Final Exam

CSC 254

19 December 2016

Directions—PLEASE READ

This exam comprises a mix of multiple-choice and short-answer questions, together with one slightly longer extra-credit question. Values are indicated for each. The regular questions total 62 points. The extra credit question is worth up to 8 additional points; it isn't part of the 62, and it won't factor into your exam score, but it may help to raise your end-of-semester letter grade.

This is a *closed-book* exam. You must put away all books and notes. Please confine your answers to the space provided. For multiple choice questions, unless otherwise instructed, darken the circle next to the single best answer. Be sure to read all candidate answers before choosing. No partial credit will be given on the multiple-choice questions.

In the interest of fairness, the proctor will generally decline to answer questions during the exam. If you are unsure what a question is asking, make a reasonable assumption and state it as part of your answer.

You must complete the exam in the time allotted. Any remaining exams will be collected promptly at 3:30 pm. Good luck!

1. (**required**) Per college policy, please write out the following statement and add your signature: "I affirm that I will not give or receive any unauthorized help on this exam, and that all work will be my own."

   Signature: _____

2. (2 points) Put your name on every page (so if I lose a staple I won't lose your answers).

Multiple Choice (2 points each)

3. A language with automatic garbage collection may require extra code (a "GC barrier") to be executed on every pointer assignment. What purpose might this code serve?

◯ **a.** to update values for a reference-counting collector
◯ **b.** to maintain a list of old-to-new pointers for a generational collector
◯ **c.** to synchronize with a concurrent collector
◯ **d.** all of the above

4. In general, we prefer to do calling sequence work in the callee rather than the caller (to save code space, given that there are often many callers for a given callee). Why, then, do we make the caller responsible for saving certain registers?

◯ **a.** Because putting this work in the caller facilitates stack unwinding when an exception is thrown.
◯ **b.** Because the to-be-saved values aren't available to the callee.
◯ **c.** Because save operations in the caller can take effect while the processor would otherwise be stalled waiting for the target of the branch to be resolved.
◯ **d.** Because there is no point saving registers in the callee if their values won't be needed after the call.

5. What is the principal difference between semaphores and the condition variables of monitors?

◯ **a.** Semaphores "remember" excess V operations, while excess signals on condition variables are lost.
◯ **b.** Semaphores allow a thread to say "if !*condition* P()", while condition variables require a loop: "while !*condition* wait()".
◯ **c.** Semaphores are scheduler-based, while condition variables use spinning.
◯ **d.** Semaphores require compare-and-swap, while condition variables can be implemented with test-and-set.

6. Why aren't `Executor` tasks (`Runnable`s) in Java allowed to wait on condition variables?

◯ **a.** Because this could cause a data race.
◯ **b.** Because this could lead to severe load imbalance across cores.
◯ **c.** Because the worker thread on which the task is running might be needed to make the condition true.
◯ **d.** Because the `Executor` system was developed before condition variables were added to the language.

Questions 7 through 10 address the following pseudocode:

```
int i = 0
int f (int a, int b) {
      a += b
      a += b
      return a + i
}
int t = f(i, i+3)
print i + t
```

7. What does this code print if a and b are passed by value?

◯ **a.** 6
◯ **b.** 12
◯ **c.** 18
◯ **d.** 27

8. What does it print if a and b are passed by reference? (You may assume, as in Fortran, that an argument that is a built-up expression rather than a variable will be passed as a reference to a temporary location.)

◯ **a.** 6
◯ **b.** 12
◯ **c.** 18
◯ **d.** 27

9. What does it print if a and b are passed by value-result? (Again, assume that expression arguments are represented by temporaries.)

◯ **a.** 6
◯ **b.** 12
◯ **c.** 18
◯ **d.** 27

10. What does it print if a and b are passed by name (without memoization)?

◯ **a.** 6
◯ **b.** 12
◯ **c.** 18
◯ **d.** 27

Questions 11 and 12 address the following variable declaration in C:

```
struct {
    char a;
    int b;
    char c;
} A[10];        // 10-element array of structs
```

11. Suppose that `char` variables occupy 1 byte and that `int` variables occupy 4 bytes, which must be 4-byte aligned. How much space is consumed by array `A`? (Remember that C compilers are not permitted to reorder the fields of a struct.)

