CS 152: Exam-II (June 11, 2020) **Name:_____**

/24

1. (4 x 6 = 24 points)

i. The storage for *static* variables declared in a *c* function
   a) can be allocated along with local variables in the stack frame
   b) can be allocated in a heap block pointed to by a global pointer
   c) can be allocated along with global variables
   d) Either (b) or (c)

ii. In a programming language without nested functions and recursion the activation frames
   a) can be statically allocated once at the start of execution
   b) must be allocated in the runtime stack
   c) must be allocated in the runtime heap
   d) none of the above

iii. Backpatching must be used sometimes
   a) during one-pass compilation
   b) during two-pass compilation
   c) during both one- and two-pass compilation
   d) backpatching it is never necessary

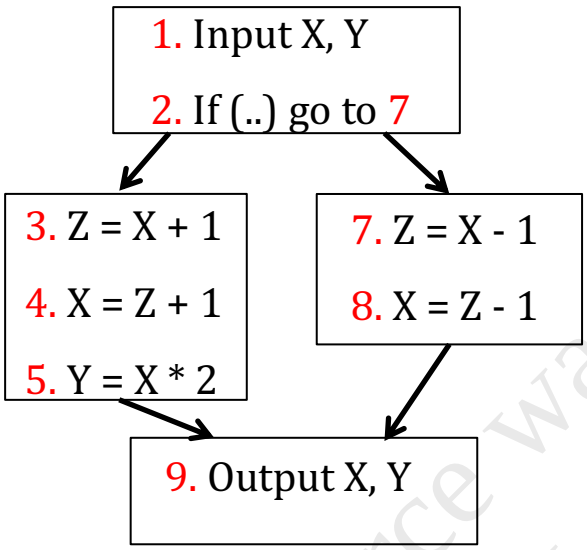
iv. In the production A → B A D, a semantic rule for evaluating an inherited attribute of non-terminal A must be placed
   a) at the beginning of the right-hand-side of the production
   b) preceding A on the right-hand-side of the production
   c) following A on the right-hand-side of the production
   d) at the end of the right-hand-side of the production

v. An intermediate code instruction *i* is a leader of a basic block if:
   a) *i* is the first instruction in a function
   b) *i* is the target of a branch instruction
   c) *i* immediately follows a branch instruction
   d) all of the above are true

vi. None of the program variables are live
   a) at the start of the program
   b) at the end of the program
   c) at both the start and end of the program
   d) inside the loops in a program

/30

2. **(3 x 10 =** 30 points) Live Ranges & Interference Graphs. Given the following control flow graph:



i. How many live ranges are there for variable X?

ii. How many live ranges are there for variable Y?

iii. How many live ranges are there for variable Z?

iv. Does a live range of X interfere with a live range of Y?

v. Does a live range of X interfere with a live range of Z?

vi. Does a live range of Y interfere with a live range of Z?

vii. What is the highest degree value in the interference graph?

viii. What is the lowest degree value in the interference graph?

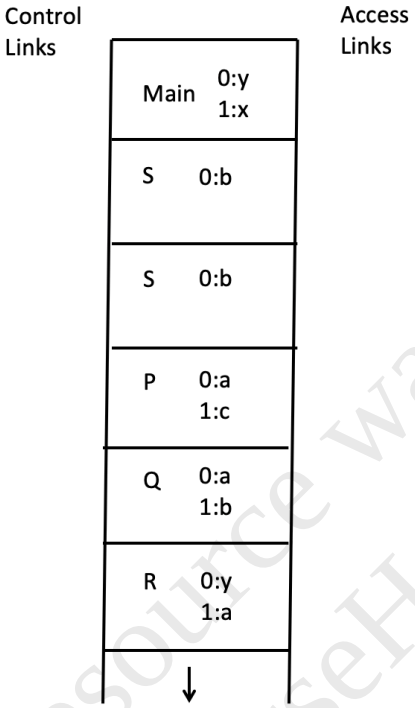
ix. What is the minimum number of colors needed to color all the live ranges in the interference graph?

x. If only two colors are available to color the interference graph, what is the minimum number of live ranges that will not be assigned a color?

/26

3. (26 points) Runtime Management: Given call sequence Main → S → S → P → Q → R

```
Main () {
  integer y, x;
  Procedure P () {
    integer a, c;
    Procedure Q () {
      integer a, b;
      Call R ();
    }
    Procedure R () {
      integer y, a;
      Call S ();
    }
    Call Q ();
  }
  Procedure S () {
    integer b;
    if (..) Call S ()
    else Call P() endif
  }
  Call S ();
}
```



In the runtime stack shown:

i. Control link of R points to _____
ii. Control link of P points to _____
iii. Access link of R points to _____
iv. Access link of Q points to _____
v. Access link of P points to _____
vi. Access link of the first frame of S points to _____
vii. Access link of the second frame of S points to _____
viii. Accessing c from R requires traversal of how many access links?
ix. Accessing x from R requires traversal of how many access links?
x. Accessing y from R requires traversal of how many access links?
xi. Setting up access link of R requires traversal of how many access links?
xii. Setting up access link of Q requires traversal of how many access links?
xiii. Setting up access link of P requires traversal of how many access links?

/20

4. (20 points) Intermediate Code Generation: The grammar below corresponds to a control construct with the following semantics. When the value of variable corresponding to **id** is less than zero, equal to zero, or greater than zero, the code in the **<less>**, **<equal>**, or **<greater>** part is executed.

<S> → switch id do <less> <equal> <greater> endswitch { <S>.code = ????? }
<less> → less <S> endless { <less>.code = <S>.code }
<equal> → equal <S> endequal { <equal>.code = <S>.code }
<greater> → greater <S> endgreater { <greater>.code = <S>.code }

(a) Provide the intermediate code that implements the following use of the construct:

```
switch a do
      less    print -a  endless
      equal   print 0   endequal
      greater print a   endgreater
endswitch
```

(b) The semantic rules for all but the first production are provided. Provide the missing code in the first production that generates the code for the entire construct and places it in <S>.code.