University of Virginia **Out: 28 November 2011**
cs1120: Introduction of Computing **Due: 11:01 am, Wednesday, 30 November**
Explorations in Language, Logic, and Machines

Exam 2

Name: _____ **UVa Email ID:** __ __ __ __ __ __

Directions

Due: 11:01am, Wednesday, 30 November 2011. No late exams will be accepted without prior arrangement or an extraordinarily good story. Since you have 10 days for this exam, the threshold for a good story is significantly higher than it was for Exam 1.

Work alone. Between receiving this exam and turning it in on 30 November, **you may not discuss these problems or anything directly related to this exam with anyone other than the course staff.** You may (and probably should), however, continue to work on Problem Set 8, but must avoid discussing anything directly related to this exam with your PS8 teammates.

Open resources. You may use any books you want, lecture notes, slides, your notes, and problem sets, including any materials posted on or linked from the course website. Unlike Exam 1, on this exam you may also use any computer programs you want, including DrRacket, Python, and code from the problem sets including PS7. You may also use external non-human sources including books and web sites. If you use anything other than the course book, slides, notes, and standard interpreters, you must cite what you used clearly in your answer. **You may not obtain any help from other humans** other than the course staff (who will only answer clarification questions).

Answer well. Answer all of the graded questions and the optional, ungraded questions on the last page. A "full credit" answer for each question is worth 10 points (but it is possible to get more than 10 points for especially elegant and insightful answers). Your answers must be clear enough for us to read and understand. If you do not use our provided print-out, your print-out should use only the front sides of pages and should be printed well enough to be easily read. You should not need more space than is provided to write good answers, but if you need more you may use the backs or attach extra sheets. If you do, make sure the answers are clearly marked.

The questions are not necessarily in order of increasing difficulty. There is no time limit on this exam, but it should not take a well-prepared student more than two hours to complete. It may take you longer, though, so please do not delay starting the exam. You should definitely not allow this exam to interfere with your Thanksgiving holiday.

Full credit depends on the clarity and elegance of your answer, not just correctness. Your answers should be as short and simple as possible, but not simpler. Your answers will be judged for correctness, clarity and elegance, but you will not lose points for trivial errors (such as missing a closing parenthesis).

Pledge: _____

Sign here to indicate that you **read, agreed to, and followed** all of the directions here in addition to the Course Pledge.

First Question

1. According to Dean Kamen's speech at the Rice Hall dedication (quoting Walt Havenstein), what is "the only sport I know where everyone can turn pro"?

   (Note: if you had the misfortune of not being able to attend Dean's talk, you should still be able to answer this question by searching for the quote using DuckDuckGo, Google, or Bing.)

Running Time Analysis

2. Describe the worst-case asymptotic running time of the all-positive? Scheme procedure defined below (from the Exam 1 comments). You may assume that all elements of **p** have values below some bound *k*, so the running time of **>** is constant. Remember to clearly define all variables you use in your answer.

```
(define (all-positive? p)
  (if (null? p)
     true ; reached end without finding non-positive, so result is true
     (if (> (car lst) 0)
        (all-positive? (cdr lst)) ; keep looking
        false))) ; found one non-positive
```

3. Describe the worst-case asymptotic running time of the allPositive Python procedure defined below (that behaves similarly to the Scheme procedure above). You may assume that all elements of **p** have values below some bound *k*, so the running time of **<=** is constant. Remember to clearly define all variables you use in your answer.

```
def allPositive(p):
  for x in p:
     if x <= 0: return False
  return True
```

4. Describe the worst-case asymptotic running time of the **goldStarsSquare(g)** Python procedure defined below. The input, **g**, is a natural number. You should state clearly all assumptions you make. Remember to clearly define all variables you use in your answer. The best answers will be in terms of the *size* of the input, not the value of **g**, but you will receive nearly full credit for a correct answer in terms of the value of **g**.

```
def rowGoldStars(g):
    for i in range(0, g):
        print "*",
    print # (end the row by printing a new line)

def goldStarsSquare(g):
    for i in range(0, g):
        rowGoldStars(g)
```

