COP4020 Fall 2004 – Final Exam

Name: _____ (Please print)
Put the answers on these sheets. You can collect 100 points in total for this exam.

1. Which C construct is classified as a *selection* statement? (mark **one**) (4 points)

   (a) `return`
   (b) `switch`
   (c) `break`
   (d) `while`

2. In which of the following situations can *expression evaluation reordering* by a compiler (using the associative and commutative properties of arithmetic operators) change the value of an expression at runtime or cause an exception that did not occur in the original expression? (mark **one or more** (4 points)

   (a) The expression includes arithmetic with extremely large positive and negative values.
   (b) The expression is referentially transparent.
   (c) Side effects occur in at least one of the operands in the expression.
   (d) None of the above.

3. What is the single *parameter passing mode* that C supports (excluding macros)? (mark **one**) (4 points)

   (a) By name
   (b) By reference
   (c) By sharing
   (d) By value

4. *Tail recursion optimization* replaces one or more recursive calls with a jump instruction (in combination with adjustments to parameters). At which points in the following program is tail recursion optimization applicable? (mark **one**) (4 points)

```
int foo(int n)
{ if (n == 0)
    return 1;           // <== [1]
  if (n % 2 == 0)
    return 2*foo(n/2); // <== [2]
  return foo(n-1);     // <== [3]
}
```

   (a) At [1]
   (b) At [2]
   (c) At [3]
   (d) At [2] and [3]

5. Which terms *unify* in Prolog? (mark **one or more**) (4 points)

   (a) `foo(X) = foo(1,2)`
   (b) `a = 4`
   (c) `plus(Y,1) = X`
   (d) `A = 3`

6. Consider the following Prolog facts and rules. Which *query* succeeds and returns YES? (mark **one**) (4 points)

```
ok(fsu,uf).
ok(fsu,miami).
ok(miami,uf).
s([X]).
s([A,B|T] :- ok(A,B), s([B|T]).
```

   (a) `s([miami,fsu,uf])`

   (b) `s([])`

   (c) `s([fsu,miami,uf])`

   (d) `s([osu,fsu])`

7. Which expression(s) are valid *l-values* in C/C++? (mark **one or more**) (4 points)

   You may assume the variables are declared as:

```
int n, a[2], *p;
```

   (a) `n+1`

   (b) `*(p+1)`

   (c) `a[n]`

   (d) `3`

8. What is the definition of a *subroutine closure*? (mark **one**) (4 points)

   (a) A `finally` clause in an exception handler to close files and clean up other resources in a subroutine.

   (b) A reference to a subroutine, together with its referencing environment passed as a parameter.

   (c) A close-up code fragment of a subroutine.

   (d) A hidden subroutine that evaluates the actual parameter in the caller's referencing environment to support parameter passing by name.

9. Parameters are always passed by reference in Fortran 77. But you can also pass (the values of) expressions. This is not a contradiction. Explain how this is done in Fortran. (7 points)

10. (7 points) Parameter passing in Fortran 77 can lead to several interesting issues. Consider for example the subroutine:

```
SUBROUTINE SHIFT(A, B, C)
INTEGER A, B, C
A = B
B = C
END
```

Suppose we want to call `SHIFT(X, Y, 0)` but we don't want to change the value of `Y`. We know that expressions can be passed, so we immediately realize that we can use an expression for the second parameter `SHIFT(X, Y+0, 0)`. However, it appears that our trick doesn't work with a particular brand of optimizing Fortran compilers. Explain why this approach is fragile.

11. (7 points) Consider the following outline of a QuickSort procedure in Pascal:

```
procedure QuickSort(var A: Array)
   procedure Partition(var A: Array; j,k: integer)
      var pivot: integer;
      procedure Swap(var A: Array; s,t: integer)
         var temp: Type;
      begin (* Swap *)
         ... <== [1]
      end; (* Swap *)
   begin (* Partition *)
      ... <== [2]
   end; (* Partition *)
begin (* QuickSort *)
   ... <== [3]
end; (* QuickSort *)
```

Indicate which variables, arguments, and subroutines are in scope at the indicated points [1], [2], and [3] in the program. Mark the variables and arguments with the procedure that defines it (for example, at [1] `A` of `Swap` is in scope).

