Final Exam

CSC 254

19 December 2017

Directions—PLEASE READ

This exam comprises 30 multiple-choice questions and one essay-style extra-credit question. The multiple-choice questions are 2 points each, for a total of 60 points (plus 2 for putting your name on every page). The extra credit question is worth up to 8 additional points; 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, notes, cell phones, and other electronic devices. Please confine your answers to the space provided. For multiple choice questions, 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 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. What is the defining difference between a compiler and a preprocessor?

○ **a.** A compiler produces machine language; a preprocessor produces source code.

○ **b.** A compiler performs full syntactic and semantic analysis; a preprocessor does not.

○ **c.** A compiler performs code improvement (optimization); a preprocessor does not.

○ **d.** none of the above

4. Which of the following is considered a *pure* functional language?

○ **a.** Lisp

○ **b.** ML

○ **c.** Haskell

○ **d.** Scheme

5. Consider the following context-free grammar:

$$\begin{array}{lcl} \textit{list} & \rightarrow & \textit{ids}\ \texttt{;} \\ \textit{ids} & \rightarrow & \textit{ids}\ \texttt{,}\ \texttt{id} \\ & \rightarrow & \texttt{id} \end{array}$$

Which of the following is a sentential form for this language?

○ **a.** $\texttt{id , id ,}\ \textit{ids}\ \texttt{;}$

○ **b.** $\textit{ids}\ \texttt{, id , id ;}$

○ **c.** $\textit{ids}\ \texttt{,}\ \textit{ids}\ \texttt{;}$

○ **d.** all of the above

6. Which of the following is true of attribute grammars?

○ **a.** Every S-attributed grammar is also L-attributed.

○ **b.** A symbol in an attribute grammar can have synthesized attributes or inherited attributes, but not both.

○ **c.** L-attributed grammars can naturally be evaluated during a bottom-up (LR) parse.

○ **d.** Every attribute grammar also has a natural expression as a context-free grammar with action routines.

7. Which of the following errors will always be caught by a correct implementation of C++?

○ **a.** array reference out of bounds

○ **b.** use of uninitialized variable

○ **c.** use of pointer to data that has already been freed (`delete`d)

○ **d.** none of the above

8. There are several reasons to prefer in-line functions over macros. Which of the following is *not* among these reasons?

○ **a.** better opportunities for compiler optimization
○ **b.** more conventional parameter-passing semantics
○ **c.** more predictable behavior for arguments with side effects
○ **d.** use of a separate scope

9. What does a compiler need to use to represent the referencing environment of a formal subroutine in C?

○ **a.** the static link
○ **b.** the saved contents of the display
○ **c.** the head pointer of the referencing environment A-list
○ **d.** nothing at all

Questions **??** and **??** address the following (abbreviated) `switch` statements, designed to map back and forth between whole number indices and Fibonacci numbers:

```
switch (i) {                        switch (f) {
  case 3 : return 2;                  case 2 : return 3;
  case 4 : return 3;                  case 3 : return 4;
  case 5 : return 5;                  case 5 : return 5;
  case 6 : return 8;                  case 8 : return 6;
  case 7 : return 13;                 case 13 : return 7;
  ...                                 ...
  case 49 : return 7778742049;        case 7778742049: return 49;
};                                  };
```

10. What implementation is likely to produce the best code (fast and not too wasteful of space) for the `switch` statement on the left?

○ **a.** linear search
○ **b.** jump table
○ **c.** hash table
○ **d.** binary search

11. What implementation is likely to produce the best code for the `switch` statement on the right?

○ **a.** linear search
○ **b.** jump table
○ **c.** hash table
○ **d.** binary search

12. Short-circuit evaluation of Boolean expressions

○ **a.** may sometimes succeed in cases where regular (full) evaluation would result in a run-time error.

○ **b.** will always lead to the same program behavior as regular evaluation, if both succeed.

○ **c.** reduces the worst-case number of branch instructions executed at run time.

○ **d.** all of the above

13. Which of the following is most often employed by production-quality garbage collection systems?

○ **a.** Mark-and-sweep

○ **b.** Generational collection

○ **c.** Tombstones

○ **d.** Locks and keys

Questions **??** and **??** address the following declaration in C++:

```
struct R {
    int x;
    char y;
    double z;
    bool b;
};
```

Assume that the machine on which we are running has 4-byte `int`s, 8-byte `double`s, 1-byte `char`s and `bool`s, and 4-byte alignment for `int`s and `double`s. Finally, assume that the compiler does not pack or re-order fields of structs.

14. What will be the offset, in bytes, of field `z` from the beginning of `struct R`?

○ **a.** 0

○ **b.** 6

○ **c.** 8

○ **d.** 12

15. What will be the size, in bytes, of a ten-element array of `struct R`?

○ **a.** 140

○ **b.** 160

○ **c.** 200

○ **d.** none of the above

Questions **??** through **??** consider contiguous and row-pointer implementations of a 5-column 2-dimensional array in C. If passed as a parameter to a function, the contiguous array might be declared `int A[][5]`; the row-pointer array might be declared `int* B[]`. Both would permit double-subscripting in the body of the function: `x = A[i][j]; y = B[i][j]`. Suppose `i`, `j`, `x`, `y`, and the addresses of `A` and `B` are all being kept in registers, but the contents of `A` and `B` are in memory. Suppose further that our hardware provides not only add, multiply, and load instructions, but also a *scaled* load instruction, which will load `V[k]` into a register as a single operation, given that `V` is the address of an array of integers or pointers, `k` is an integer index, and both `A` and `k` are in registers.

