Directions; PLEASE READ

This exam comprises a mixture of multiple-choice and short-answer questions. Values are indicated for each; they total 100 points. Question 34 is worth up to 8 extra credit points; this won’t factor into your exam score, but may help to raise your end-of-semester letter grade.

This is a closed-book exam. You must put away all books, notes, and 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 clarify any 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 will have a maximum of three hours to complete the exam, though hopefully you won’t need it all. Any remaining exams will be collected promptly at 3:30 pm. Good luck!

1. (required) Per college policy, please write out the following statement by hand, 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.”

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

Multiple Choice (3 points each)

3. Which of the following is not considered an essential feature of object-oriented programming?
   ○ a. encapsulation
   × b. generics
   ○ c. dynamic method dispatch
   ○ d. inheritance

4. Why are in-line functions useful?
   ○ a. They eliminate subroutine-call linkage overhead.
   ○ b. They enable the compiler to perform more code improvement.
   ○ c. They may improve I-cache locality.
   × d. All of the above.
5. Why might a compiler choose not to in-line a function?

   a. It may be logically impossible.
   b. It may worsen I-cache locality.
   c. It may increase code size.
   d. All of the above.

6. Why do most languages not specify the order of evaluation of arguments?

   a. Because it doesn’t change the meaning of programs.
   b. Because it’s already constrained by the precedence and associativity rules.
   c. Because many important code improvement techniques depend on being able to change the order.
   d. Because the order can’t be specified with context-free rules.

7. Which of the following is least likely to be used to generate code for a switch statement?

   a. hash table
   b. linear sequence of tests
   c. balanced search tree
   d. binary search in an array

8. What is the principal advantage of generational garbage collection, compared to mark-and-sweep?

   a. It avoids the need to repeatedly scan long-lived objects.
   b. It can be implemented in a language like C, which is not strongly typed.
   c. It requires roughly half the heap space.
   d. It has much better cache behavior.

Short Answer (4 points each)

9. Why is the distinction between mutable and immutable values important when implementing a language with a reference model of variables?

   Answer: Because the compiler can safely create copies of immutable values, and avoid the overhead of indirection. (Note that this is not a question about why it’s useful to have const parameters.)

10. A Java compiler will reject, at compile time, all programs that use an uninitialized variable. Determining exactly which uses these are, however, is undecidable (equivalent to the halting problem). How does Java get around this seeming contradiction?

    Answer: By rejecting a broader set of programs, including some that do not use an uninitialized variable.

11. Why is it desirable, from a compiler-writer’s perspective, for the step size of a for loop to be a compile-time constant?

    Answer: It simplifies compilation by allowing the the direction of the end-of-loop test (< or >) to be statically determined. It may also simplify loop unrolling and related optimizations, but those are possible even with bounds that are not known until run time. (Note that this is not a question about why it’s useful to not to change the bounds of a loop from within the loop body: that’s generally a good idea even with bounds that are not compile-time constants.)
12. How can a compiler accommodate step sizes that are not known at compile time?

**Answer:** Don’t test the index to detect termination; rather, calculate the number of iterations in advance, count down one per iteration, and stop when you reach zero.

13. Why doesn’t a program run any faster if you leave out optional parameters in subroutine calls?

**Answer:** Because the compiler inserts the missing parameters for you; the work performed at run time is exactly the same.

14. Older compilers often pushed subroutine arguments onto the stack one at a time, using instructions that automatically updated the stack pointer on each push. Modern compilers typically preallocate the space for in-stack arguments, and store into that space using static offsets from the stack pointer. What might explain the change?

**Answer:** Pre-allocating the space is more acceptable on modern machines, which have lots of memory, and it allows arguments to be computed (and stored) out of order, if the compiler’s analysis suggests that this will result in faster code. It also allows back-up space to be reserved in the stack, at zero cost, for arguments that are passed in registers.

**Multiple Choice**

15. (3 points) Why does OCaml require that all elements of a list have the same type (in contrast to Scheme, which allows them to have different types)?

- a. So that the language implementation can perform all type checks at compile time.
- b. Because the language designers believed that heterogeneous lists were bad programming style.
- c. To eliminate the need for generics.
- d. Because heterogeneous lists are incompatible with OCaml’s richer syntax (in contrast to Scheme, where everything looks like a parenthesized list anyway.)

16. (3 points) In which of the following situations is a spin lock an appropriate mechanism for mutual exclusion?

