Banner number: Name:

Final Exam

CSCI 3136: Principles of Programming Languages

April 14, 2018

Binding		Control flow		Control abstractions		Data abstractions		
Q1.1		Q2.1		Q3.1		Q4.1		$\sum$
Q1.2		Q2.2		Q3.2		Q4.2		
Q1.3		Q2.3		Q3.3		Q4.3		
$\sum$		$\sum$		$\sum$		$\sum$		

Instructions:

- Provide your answer in the box after each question. If you absolutely need extra space, use the backs of the pages; but try to avoid it. Keep your answers short and to the point.
- You are not allowed to use a cheat sheet.
- Make sure your answers are clear and legible. If I can't decipher an answer or follow your train of thought with reasonable effort, you'll receive 0 marks for your answer.
- Read every question carefully before answering.
- Do not forget to write your banner number and name on the top of this page.
- This exam has 11 pages, including this title page. Notify me immediately if your copy has fewer than 11 pages.

1 Binding

Question 1.1: Scopes and referencing environments — 6 marks

Define the following terms:

Scope of a binding

Scope

Referencing environment

Question 1.2: Static scoping vs lexical scoping — 4 marks

Consider the following Scheme function:

```
(define (f)
  (let ((z 1))
    (define (g) z)
    (define (h) (let ((z 2)) (g)))
    (h)))
```

What value does the function invocation `(f)` return?

Using static scoping

Using dynamic scoping

Question 1.3: Aliasing **5 marks**

Consider the following C function for computing the length of a vector:

```
float veclen(float *x, float *y) {
  *x *= *x;
  *y *= *y;
  *x += *y;
  return sqrt(*x);
}
```

If, before calling this function, x is the value stored in the memory location referenced by x and y is the value stored in the memory location referenced by y, does this function always return $\sqrt{x^2 + y^2}$ as intended? Explain.

2 Control Flow

Question 2.1: Control flow primitives **9 marks**

List and briefly describe three control flow primitives. Try to separate the control flow principle from its syntax in a particular language.

1.

2.

3.

Question 2.2: Switch statements — 5 marks

Programming languages that have switch-statements usually have them because they are more efficient than multi-way if-statements. Describe an implementation of switch-statements that is more efficient than a multi-way if-statement.

Question 2.3: Applicative-order and normal-order evaluation — 4 marks

Consider the following piece of code:

```
bool print_or_stop(int x) {
  if (x < 10) {
    printf("%d ", x);
    return true;
  }
  else {
    return false;
  }
}

int f() {
  static int x = 0;
  return x++;
}

void main() {
  while (print_or_stop(f()));
}
```

What does this program output?

Using applicative-order evaluation of function arguments

Using normal-order evaluation of function arguments

3 Control Abstractions

Question 3.1: Parameter passing modes — 9 marks

Explain the difference between call by value, call by reference, and call by sharing, both in terms of the mechanics of these parameter passing modes and in terms of the degree to which the called function can alter the contents of the caller's variables.

Question 3.2: Exception handling — 5 marks

There are numerous ways in which the runtime system of a language supporting exceptions can implement the handling of exceptions. Describe one method that ensures that exception handling incurs no runtime overhead at all as long as no exceptions are thrown while still handling exceptions reasonably efficiently when they are thrown.

Question 3.3: Closures — 3 marks

Consider the following piece of Scheme code:

```
(define count-all 0)

(define (new-counter)
  (let ((counter 0))
    (lambda () (set! counter (+ counter 1))
               (set! count-all (+ count-all 1))
               (list counter count-all))))

(define (run)
  (let ((a (new-counter))
        (b (new-counter)))
    (list (a) (a) (b) (a) (b))))
```

Note: `(list ...)` creates a list containing its arguments.
`(set! var val)` stores the value `val` in the variable `var`.

What is the return value of the function call `(run)`?

4 Data Abstractions

Question 4.1: Type systems — 5 marks

Explain the difference between a statically typed and a dynamically typed language. Assume that both are strongly typed.

Question 4.2: Dynamic local arrays **5 marks**

Efficient access to local variables at runtime is based on knowing, at compile time (!), the address of each variable relative to the frame pointer. Explain how you can ensure this while also supporting local arrays, allocated on the stack, whose size is determined only at runtime. How do you support testing for out-of-bounds accesses on such arrays?

Question 4.3: Tombstones **5 marks**

Tombstones are used to detect a common programming error in languages with manual memory management (no garbage collection). Which type of error do they detect?

Explain how tombstones work.