Name (PRINTED): ____________________________

University ID #: ____________________________

Circle your TA’s name: Guilherme Nir

Circle your discussion time: 12:00 1:00

---

CMSC 330 Final Exam Fall 2006

Warning: These solutions may still have some bugs in them...

Do not open this exam until you are told. Read these instructions:

1. This is a closed book exam. No notes or other aids are allowed.

2. You must turn in your exam immediately when time is called at the end.

3. This exam contains 14 pages, including this one. Make sure you have all the pages. Each question’s point value is next to its number. Write your name on the top of all pages before starting the exam.

4. In order to be eligible for as much partial credit as possible, show all of your work for each problem, and clearly indicate your answers. Credit cannot be given for illegible answers.

5. If you finish at least 15 minutes early, bring your exam to the front when you are finished; otherwise, wait until the end of the exam to turn it in. Please be as quiet as possible.

6. If you have a question, raise your hand. If you feel an exam question assumes something that is not written, write it down on your exam sheet. Barring some unforeseen error on the exam, however, you shouldn’t need to do this at all, so be careful when making assumptions.

7. If you need scratch paper during the exam, please raise your hand. Scratch paper must be turned in with your exam, with your name and ID number written on it. Scratch paper will not be graded.

8. Small syntax errors will be ignored in any code you have to write on this exam, as long as the concepts are correct.

9. The Campus Senate has adopted a policy asking students to include the following handwritten statement on each examination and assignment in every course: “I pledge on my honor that I have not given or received any unauthorized assistance on this examination.” Therefore, just before turning in your exam, you are requested to write this pledge in full and sign it below:


Good luck!
1. [20 pts.] **Short Answer.**

   a. [4 points] Consider the following program, written in a C-like language:

   ```c
   void f(integer x, integer y) {
      y = y+1;
      x = y;
   }
   void main() {
      integer x = 0;
      integer y = 10;
      f(x,y);
      printf("x=%d, y=%d\n", x, y);
   }
   
   Assume x is always call-by-reference. What does the program print if y is call-by-value? What is printed if y is call-by-reference?
   
   **Answer:** Under call-by-value, the program prints \(x=11\) \(y=10\). Under call-by-reference, the program prints \(x=11\) \(y=11\).

   b. [5 pts.] Consider the Ruby expression a+4, without knowing anything else about object a. What does the “+” refer to? What sequence of events happens when this expression is evaluated? **Note:** This is intended to be an easy question—we’re not looking for anything deep here.

   **Answer:** The “+” refers to the “+” method of a. When this expression is evaluated, the integer 4 is evaluated, and then it is passed as a parameter to the method.
c. [4 pts.] PL/1 is an imperative procedure-oriented language from the 1970s. The first PL/1 compiler did not implement a stack, and instead, each activation record was implemented as an allocate() and a free() of the storage from the heap as the procedure was entered and exited, respectively. Other data could also be allocated on the same heap by the programmer. In two sentences, describe two potential advantages or disadvantages of this implementation.

Answer:

- Very slow procedure activation and return due to need to call the allocator.
- Increased memory pressure on the heap, which could cause fragmentation if the program allocates heap objects as well.
- Activation records are likely to be different sizes, hence they will not be reused nearly as efficiently as the stack.
- Hard to detect infinite recursion, because the stack is mixed up with the heap.
- By the same token, the stack can grow as much as possible given the memory space.
- Support for higher-order functions may be easier, because local variables are all on the heap.

d. [4 pts.] A real-time program is one where various actions the program carries out must occur within a certain amount of time. For example, if a car’s braking system were controlled by software, then if the software didn’t do its job in time, you might crash. Why is an implementation that uses mark-and-sweep or stop-and-copy garbage collection not a good idea in a real-time program? Is reference counting a better solution? Explain your answer.

Answer: Garbage collection causes execution to halt, perhaps for a long period, while the garbage collector reclaims unused space. This may be a significant problem in a real-time program that needs to respond quickly to events.

Reference counting may be a better solution because its cost is spread out among all the operations in the program. However, reference counting can still have significant pauses, because removing the last reference to a large data structure could cause the whole structure to be collected.
e. [3 pts.] Fill in the following blanks with the correct language from the following list. Write one answer per blank. Languages may be used zero or more times in different answers.

ALGOL, BASIC, C, CLU, COBOL, FORTRAN, LISP, PL/1

i. ________________ was designed mostly for scientific computing

ii. ________________ was designed mostly for business computing

iii. ________________ used garbage collection to reclaim heap storage

iv. ________________ included exception handling

v. ________________ is considered the first high-level language

vi. ________________ was originally described using a BNF grammar

**Answer:**

i. FORTRAN or PL/1 or ALGOL

ii. COBOL or PL/1

iii. LISP (or CLU)

iv. CLU or PL/1

v. FORTRAN

vi. ALGOL
2. [10 pts.] **OCaml Types.** Write down the type the OCaml interpreter will assign to each of the following functions. If the OCaml interpreter would use polymorphic types like 'a, you can pick any mutually-consistent set of polymorphic type variables. (E.g., 'a->'a and 'b->'b are the same.)

a. let f x y = x::y
   Answer: 'a -> 'a list -> 'a list

b. let f (x, y) z = z x y
   Answer: 'a * 'b -> ('a -> 'b -> 'c) -> 'c

c. let f x y = x (x y)
   Answer: ('a -> 'a) -> 'a -> 'a

d. let rec f x y = match x with [] -> y | (_::xs) -> f xs (y+1)
   Answer: 'a list -> int -> int

e. let f a b c = b a c
   Answer: 'a -> ('a -> 'b -> 'c) -> 'b -> 'c
3. [20 pts.] Formal Languages.

   a. [4 pts.] Give a DFA for the set $R = \{a^m b^n c^p d^q \mid m, n, p, q \text{ are all } \geq 1\}$.

      Answer:

   b. [4 pts.] Give a context-free grammar for $R$ with $m = n$ and $p = q$.

      Answer:
      $S \rightarrow XY$
      $X \rightarrow aXb | ab$
      $Y \rightarrow cYd | cd$

   c. [4 pts.] Give a context-free grammar for $R$ with $m = p$

      Answer:
      $S \rightarrow XD$
      $X \rightarrow aXc | aBc$
      $B \rightarrow bB | b$
      $D \rightarrow dD | d$
d. [4 pts.] Let $L$ be the language recognized by the following DFA over the alphabet $\{0, 1\}$.

![DFA Diagram](image)