- a. Synchronization among threads of a scientific simulation, running on a parallel supercomputer.
- b. Synchronization between a graphical program and its signal handlers.
- c. Synchronization between processes connected by a Unix pipe (as in \texttt{grep} foo *.c | less).
- d. Control of access to a critical section that writes data to a shared file.
Questions 17–20 (3 points each) refer to the following declarations, in an implementation of C on a machine with 8-byte doubles, 2-byte shorts, and 1-byte chars, each of which must be stored in memory at an address that is a multiple of its size.

```c
struct foo {
  double d;
  char c;
  short s;
};
struct foo v;
struct foo A[10];
struct foo B[10][10] = ...
struct foo *C[10] = ...

double x = B[i][3].d;
double y = C[i][3].d;
```

Note that under C layout rules, B will have contiguous layout, while C will have row-pointer layout.

17. What is the offset of field s in v?
   - a. 9 bytes
   - b. 10 bytes
   - c. 12 bytes
   - d. None of the above

18. What is the overall size of A?
   - a. 110 bytes
   - b. 120 bytes
   - c. 160 bytes
   - d. None of the above

Now consider the assignments to x and y. Assume that i is a variable whose value cannot be determined at compile time, but which has already been loaded into a register at run time. Assume further that the machine provides scaled indexing, so \( w + z \times p \) can be calculated with a single addition instruction, if \( p \) is the size of a pointer and \( w \) and \( z \) are in registers.

19. How many operations will be required to load the right-hand size of the declaration of x into a register (from which it could subsequently be stored to memory)? Assume that the language implementation calculates whatever it can compile time.
   - a. one multiplication, one addition, and a load
   - b. one multiplication, two additions, and a load
   - c. two multiplications, two additions, and a load
   - d. two additions and two loads
20. How many operations will be required to load the right-hand size of the declaration of \( y \) a register (from which it could subsequently be stored to memory)? Again, assume that the language implementation calculates whatever it can at compile time.

- a. one multiplication, one addition, and a load
- b. one multiplication, two additions, and a load
- c. two multiplications, two additions, and a load
- d. two additions and two loads

Questions 21–24 (2 points each) refer to the following pseudocode.

```c
int a := 1
int b := 2
print f(a, a+b)
print a+b
```

```c
int f(int i, int j) {
   ++i; ++b
   print f(a, a+b)
   print a, j
   return i + j
}
```

**Answer:** This was a tricky set of questions. Some languages do not allow a built-up expression (an r-value) to be passed by reference or value-result. If they do, implementations must create a temporary to hold the argument, and throw it away after the function returns.

21. What will this code print if \( i \) and \( j \) are passed by value?

- a. 1, 3, 5, 4
- b. 1, 3, 5, 5
- c. 2, 3, 5, 5
- d. 2, 5, 7, 5

22. What will this code print if \( i \) and \( j \) are passed by reference?

- a. 1, 3, 5, 4
- b. 1, 3, 5, 5
- c. 2, 3, 5, 5
- d. 2, 5, 7, 5

23. What will this code print if \( i \) and \( j \) are passed by value/result?

- a. 1, 3, 5, 4
- b. 1, 3, 5, 5
- c. 2, 3, 5, 5
- d. 2, 5, 7, 5

24. What will this code print if \( i \) and \( j \) are passed by name?

- a. 1, 3, 5, 4
- b. 1, 3, 5, 5
- c. 2, 3, 5, 5
- d. 2, 5, 7, 5
25. (3 points) Consider the following C++ code.

```cpp
#include <iostream>
using std::cout, std::endl;

class Bar {
  int q;
public:
  Bar(const int n) {q = n;}
  Bar(const Bar& other) {q = other.q; cout << '*';}
  int op(const Bar c) {return q + c.q;}
};

int foo(const Bar& one, Bar two) {
  Bar a = one;
  two = a;
  return a.op(two);
}

int main() {
  Bar b1(1);
  Bar b2(b1);
  cout << foo(b1, b2) << endl;
}
```

When this code is run, how many times will the copy constructor for Bar be called (i.e., how many asterisks will be printed)?

- [ ] a. 3
- [x] b. 4
- [ ] c. 5
- [ ] d. 6

26. (3 points) Why doesn’t Java allow you to specify built-in types as generic parameters?

- [ ] a. Because it would preclude type-checking the definition of the generic itself: Java would have to type-check instantiations, the way C++ does.
- [x] b. Because generic parameters of built-in type would violate the language designers’ desire to “make everything an object.”
- [x] c. Because Java’s erasure mechanism requires that every object of generic type be a reference, not a value.
- [ ] d. Because the need to handle both built-in types and references would impose unacceptable overhead on just-in-time compilation.
27. (3 points) How does a virtual machine differ from an “ordinary” run-time system?

- a. It is customized for a specific programming language.
- b. It is customized for a specific hardware architecture.
- c. It declares key components without defining them, for the sake of dynamic method binding.
- d. It provides everything needed by the program that runs above it.

