Comprehensive Exam: Programming Languages Autumn 2001

This is a 30-minute closed-book exam and the point total for all questions is 30.

All of the intended answers may be written within the space provided. You may use the back of the preceding page for scratch work. If you to use the back side of a page to write part of your answer, be sure to mark your answer clearly.

The following is a statement of the Stanford University Honor Code:

A. *The Honor Code is an undertaking of the students, individually and collectively:*

   (1) *that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;*

   (2) *that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.*

B. *The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.*

C. *While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.*

By writing my "magic number" below, I certify that I acknowledge and accept the Honor Code.

____

(Number)

Prob	# 1	# 2	# 3	# 3	# 3	Total
Score						
Max	4	4	4	8	10	30

54

1. (4 points) *Lifetime and Scope.* Define *lifetime of a data object* and *scope of a declaration*. Describe a situation in which two variable declarations with the same scope produce two locations with different lifetimes.

2. (4 points) *Static and Dynamic Scope.* Explain the difference between static and dynamic scope. Give an example of a language or language feature that uses dynamic scope.

3. (4 points) *Tail Recursion.* What is the primary advantage of optimizing tail recursive functions so that recursive calls are eliminated? Specifically, suppose that a call `f(n)` to an unoptimized tail-recursive function `f` runs in time and space $O(\mathtt{n})$. If an optimizing compiler performs the optimization commonly called "tail recursion elimination," what are the time and space requirements of a call `f(n)`? Use "big-oh" notation; your answers should be $O(1)$ or $O(\mathtt{n})$ or $O(\mathtt{n}^2)$ or something of this form.

55

4. (8 points) *Parameter Passing*
The C programming language uses pass-by-value parameter passing, while C++ also has pass-by-reference.

(a) (4 points) Describe the sequence of storage allocations and assignments associated with the execution of the function call `increment(y)` in the following C fragment.

```
void increment (int x) {
     x++;
};
int y=0;
...
increment(y);
```

Explain why the call `increment(y)` does not increment y.

(b) (1 points) Write the call to `increment`, passing y, so that the call to `increment` in the following fragment increments the value of y.

```
void increment (int * x) {
     (*x)++;
};
int y=0;
...
<call_to_increment>;
```

52

(c) (3 points) Is the following C++ code, using pass-by-reference, more efficient than the pass-by-value code in part (a)? Explain. Why is pass-by-reference often more efficient?

```
void increment (int & x) {
     x++;
};
int y=0;
...
increment(y);
```

5. (*10 points*) *Type Conversion*
This question asks you to read two short passages from documents explaining C++ and comment on their content and the connections between them.

Explanation of type conversion, from "C++ in Hypertext":
Most operators assume that their operands are of the same type. Thus, it is not possible to add an integer and a string - what would be the meaning of such an operation? However, if this rule of 'sameness' were enforced too strictly, it would make C++ an awkward language. For example, it makes sense to add an integer to a long or to a double. To handle this C++ allows what are called implicit conversions (also called type coercions) of one operand to the type of a second operand - when the conversion is appropriate. In the example involving the addition of an int and a long, the integer would be converted to a long. In the example involving an int and a double, the integer would be converted to a double. Such conversions are called 'implicit' conversions because they take place automatically.

C++ provides a set of conversion rules (also called 'promotion' rules) that guide the type of implicit conversions that take place.

The built-in types of C++ are arranged in order from 'lower' to 'higher' types, where a value of a lower type can be implicitly converted to a value of a higher type. 'Higher' here essentially means that the type can hold all the information contained in a 'lower' type and no information is lost in a conversion. Thus, C++ will implicitly convert an int to a double but not a double to an int. In the latter case, information is lost when the 'data' to the right of the decimal point is truncated.

57

Here is part of the conversion hierarchy:

```
double
float
long
int
char
```

Explanation of type conversion, from "C++ FAQ Lite":
Is it OK to convert a pointer from a derived class to its base class?

Yes. An object of a derived class is a kind of the base class. Therefore the conversion from a derived class pointer to a base class pointer is perfectly safe, and happens all the time. For example, if I am pointing at a car, I am in fact pointing at a vehicle, so converting a Car* to a Vehicle* is perfectly safe and normal:

```
void f(Vehicle* v);
void g(Car* c) { f(c); }  // Perfectly safe; no cast
```

Questions

(a) (*1 points*) Based on the first passage, what do you believe are the relative sizes (number of bytes) of `double, float, long, int, char`.

(b) (*2 points*) Why does this passage "'Higher' here essentially means that the type can hold all the information contained in a 'lower' type and no information is lost in a conversion." suggest that there is a correlation between implicit conversion and the number of bytes used to represent a datum?

50

(c) (*2 points*) If `Car` is a derived class with public base class `Vehicle`, which class of object is likely to require a larger representation in memory?

(d) (*2 points*) Is the order of conversion from `Car *` to `Vehicle *` consistent with the order of conversation from `int` to `float`? Explain.

(e) (*3 points*) Does it make more sense to convert from `Car` to `Vehicle` or `Vehicle` to `Car`? Explain one problem with each form of conversion.

57