The complement of language $L$ is defined as $\neg L = \{s \mid s \not\in L\}$. Draw a DFA that accepts $\neg L$, and briefly describe a general procedure for complementing a DFA.

**Answer:** To complement a DFA, just switch the accepting and non-accepting states, being sure to account for the implicit dead state.

e. [2 pts.] Give the DFA with the fewest states that accepts $\neg L \cup L$, i.e., strings that are either in $\neg L$ or in $L$.

**Answer:**

f. [2 pts.] Does the same procedure you described in part 3d work for complementing an NFA? Justify your answer by either arguing it does or showing by counterexample it does not.

**Answer:**
4. [8 pts.] Lambda calculus

a. [4 pts.] Reduce the following term until no more reductions are possible: \((\lambda z.z) (\lambda x. x) (\lambda x. y)\)

Answer:

\[
\begin{align*}
(\lambda z.z) (\lambda x. x) (\lambda x. y) & \rightarrow (\lambda x. x) (\lambda x. y) \\
& \rightarrow (\lambda x. y) (\lambda x. y) \\
& \rightarrow (\lambda x. y) \, y \\
& \rightarrow y \, y
\end{align*}
\]

b. [4 pts.] Suppose we define the following combinators:

- \textit{true} = \lambda x. \lambda y. x
- \textit{false} = \lambda x. \lambda y. y
- \textit{not} = \lambda x. x \, \textit{false} \, \textit{true}

Show that \(\textit{not} \, (\textit{not} \, x) \rightarrow \cdots \rightarrow x\) where \(x\) is either \textit{true} or \textit{false}. Hint: To save writing, first reduce \(\textit{not} \, (\textit{not} \, x)\) as much as possible, and then set \(x\) to either \textit{true} or \textit{false}.

Answer:

\[
\begin{align*}
\textit{not} \, (\textit{not} \, x) &= \textit{not} \, ((\lambda x. x \, \textit{false} \, \textit{true}) \, x) \\
& \rightarrow \textit{not} \, (x \, \textit{false} \, \textit{true}) \\
& = (\lambda x. x \, \textit{false} \, \textit{true}) \, (x \, \textit{false} \, \textit{true}) \\
& \rightarrow x \, \textit{false} \, \textit{true} \, \textit{false} \, \textit{true}
\end{align*}
\]

So then

\[
\begin{align*}
\textit{not} \, (\textit{not} \, \textit{true}) &= \textit{true} \, \textit{false} \, \textit{true} \, \textit{false} \, \textit{true} \\
& = (\lambda x. \lambda y. x) \, \textit{false} \, \textit{true} \, \textit{false} \, \textit{true} \\
& \rightarrow \textit{false} \, \textit{false} \, \textit{true} \\
& = (\lambda x. \lambda y. y) \, \textit{false} \, \textit{true} \\
& \rightarrow \textit{true}
\end{align*}
\]

and

\[
\begin{align*}
\textit{not} \, (\textit{not} \, \textit{false}) &= \textit{false} \, \textit{false} \, \textit{true} \, \textit{false} \, \textit{true} \\
& = (\lambda x. \lambda y. y) \, \textit{false} \, \textit{true} \, \textit{false} \, \textit{true} \\
& \rightarrow \textit{true} \, \textit{false} \, \textit{true} \\
& = (\lambda x. \lambda y. x) \, \textit{false} \, \textit{true} \\
& \rightarrow \textit{false}
\end{align*}
\]
5. [8 pts.] **Operational Semantics.** Consider the following language, which is lambda calculus to which we’ve added booleans as primitives:

\[ e ::= x \mid e e \mid \lambda x.e \mid true \mid false \mid not e \]

We would like to give an operational semantics for this language that defines reductions of the form \( A; e \to v \), meaning in environment \( A \) (which binds variables to values), expression \( e \) evaluates to value \( v \). The values \( v \) are given by

\[ v ::= (A, \lambda x.e) \mid true \mid false \]

Here \( true \) and \( false \) are the appropriate boolean values, and \((A, \lambda x.e)\) is a closure with environment \( A \) and function \( \lambda x.e \).

Here is the start of an operational semantics:

\[
\begin{align*}
A; x & \to A(x) \\
A; \lambda x.e & \to (A, \lambda x.e) \\
A; true & \to true \\
A; e_1 & \to (A', \lambda x.e) \\
A; e_2 & \to v \\
A', x : v; e & \to v' \\
A; e_1 e_2 & \to v'
\end{align*}
\]

a. [4 pts.] What is the parameter passing mechanism represented by these semantics? In order to receive credit for your answer, justify it by referring to the rules.

**Answer:** This is call-by-value, because the expression \( e_2 \) is evaluated to a value \( v \) before the function is invoked.

b. [4 pts.] Write down operational semantics rules for \( false \) and \( not \), following the pattern above. (Don’t worry about the fonts.)

**Answer:**

\[
\begin{align*}
A; false & \to false \\
A; e & \to true \\
A; not e & \to false \\
A; e & \to false \\
A; not e & \to true
\end{align*}
\]
6. [8 pts.] Threading in Java. In this question, you will write Java code for a synchronization construct called a barrier. A barrier is created with a certain number of parties n. When a thread calls `await()`, it enters the barrier and blocks until a total of n parties have entered the barrier. When the nth party enters the barrier, all the threads waiting at the barrier wake up and unblock, and the nth thread continues without blocking. A barrier may also be reset so that it starts fresh in counting up to n.

Write a class `Barrier` that implements this behavior. Your class should have a constructor `Barrier(int n)` to create a barrier with n parties, a method `void await()`, and a method `void reset()`.

**Answer:**