Questions 28–30 (2 points each) consider a thread-safe partition abstraction that comprises a collection of objects and buckets. It maintains the invariant that a given object lies in exactly one bucket at any given point in time. A move method of the abstraction serves to change the bucket in which an object lies (i.e., to move it from one bucket to another), atomically. One possible implementation uses a single lock to protect the entire partition. Another uses a separate lock for every bucket.

28. What is the principal drawback of the single-lock implementation?

- a. The compiler has no way to determine whether the lock should be implemented by spinning or by blocking.
- b. It’s not clear how to structure the code, given that every bucket needs to refer to the same, single lock.
- c. The code has no concurrency: only one thread can update the partition at a time.
- d. The code may deadlock if two threads try to operate on the same bucket concurrently.

29. Why might the separate-lock-per-bucket implementation be slower than the single-lock implementation?

- a. Because the implementation has to search among multiple locks at run time, to find the right ones to access.
- b. Because with multiple locks we’ll need to use an implementation based on blocking instead of spinning.
- c. Because the extra locks increase the size of the partition, reducing cache performance.
- d. Because every move operation has to acquire and release two locks instead of one.

30. In the separate-lock-per-bucket implementation, why might the move operation sometimes acquire the lock on the “from” bucket first and sometimes acquire the lock on the “to” bucket first?

- a. To avoid deadlock by always acquiring locks in some canonical order (e.g., in order of address).
- b. To increase concurrency, by acquiring the other lock first, if one is currently held.
- c. To maximize cache locality.
- d. To improve the performance of the processor pipeline.
Short Answer (5 points each)

31. Describe, in terse, high-level English, the language generated by the following regular expression: \( a^+ b (a \mid b)^* \mid b^+ a (a \mid b)^* \)

Do not provide a low-level mechanical explanation of the notation.

**Answer:** Strings of \( a \)'s and \( b \)'s containing at least one of each.

32. Explain why the following OCaml function is difficult to translate into C.

```ocaml
let plus_n n = fun x -> n + x;;
```

**Answer:** First, it employs a lambda expression, which C lacks. (C actually lacks nested functions of any kind.) Second, it requires \( n \) to have unlimited extent, which C also lacks. (BTW, this is not a question about currying, except in the sense that \( \text{plus}_n \) is a manually curried version of the more familiar + function.)

33. Consider the following context-free grammar:

\[
\begin{align*}
expr & \rightarrow \text{num} \mid \text{id} \mid \text{fn\_call} \\
\text{fn\_call} & \rightarrow \text{id} (\ \text{args}\ ) \\
\text{args} & \rightarrow \text{expr} \ \text{more\_args} \mid \epsilon \\
\text{more\_args} & \rightarrow , \ \text{expr} \ \text{more\_args} \mid \epsilon
\end{align*}
\]

Explain why this grammar is not LL(1).

**Answer:** \( \text{id} \in [\text{PREDICT}(\text{expr} \rightarrow \text{id}) \cup \text{PREDICT}(\text{expr} \rightarrow \text{fn\_call})] \). Equivalently, when expecting an \( \text{expr} \) and looking at an \( \text{id} \), we won’t be able to tell whether to use the second or third production.
34. (Extra Credit: up to 8 points) The concurrent functional language Multilisp is a dialect of Scheme with one additional built-in functional form, called \texttt{future}. Any expression in Multilisp can be embedded inside a future:

\[(\text{future old-expression})\]

The future arranges for its embedded expression to be evaluated by a separate thread of control. The parent thread continues to execute until it actually tries to use the expression’s value, at which point it waits for execution of the future to complete. If two or more arguments to a function are enclosed in futures, then evaluation of the arguments can proceed in parallel:

\[(\text{parent-func (future arg-expr-1) (future arg-expr-2)})\]

Discuss the circumstances under which a Scheme programmer can safely and profitably insert futures into a program. Hint: consider the use of functional and imperative language features. Also consider the overhead of thread creation, scheduling, and synchronization.

\textbf{Answer:} In a purely functional program, \texttt{future} is semantically neutral: since Scheme uses applicative-order (eager) evaluation, program behavior will be exactly the same as if the embedded expression had appeared without the surrounding call. In a program that uses imperative features of Scheme, the Multilisp programmer must ensure that there are no races between the argument to a future and any other code that might be executed in parallel.

Even in code that is purely functional, the programmer should consider whether an expression is complicated enough to warrant evaluation by a separate thread. Passing too small an expression to a future can result in a loss in performance, if the parent thread spends more time forking and joining with the child than it would have spent evaluating the expression itself. Passing an expression to a future also makes no sense unless a “sibling” argument is also made a future, or unless the parent will not need the value for a while.