COP4020 Spring 2010 – Final Exam

Name: ______________________________ (Please print)

Put the answers on these sheets. You can collect 100 points in total for this exam.

1. Consider the following C program fragment:

   ```
   int foo(int bar)
   {
     return bar + 1;
   }
   ```

   Where is the value of the argument `bar` allocated and stored at run time? (mark **one**) (4 points)

   (a) Statically allocated as global data for function `foo`
   (b) Stack allocated in the subroutine frame of function `foo`
   (c) Dynamically allocated on the heap for function `foo`
   (d) Depends on the compiler implementation

2. Consider the following Scheme program fragment:

   ```
   (define foo
     (lambda (bar)
       (let ((result (list 1 bar)))
         result
       )
     )
   )
   ```

   Where is the list data structure in `result` allocated at run time? (mark **one**) (4 points)

   (a) Statically allocated as global data for function `foo`
   (b) Stack allocated in the subroutine frame of function `foo`
   (c) Dynamically allocated on the heap for function `foo`
   (d) Allocated when the Scheme program was loaded into memory

3. Axiomatic semantics is a framework to prove the *partial correctness* of a program. What does this term mean? (mark **one**) (4 points)

   (a) it means that an axiomatic semantics proof of a program may or may not prove the correctness of that program
   (b) it means that the postcondition holds regardless of the precondition
   (c) it means that the precondition and program termination imply that the postcondition holds
   (d) it means that the precondition, program termination, and the postcondition all hold

4. Consider the following pseudo code program fragment:

```
procedure P(a : integer)
   var b : integer;
   procedure Q(c : integer)
      var d : integer
      procedure R(e : integer)
         var f : integer
      begin (* of R *)
         ...
      end (* of R *)
   begin (* of Q *)
      ... <= [*]
   end (* of Q *)
begin (* of P *)
   ...
end (* of P *)
```

Suppose the programming language implements the *closest nested scope rule*. Which arguments and variables are visible at the program point indicated by `<= [*]` ? (mark **one**) (4 points)

(a) `a b c d e f`

(b) `a b c d`

(c) `c d`

(d) None of the above

5. Consider the following pseudo code program fragment:

```
procedure swap(a, b)
begin
  t := a;
  a := b;
  b := t;
end
```

Suppose that the programming language adopts the *reference model* for variables (similar to Java) and thus the parameters are *passed by sharing*. What is the result after calling `swap(x, y)` on two variables `x` and `y`? (mark **one**) (4 points)

(a) the values of `x` and `y` are unchanged

(b) the values of `x` and `y` are interchanged

(c) the value of `x` is set to `y` and `y` is unchanged.

(d) the values of `x` and `y` are undefined

6. Consider the following pseudo code program fragment again:

```
procedure swap(a, b)
begin
  t := a;
  a := b;
  b := t;
end
```

Suppose that the programming language adopts *parameter passing by name* Let `i`=1 and all values of `a[]`=2. What is the result after calling `swap(i,a[i])`? (mark **one**) (4 points)

(a) `i`=1 and `a[1]`=2

(b) `i`=1 and `a[2]`=2

(c) `i`=2 and `a[1]`=1

(d) `i`=2 and `a[2]`=1

7. Tail recursion elimination is an optimization to replace recursion by iteration (a `goto` back into the code). Consider the following C function:

```
int collatz(int n, int k)
{
  if (n == 1)
    return k;                     <= [1]
  if (n % 2)
    return collatz(3*n+1, k+1);   <= [2]
  return collatz(n/2, k) + 1;     <= [3]
}
```

Which of the three points admit a tail-recursive call optimization (without rewriting the code)? (mark **one**) (4 points)

(a) `[1]`

(b) `[2]`

(c) `[3]`

(d) None of the above

8. What is *short-circuit evaluation* of operations in expressions? (mark **one**) (4 points)

(a) Operations in expressions are evaluated at compile time, when possible

(b) When the result of an operation can be determined from evaluating only one operand

(c) Redundant operations are eliminated by the compiler using common-subexpression elimination

(d) Logical operations are performed with bit-wise arithmetic on short integers

9. Fortran 77 uses only one parameter passing method. Which one? (mark **one**) (4 points)

   (a) call by value
   (b) call by reference
   (c) call by result
   (d) call by name

10. Dangling pointers (or references) is guaranteed to never occur when? (mark **one <u>or more</u>**) (4 points)

    (a) when explicit deallocation is forbidden (garbage collection is used)
    (b) when all objects and data are statically allocated (global, as in Fortran 77)
    (c) when all objects and data are allocated on the stack (in subroutine frames to be specific) as part of subroutine invocation
    (d) when the value model of variables is used by the programming language

11. Given the following axiomatic rules:

    Assignment:

$$\{P[V \rightarrow E]\}\ \ V := E\ \ \{P\}$$

    Sequencing:

$$\frac{\{P\}\ \ C_1\ \ \{Q\} \qquad \{Q\}\ \ C_2\ \ \{R\}}{\{P\}\ \ C_1; C_2\ \ \{R\}}$$

    If-then-else:

$$\frac{\{P \text{ and } B\}\ \ C_1\ \ \{Q\} \qquad \{P \text{ and not } B\}\ \ C_2\ \ \{Q\}}{\{P\}\ \ \textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2 \textbf{ end if}\ \ \{Q\}}$$

    where the formula for P:

$$P = (\text{not } B \text{ or } P_1) \text{ and } (B \text{ or } P_2)$$

    can be used when precondition P_1 of C_1 and precondition P_2 of C_2 are given.