◯ **a.** 60 bytes
◯ **b.** 80 bytes
◯ **c.** 90 bytes
◯ **d.** 120 bytes

12. Suppose the programmer reversed fields `b` and `c`, so that `b` came last in the struct. Now how much space would `A` consume?

◯ **a.** 60 bytes
◯ **b.** 80 bytes
◯ **c.** 90 bytes
◯ **d.** 120 bytes

13. Which of the following languages introduced the notion of inheritance?

◯ **a.** Smalltalk
◯ **b.** Simula
◯ **c.** Algol 68
◯ **d.** C++

14. Which of the following is *not* a defining characteristic of object-oriented programming?

◯ **a.** abstraction and information hiding
◯ **b.** inheritance
◯ **c.** first-class subroutines
◯ **d.** dynamic method dispatch

15. Which of the following is the most plausible series of phases for a modern compiler?

◯ **a.** scanning, parsing, intermediate code generation, semantic analysis, machine-independent code improvement, target code generation, machine-specific code improvement
◯ **b.** scanning, parsing, semantic analysis, intermediate code generation, machine-independent code improvement, target code generation, machine-specific code improvement
◯ **c.** scanning, parsing, semantic analysis, intermediate code generation, machine-independent code improvement, machine-specific code improvement, target code generation
◯ **d.** scanning, parsing, semantic analysis, machine-independent code improvement, intermediate code generation, target code generation, machine-specific code improvement

Short Answer (3 points each)

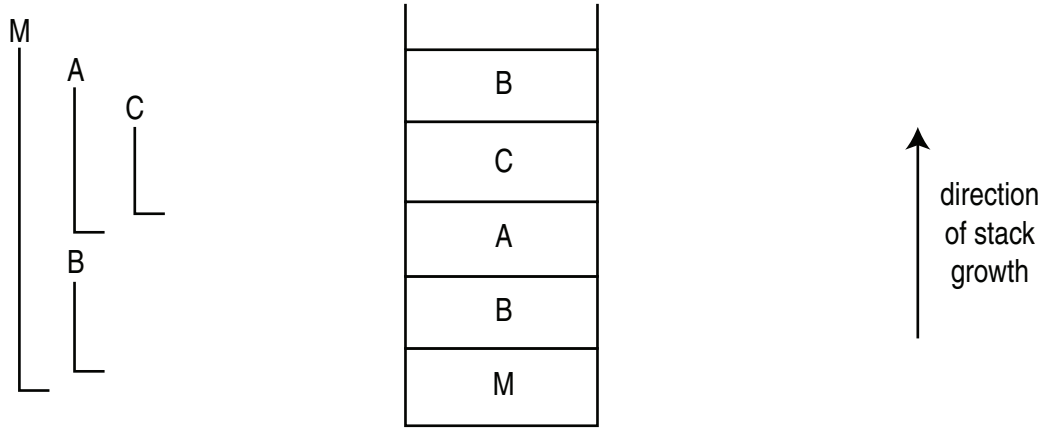
16. Consider the regular language consisting of all strings of `a`s and `b`s that contain at least one `a` and at least one `b`. How many states are there in the minimal DFA for this language?

17. Continuing the previous problem, give a regular expression for this language. (Hint: if you use "Kleene plus," you can do it in 15 characters.)

18. How should one choose whether to implement dynamic web page content on the server side (e.g., in PHP) or the client side (e.g., in Javascript)?

19. Why is pointer arithmetic less appealing today than it was in the 1970s, when C was first designed?

20. Consider a collection of four subroutines, lexically nested as shown on the left below. Suppose that M calls B, which calls A, which calls C, which calls B again. On the diagram of the stack in the middle, draw the static links.



Multiple Choice (2 points each)

21. Old compilers for the x86 typically used `push` instructions to pass arguments to subroutines. New compilers, of course, pass as much as they can in registers. When they have to use the stack, though, they typically use ordinary stores at fixed offsets from a fixed stack pointer, rather than `push`es. What accounts for the difference—why give up on `push`?