16. How many instructions does it take to load `A[i][j]` into `x`?

○ **a.** 2 multiplies, 2 adds, and a load
○ **b.** 2 adds and 2 loads
○ **c.** a multiply, an add, and a scaled load
○ **d.** 2 scaled loads

17. How many instructions does it take to load `B[i][j]` into `y`?

○ **a.** 2 multiplies, 2 adds, and a load
○ **b.** 2 adds and 2 loads
○ **c.** a multiply, an add, and a scaled load
○ **d.** 2 scaled loads

18. Which access—to `A[i][j]` or `B[i][j]`—is likely to be faster on a modern machine?

○ **a.** B, because it takes fewer instructions.
○ **b.** B, because row pointer accesses are more easily pipelined.
○ **c.** A, because arithmetic is faster than memory access today.
○ **d.** neither one: they're likely to be equally fast.

19. When `A` and `B` are declared as function parameters, why does `A` have to specify its row length but `B` doesn't?

○ **a.** Because the compiler is able to infer the length in the row-pointer case.
○ **b.** Because accesses in `A` must multiply by the row length.
○ **c.** Because C performs bounds checking for contiguous arrays but not for row-pointer arrays.
○ **d.** None of the above: all lengths are optional when declaring array parameters in C.

20. C uses zero for the lower bound on all array indices. Why?

○ **a.** To avoid the cost of subtracting off a nonzero bound at run time.
○ **b.** To simplify the syntax of array declarations.
○ **c.** To facilitate the interoperability of arrays and pointers.
○ **d.** To allow row-pointer and contiguous arrays to use the same notation.

21. Under *name equivalence*, two variables have the same type if

○ **a.** they have the same internal structure.

○ **b.** they were declared using the same lexical occurrence of a type constructor.

○ **c.** they can hold the same set of values.

○ **d.** they support the same set of methods.

22. In OCaml, if `foo` is known to be of type `'a * bool` and `bar` is known to be of type `int * 'b`, what is the type of `if flag then foo else bar`?

○ **a.** bool

○ **b.** `int * bool`

○ **c.** `'a * bool | int * 'b`

○ **d.** It depends on the value of `flag`.

23. The counter in a semaphore represents

○ **a.** the number of available resources—the number of times that P could be called without waiting.

○ **b.** the number of waiting threads.

○ **c.** the duration of the current *lease*—the time remaining until the semaphore resource will be released.

○ **d.** the total number of V operations that have been called on the semaphore since it was initialized.

24. Consider the following program:

```
// Globals:
Widget w;
Boolean ready = false;

// Thread 1:
w = new Widget();               // Thread 2:
ready = true;                   while (!ready);     // spin
                                w.print();
```

This program has data races on both `ready` and `w`. The programmer's intent is clearly to make thread 2 wait until `w` has been initialized before trying to use it. Why might the program not work as expected?

○ **a.** The compiler could reorder the assignments in thread 1, or prefetch fields of `w` in thread 2.

○ **b.** The processor could reorder instruction execution in either thread.

○ **c.** The memory system could propagate the new value of `ready` from thread 1 to thread 2 before it propagates the new value of `w`.

○ **d.** All of the above.

Questions **??** through **??** address the following pseudocode:

```
function f (a, b)
    a++
    b++
    return a
. . .
i := 0;   M[0] := 1;   M[1] := 2
t = f (i, M[i])
print i + M[1] + t
```

25. What will this code print if all parameters are passed by value?

○ **a.** 3
○ **b.** 4
○ **c.** 5
○ **d.** 6

26. What will it print if all parameters are passed by reference?

○ **a.** 3
○ **b.** 4
○ **c.** 5
○ **d.** 6

27. What will it print if all parameters are passed by value/result?

○ **a.** 3
○ **b.** 4
○ **c.** 5
○ **d.** 6

28. What will it print if all parameters are passed by name?

○ **a.** 3
○ **b.** 4
○ **c.** 5
○ **d.** 6

29. What is the principal reason to employ both a frame pointer and a stack pointer?

○ **a.** To make use of special hardware instructions that embed the two-pointer assumption.
○ **b.** To make it easier to find the address of the nearest handler when an exception is thrown.
○ **c.** To protect the stack from being overwritten by Linux signal handlers.
○ **d.** To handle cases is which the size of the frame is not statically known.

30. Which of the following is *not* a function of the "trampoline" code used to implement asynchronous signals?

    ○ **a.** Launch a thread to handle the signal.

    ○ **b.** Execute the calling sequence for the signal handler.

    ○ **c.** Re-enable signals when the handler has finished running.

    ○ **d.** Restore the state of the interrupted program before returning.

31. Why does Java provide mix-in inheritance instead of full multiple inheritance?

    ○ **a.** Because it avoids discontiguous objects.

    ○ **b.** Because it imposes no costs on programs that use only single inheritance.

    ○ **c.** Because it avoids semantic ambiguity when the same field is inherited over two different paths in the inheritance tree.

    ○ **d.** All of the above.

32. When declaring a class in a `.h` file in C++, one can provide just the headers of methods, leaving the code to a `.c` file that is not visible to callers. One must, however, declare all fields (data members) of the class, even if they are private, and thus never visible outside the class. Why is this required?

    ○ **a.** To make fields visible to derived classes.

    ○ **b.** To allow the compiler to call field constructors.

    ○ **c.** To increase maintainability: the `.h` file serves as the documentation of the class, and the fields are an essential part.

    ○ **d.** To allow the compiler to calculate the size of objects when allocating them as values.

Extra Credit

33. (8 points max) As you may recall, the generics of Ada, C++, Java, C#, and others provide a form of explicit parametric polymorphism. Other languages, including Lisp, ML, Smalltalk, and most scripting languages, provide *implicit* parametric polymorphism: functions naturally accept arguments of any type that supports the operations applied to them by the function. Discuss the comparative strengths and weaknesses of explicit and implicit parametric polymorphism. For maximum points, illustrate your discussion with examples.