    Complete the proof of correctness of the program fragment below: (8 points)

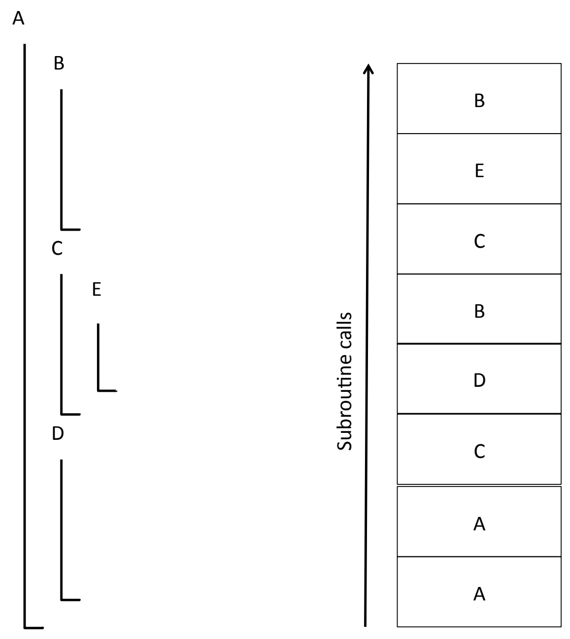
```
{true} ⇒
{...                          }
a := b + 1
{...                          }
if a > b then
    {...                          }
    x := a
    {x ≥ a and x ≥ b}
else
    {...                          }
    x := b
    {x ≥ a and x ≥ b}
end if
{x ≥ a and x ≥ b}
```

12. What is a *subroutine closure* and why are subroutine closures essential to functional programming languages in which functions are first-class objects/types? (7 points)

13. What is *heap fragmentation* and explain the two main causes. (8 points)

14. Explain why early Fortran 77 does not (and cannot) support recursion. (7 points)

15. Consider the following diagram depicting subroutine nesting levels and the stack frames after a sequence of subroutine calls:



Draw the static link from each frame to its static parent frame on the stack. (6 points)

16. Consider the following pseudo-code program:

```
a : integer /* global */
b : integer /* global */
procedure foo(P : procedure)
   a : integer
   a := 2
   P()
   b := a
procedure bar
   a := 1
begin /* main program */
   a := 0
   foo(bar)
   a := b
   write_integer(a)
end /* main program */
```

Assuming static scoping or dynamic scoping rules are used (with shallow or deep bindings). What value does the program print? (8 points)

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

17. What is the value printed by the following pseudo-code program for each of the four parameter passing modes shown in the table? (8 points)

```
a : integer /* global variable */
procedure count(x : integer)
    a := 1
    x := x + a
begin /* main program */
    a := 2
    count(a)
    write_integer(a)
end /* main program */
```

	By value	By reference	By value/result	By name
Output:				

18. Consider the following outline of a Queue class in C++ that includes exception handling.

```
class OutOfMemory {};
class DatabaseError {};
class DatabaseNotConnected : public DatabaseError {};
class DatabaseNotFound : public DatabaseError {};
class DatabaseFull : public DatabaseError {};

class Database
{ public:
    Database()
    { ...
      if (...) throw OutOfMemory();
    }
    void connect()
    { ...
      if (...) throw DatabaseNotConnected();
    }
    void store()
    { ...
      if (...) throw DatabaseFull();
    }
    void query(...)
    { ...
      if (...) throw DatabaseNotFound();
    }
};
void demo()
{ Database db();   // <----- (1)
  try {
    ...
    db.connect();  // <----- (2)
    ...
    db.store();    // <----- (3)
    ...
    db.query();    // <----- (4)
    ...
  } catch(DatabaseFull)
  { ... // <--- Handler (A)
  } catch(OutOfMemory)
  { ... // <--- Handler (B)
  } catch(DatabaseNotConnected)
  { ... // <--- Handler (C)
    throw DatabaseError();
  }
}
void main()
{ try {
    ... // <----------- (5)
    demo();
    ... // <----------- (6)
  } catch(DatabaseError)
  { ... // <--- Handler (D)
  } catch(OutOfMemory)
  { ... // <--- Handler (E)
  }
  ... // <------------- (7)
}
```

There are three types of exceptions associated with the Database class. The Database class methods can only raise the exceptions shown in the program outline above. Suppose exceptions are raised by the four statements `(1)` to `(4)`. For each exception raised at a point, indicate in the table below which handlers are executed (`A`–`E`) and where the execution continues at program point `(5)` to `(7)`? (8 points)

Exception caused by	Handlers executed and the continuation point
`(1)`	
`(2)`	
`(3)`	
`(4)`	