◯ **a.** The more recent convention makes it easier for the processor to pipeline the calling sequence, or to execute it out of order.

◯ **b.** The more recent convention makes it easier for the compiler to get by without an explicit frame pointer.

◯ **c.** Space for code and data is less of a scarce resource on modern machines.

◯ **d.** All of the above.

22. What is the principal purpose of type constraints on generic parameters?

◯ **a.** To control the visibility of parameter names.

◯ **b.** To determine whether a container is covariant or contravariant in the type of its elements.

◯ **c.** To allow the compiler to type-check the generic code independent of any particular instantiation.

◯ **d.** To allow the compiler to generate more efficient code than it could if parameters were entirely unconstrained.

23. Which of the following languages provides true iterators?

◯ **a.** C++

◯ **b.** Java

◯ **c.** C#

◯ **d.** OCaml

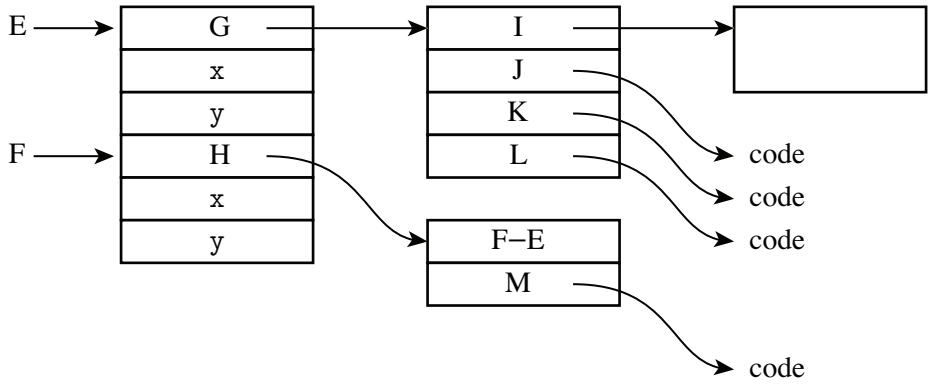
Questions 24 through 29 address the following Java program and memory layout diagram:

```
interface R {
    public int f();
}

class P implements R {
    int x;
    int y;
    public P () { x = 2; y = 3; }
    public void g() { System.out.println(x * f()); }
    public int f() { return y; }
}

class Q extends P {
    int x;
    int y;
    public Q () {
        super();
        x = 5; y = 7;
    }
    public void g() { System.out.println(x * f()); }
}

public class Obj {
    public static void main(String[] args) {
        P p1 = new Q();
        p1.g();
    }
}
```



24. What does this program print?

◯ **a.** 6
◯ **b.** 14
◯ **c.** 15
◯ **d.** 35

25. Suppose the program is translated all the way to machine code by a just-in-time or ahead-of-time compiler, using the memory layout strategy described in the textbook. In the memory layout diagram, which of the labeled pointers is likely to represent `p1`?

◯ **a.** E
◯ **b.** F
◯ **c.** G
◯ **d.** H

26. Suppose we added the line `R p2 = p1;` to the end of `main`. Which of the labeled pointers in the picture is likely to represent `p2`?

◯ **a.** E
◯ **b.** F
◯ **c.** G
◯ **d.** H

27. What is likely to be the purpose of the box referred to by the pointer labeled I?

◯ **a.** type descriptor for `P`
◯ **b.** type descriptor for `Q`
◯ **c.** vtable for `R`
◯ **d.** data for objects of some type derived from `Q`

28. What code is referred to by the pointer labeled `M`?

◯ **a.** `P`'s constructor
◯ **b.** `Q`'s constructor
◯ **c.** `Q`'s method `g`
◯ **d.** `P`'s method `f`

29. (Short Answer: 3 points) What is the purpose of the field labeled F−E?

30. (8 points **Extra Credit only**) Discuss the implications of (and the tradeoffs between) the value and reference models of variables. Note that this question is *not* (mostly) about parameter passing.