```java
public class Barrier {
    private int n, cur;

    public void Barrier(int n) {
        this.n = n; cur = 0
    }

    public synchronized void await() {
        cur++;
        while (cur < n)
            wait();
        notifyAll();
    }

    public synchronized void reset() {
        cur = 0;
    }
}
```
7. [16 pts.] **Boolean expression.** A *boolean expression* is a boolean function built from operators like *and* and *not*. Examples are \((x_1 \land x_2) \land x_3\) or \(\neg x_1 \land x_3 \land x_4\). (Recall that \(\land\) means *and.*) Here is an OCaml data type for abstract syntax trees for boolean expressions:

```ocaml
type expr =
    | False
    | True
    | Var of string
    | And of expr * expr
    | Not of expr
    | Forall of string * expr
```

Here `False`, `True`, `And`, and `Not` represent the usual boolean values and operators. In this AST, boolean expressions may include variables, given by AST node `Var`. We will use an associative list of the following type to track the values of variables:

```ocaml
type assn = (string * bool) list
```

Here if an `assn` contains the pair \((x, b)\), then that assignment gives the variable `x` the value `b`.

The last expression represents \(\forall\), where the expression \(\forall x. e\) is true if \(e\) is true under all possible assignments to \(x\), i.e., if \(e\) is true for \(x = true\) and \(x = false\).

a. [4 pts.] Write a function `free : expr -> string list` that returns a list of free variables in the expression. Just as in a programming language, the free variables of a boolean expression are those not bound by `Forall`. For example, `free (Var "x") = ["x"]` and `free (Forall ("x", And (Var "x", Var "y"))) = ["y"]`.

**Answer:**

```ocaml
let rec free = function
    | False -> []
    | True -> []
    | Var v -> [v]
    | And (e1, e2) -> (free e1) @ (free e2)
    | Not e -> free e
    | Forall (x, e) -> List.filter (fun y -> not (y = x)) (free e)
```
b. [6 pts.] Write a function \texttt{eval} : \texttt{assn} \to \texttt{expr} \to \texttt{bool} that evaluates the boolean expression on the given variable assignment. For example, given the OCaml value \texttt{And(And(Var "x1", Var "x2"), Var "x3")} and the assignment \texttt{["x1", true]; ("x2", false); ("x3", true)], the eval function should return false. You may assume that any variable encountered during evaluation is either given a value by the assignment or bound by an enclosing \texttt{Forall}.

\textbf{Answer:}

\begin{verbatim}
let rec eval env = function
  False -> false
| True -> true
| Var v -> List.assoc env x
| And (e1, e2) -> (eval env e1) && (eval env e2)
| Not e -> not (eval env e)
| Forall (x, e) -> (eval (x, true)::env e1) && (eval (x, false)::env e1)
\end{verbatim}

c. [4 pts.] Write a function \texttt{taut} : \texttt{expr} \to \texttt{bool} that returns true if and only if the expression is a tautology (i.e., is always true under any assignment to the free variables). You may use functions you wrote for parts a and b.

\textbf{Answer:}

\begin{verbatim}
let taut e =
  let vars = free e in
  let taut' env = function
      [] -> eval env e
    | (x::xs) ->
        let t = taut' ((x, true)::env) xs in
        let f = taut' ((x, false)::env) xs in
        t && f
  in
  taut' [] vars
\end{verbatim}

d. [2 pts.] Consider the following grammar for a subset of boolean expressions, where we've chosen \texttt{prefix} notation, so that the operator appears first, followed by its arguments.

\[ expr \rightarrow \texttt{true} | \texttt{false} | \texttt{and} \ expr \ expr | \texttt{not} \ expr \]

Is this grammar ambiguous? Justify your answer.

\textbf{Answer:} No. One way to see this is to observe that in a predictive parser, we always know which production to choose based on the lookahead.
8. [10 pts.] Language Comparison. Interactive fiction is a computer gaming genre based on text input and output. The game presents the player with text descriptions of the world, and the player moves around the world and interacts with it by typing in a limited vocabulary of noun-verb commands. For example, here is a brief transcript of playing the game Zork, which was developed in the late 1970's. The text entered by the player is preceded by the > prompt, and everything else is displayed by the game.

```
Welcome to Zork (originally Dungeon). This version created 11-MAR-91 (PHP mod 03-AUG-05)
...
You are in an open field west of a big white house with a boarded front door.
There is a small mailbox here.

> open mailbox
Opening the mailbox reveals:
A leaflet.

> take leaflet
Taken.

> read leaflet
Welcome to Zork (originally Dungeon)!
...
```

Suppose you were asked to write a program that implements an interactive fiction game. Out of Ruby, OCaml, or Java, which language(s) would you choose for your implementation, and why? You will receive two points for each advantage or disadvantage you list for using the language for this application. We will subtract points if you say something that is wrong. Thus we recommend keeping your answer to roughly one paragraph, or two at most. You may continue on the next page if you are feeling expansive.
You may continue your answer here