generate code for the above control flow construct. The generated code should be available in the <S>.code attribute. Assume that semantic rules specified by other productions not shown here will place the code generated for <otherstatements> in the attribute <otherstatements>.code.

Solution:

<condt> → id_1 relop id_2 {
truelabel = newlabel();
<condt>.falselabel = newlabel();
<condt>.code = gen("if" id_1.place "relop" id_2.place "go to" truelabel)
|| gen("go to" <condt>.falselabel) || gen(truelabel":")
}

<S> → if <condt> then <otherstatements>
{
<rest>.ifalselabel = <condt>.falselabel;
<rest>.iexit = newlabel();
<rest>.icode = <condt>.code || <otherstatements>.code ||
gen("go to" <rest>.iexit)
}
<rest> { <S>.code = <rest>.scode }

<rest$_1$> → elseif <condt> then <otherstatements>

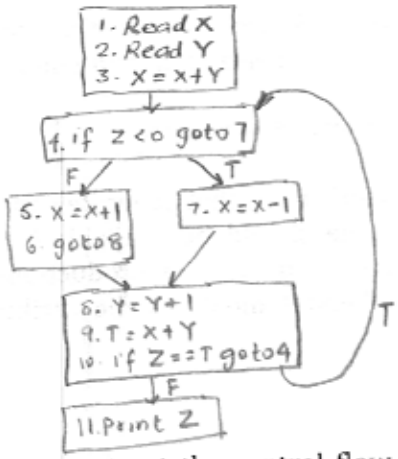
```
{
  <rest2>.icode = <rest1>.icode || gen(<rest1>.ifalselabel ":") ||
    <condt>.code || <otherstatements>.code || gen("go to" <rest1>.iexit);
  <rest2>.ifalselabel = <condt>.falselabel;
  <rest2>.iexit = <rest1>.iexit
}
<rest2> { <rest1>.scode = <rest2>.scode }
```

<rest$_1$> → else <otherstatements> endif

```
{
   <rest1>.scode = <rest1>.icode || gen(<rest1>.ifalselabel ":")
        || <otherstatements>.code || gen(<rest1>.iexit ":")
}
```

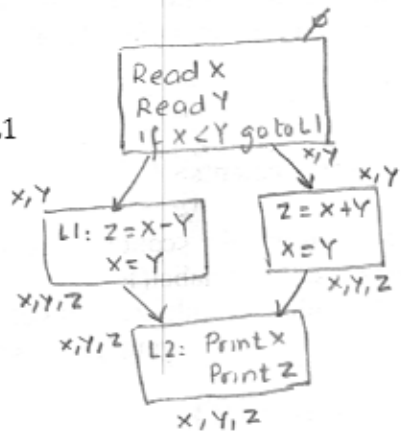
Q2. Construct the control flow graph for the following intermediate code sequence.

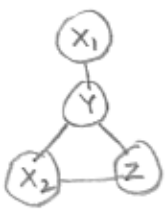
1. Read X
2. Read Y
3. X = X + Y
4. If Z < 0 go to 7
5. X = X + 1
6. Go to 8
7. X = X – 1
8. Y = Y + 1
9. T = X + Y
10. If Z == T go to 4
11. Print Z



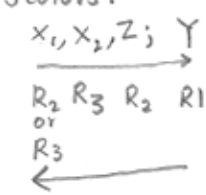
Q3. For the following code sequence: construct the control flow graph: perform liveness analysis for all variables; construct live ranges; and construct the interference graph. Color the interference graph using (a) 2 colors; and (b) 3 colors. Assume all variables are live at the end of the code sequence.

```
        Read X
        Read Y
        If X < Y go to L1
        Z = X + Y
        X = Y
        Go to L2
L1:
        Z = X – Y
        X = Y
L2:
        Print X
        Print Z
```

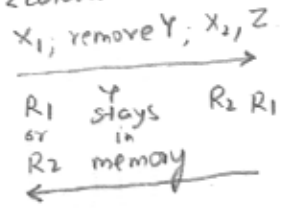




3 colors:

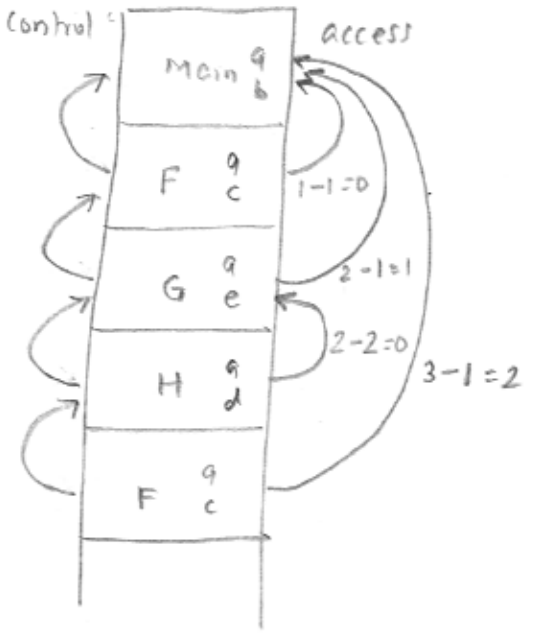


2 colors:



Q4. For the program given below show the contents of the runtime stack (activation records, local variables, control links, and access links) for the following call sequence: Main → F→ G → H → F

```
Main () {
        Int a, b;

        F() {
                Int a, c;
                Call G();
        }

        G() {
                Int a, e;
                H() {
                        Int a, d;

                        Call F();
                }

                Call H();
        }

        Call F()
}
```



(a) Show how the access links are computed and setup; and
(b) Show how various non-local variables are accessed within each function using the access links.

eg. access variable b from H.

3 − 1 = 2 (2, offset)