12. (6 points) Consider the following pseudo-code program

```
var x : integer; /* global variable */

procedure Update
   x := x + 1;

procedure DoCall(P: procedure)
   var x : integer;
   x := 1;
   P();
   write_integer(x);

begin /* body of main program */
   x := 0;
   DoCall(Update);
end /* body of main program */
```

Assuming static scoping or dynamic scoping rules are used (with shallow or deep bindings). What value does the program print?

	Static Scoping	Dynamic Scoping with Shallow Binding	Dynamic Scoping with Deep Binding
Output:			

13. (6 points) Consider the following program:

```
procedure do_the_math(x, y, z)
    begin /* do_the_math */
        x := x * y;
        y := x + z;
    end; /* do_the_math */

begin /* body of main program */
    integer a := 2;
    integer b := 3;
    do_the_math(a, b, a);
    write_integer(b)
end /* body of main program */
```

For each of the parameter passing modes shown in the table below show the value printed by the program.

	By value	By reference	By value/result
Output:			

Note: for passing by value/result, you should assume that parameter `z` is passed as an in-mode parameter to prevent its value from being passed back out.

14. (6 points) Consider the following sequence of assignments

```
a = b + c
d = b + c + e
f = e + a + g
```

Optimize the sequence of assignments by eliminating as many common subexpressions as you can.

15. (6 points) The following loop attempts to produce a sequence of dollar amounts 1.00, 1.01, 1.02, ... 10.00 to sum up totals.

```
for (float cost = 1.00; cost <= 10.00; cost += 0.01)
{ if (verify_cost(cost))
    add_dollar_amount(cost);
}
```

We saw that this approach can lead to problems. Rewrite the loop to use an integer loop counter `cents`. In the loop compute the value of `cost` as a function of `cents`.

16. Consider the following Prolog program

```
box(yellow, hotweels).
box(yellow, puzzle).
box(red, dress).
box(red, shoes).
box(blue, book).
addressed_to(yellow, alex).
addressed_to(blue, alex).
addressed_to(red, becky).
addressed_to(green, chris).
gift(Person, Gift) :- addressed_to(Color, Person), box(Color, Gift).
no_gift(Person) :- addressed_to(Color, Person), not box(Color, Gift).
```

(a) What is the first value of `X` returned by the query `gift(alex, X)`? (5 points)

(b) Which person does not receive a gift? That is, what is the value of `X` returned by the query `no_gift(X)`? (5 points)

17. Consider the following C++ program with exceptions:

```
class GeneralException { };
class SpecificException : public GeneralException { };
class TerminateException : public SpecificException { };

void foo(int n)
{ try
  { if (n == 1)
      throw GeneralException();
    if (n == 2)
      throw SpecificException();
  }
  catch (SpecificException)
  { cerr << "Specific exception in foo" << endl;
  }
}

int main()
{ int k = ???; // <= see questions
  try
  { foo(k);
    throw TerminateException();
  }
  catch (SpecificException)
  { cerr << "Specific exception in main" << endl;
  }
  catch (GeneralException)
  { cerr << "General exception in main" << endl;
  }
}
```

(a) (5 points) Which exception(s) can `foo` raise that are propagated to the caller?

(b) (8 points) Suppose we assign 0, 1, or 2 to `k` at the point indicated in the program, then for `k=0`, which messages are produced (**mark zero, one, or more**)?

	Specific exception in foo
	Specific exception in main
	General exception in main

for `k=1`, which messages are produced (**mark zero, one, or more**)?

	Specific exception in foo
	Specific exception in main
	General exception in main

for `k=2`, which messages are produced (**mark zero, one, or more**)?

	Specific exception in foo
	Specific exception in main
	General exception in main