Mutation

5. Define a procedure that takes as input a mutable list of numbers, and modifies the list so that each element is replaced with its negation. You may use *either* Scheme or Python to define your procedure (your choice). For example, in Scheme you would define the **mlist-negate!** procedure that behaves like this:

```
> (define p (mlist 1 -2 3 0 -17))
> (mlist-negate! p)
> p
{-1 2 -3 0 17}
```

   In Python you would define the **listNegate** procedure that behaves like this:

```
>>> p = [1, -2, 3, 0, -17]]
>>> listNegate(p)
>>> p
[-1, 2, -3, 0, 17]
```

6. Define a Scheme procedure **make-cumulative!** that takes as input a mutable list, and modifies the list so that each element is the cumulative total of all elements up to and including itself. For example,

```
> (define p (mlist 1 2 3 4 5))
> (make-cumulative! p)
> p
{1 3 6 10 15}
```

Hint: you probably should start by defining a helper procedure. (For full credit for this question, you must use Scheme. But, a correct answer using Python is worth most of the credit.)

Bonus: what is the asymptotic running time of your **make-cumulative!** procedure. You should **not** assume the running time of the **+** procedure is constant; instead, assume that its running time is linear in the size (number of bits) of its inputs. *(Use the back of this page to answer the bonus question, but circle Bonus to indicate that you have something we should look at for this.)*

Interpreters

Suppose we want to add a new special form to Charme similar to the **and** special form in Scheme. The grammar for the **and** special form is:

> *Expression* → *AndExpression*
> *AndExpression* → **(and** *ExpressionList***)**
> *ExpressionList* → ε
> *ExpressionList* → *Expression ExpressionList*

The evaluation rule is:

> To evaluate the **and** expression, **(and** E_0 E_1 **…** E_n**)**, evaluate each sub-expression in order until one evaluates to a **false** value. If any sub-expression evaluates to a false value, the value of the **and** expression is false, and none of the following sub-expression is evaluated. If none of the sub-expressions evaluate to a false value, the value of the and expression is **true**. (Note that this means **(and)** should evaluate to **true**.)

7. Explain why **and** needs to be a special form (that is, it requires modifying the Charme evaluator and could not be implemented as a primitive procedure).

8. Define an **evalAnd(expr)** procedure that implements the evaluation rule for the and special form. You may assume the value passed in as **expr** is parsed Charme expression and that corresponds to a syntactically valid and expression.

Computability

The cs1120 staff is getting tired of having to grade so many programming questions[1], so wants to build a procedure that automates grading by checking if a solution submitted implements the same function as our reference solution.

9. Is the *CORRECT-ANSWER* problem defined below computable? For full credit, your answer must include a convincing proof supporting your answer.

   Input: Strings that define two Python procedures, *r* (the reference procedure) and *s* (the submitted procedure).

   Output: True if the procedure defined by *s* correctly implements the same function as the procedure defined by *r*. We say *s* correctly implements the same function as *r* if for every input for which *r* produces a value, s produces the same value.

[1] The obvious solution of just asking fewer questions had not occurred to us until we read your PS7 responses, but not everyone finds this solution satisfying.

10. Is the *PROBABLY-CORRECT-ANSWER* problem defined below computable? For full credit, your answer must include a convincing proof supporting your answer.

    Input: Strings that define two Python procedures, *r* (the reference procedure) and *s* (the submitted procedure), a set *V* of test inputs, and a number *t* for the maximum number of steps.

    Output: True if the procedure defined by *s* probably implements the same function as the procedure defined by *r*. We say *s* probably implements the same function as *r* if for each element *v* of *V*, if *r*(*v*) produces a value within *t* steps, *s*(*v*) produces the same value within *t* or fewer steps.

Problem	Score	Notes
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
Total		