Final

Name: Cleverdon,Michael

General Instructions

You have 120 minutes to complete this exam. You may put your solutions on the test, or create additional files. You may refer to your notes, coursework, textbook, and the Internet. You may neither observe nor consult your classmates, or anyone else. Read the problems carefully. They are not in any particular order. If you have a question, ask during the test:

Zoom: 963 4792 6575
Email: `buff@cs.boisestate.edu`

Please submit your exam, by 4:45 PM:

```
onyx$ submit jbuffenb cs354 final
```

Suggestion: Do not translate and/or run code. It wastes time. A solution does not need to be perfect. A good solution demonstrates that you understand the concepts.

I have provided the grammar from Interpreter Assignment #2, on the last page of this exam. It will be useful, for some of the problems.

Problem	*Description*	*Points Awarded*	*Points Possible*
1	Grammars: Parsing		5
2	Translation: Stack Frames		10
3	Translation: Control Flow		20
4	Languages: Declarative and Functional		10
5	Languages: Memory		10
	Total		55

Name: Cleverdon,Michael

Problem #1

5

According to the grammar for Interpreter Assignment #2, the following program can be parsed in two ways, as indicated by the indentation:

```
if ··· then
    if ··· then
        ...
else
    ...
```

(left)

```
if ··· then
    if ··· then
        ...
    else
        ...
```

(right)

Show a parse tree for the program on the right, containing "···" for subtrees.

Prog

block

stmt

'if' ... 'then' stmt

'if' ... 'then' ... 'else' ...

Name: Cleverdon,Michael

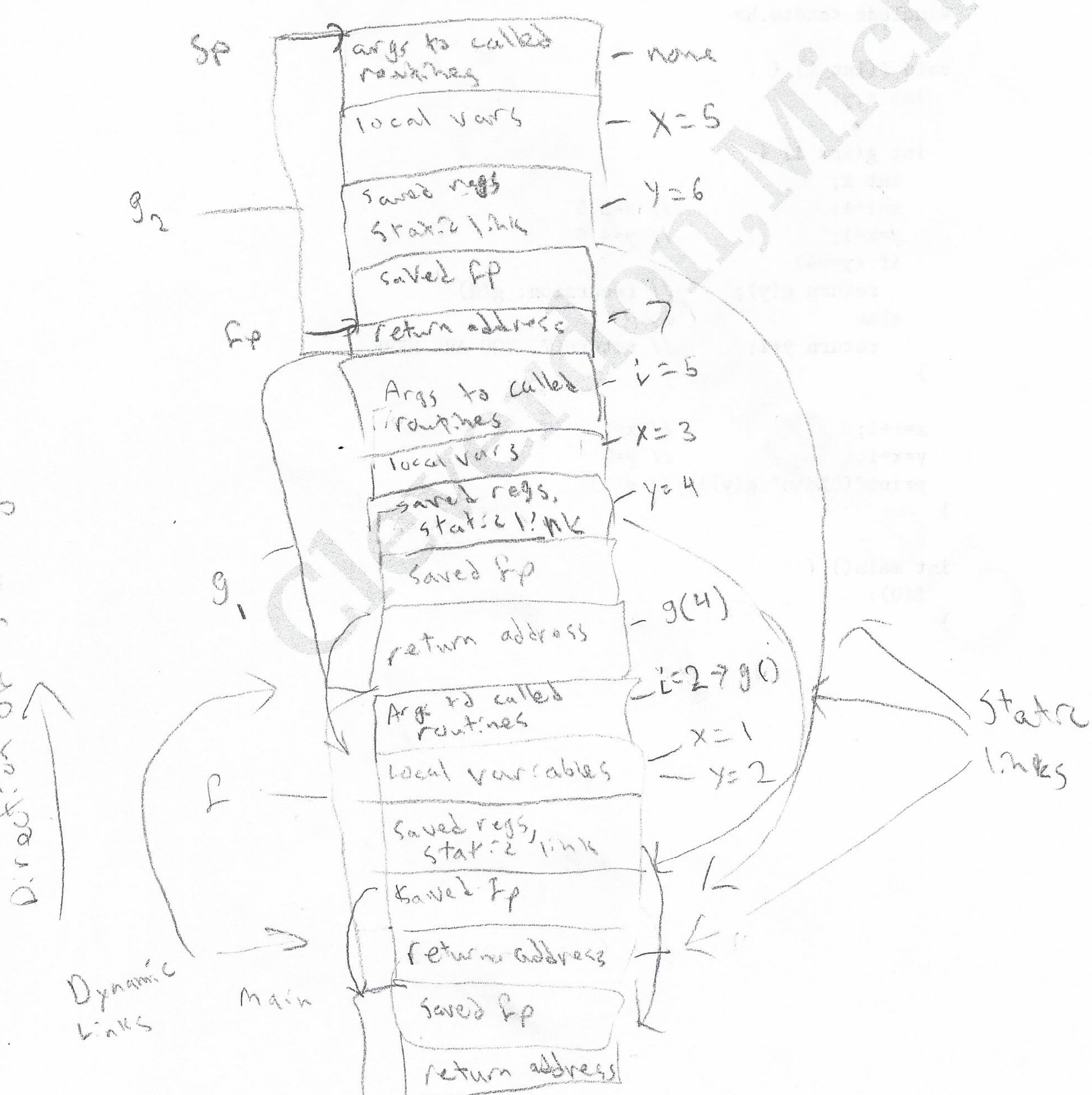
Problem #2

10

Suppose the following C program is just about to execute the indicated `return` statement. Draw a picture of a reasonable upward-growing *display-based* run-time stack. Show all of the stack frames, including: links, return addresses and values, formal parameters and values, local variables and values, and the stack and frame pointer. You may need to make some assumptions. You may use the next page.

```
#include <stdio.h>

void f(int i) {
  int x,y;

  int g(int i) {
    int x;
    x=i+1;              // x=3,5
    y=x+1;              // y=4,6
    if (y==4)
      return g(y);      // recursion: g(4)
    else
      return y+1;       // return 7, YOU ARE HERE
  }

  x=i+1;                // x=1
  y=x+1;                // y=2
  printf("%d\n",g(y)); // g(2)
}

int main() {
  f(0);
}
```

Name: Cleverdon,Michael



Name: Cleverdon,Michael

Problem #3
$\overline{20}$

The language of Interpreter Assignment #2 provides a `while` statement. Explain *all* of the steps necessary to add `break` and `continue` statements, as found in many languages (e.g., Java).

Be concise and complete. Explain each change and its location. You don't have to show code, but you can.

break and continue statements both can occur anywhere inside of a while loop, so we would need to create a new rule called loopStmt that can be a

```
loopBlock: loopStmt ';' loopBlock
         | loopStmt

loopStmt : stmt
         | break stmt
         | continue stmt
         | 'begin' loopBlock 'end'
```

Here I'm also including a loop block that uses loopStmt's instead of normal stmt's in order to preserve normal functionality of all the other keywords and making break/continue exclusive to loops (like they are in java). These grammar changes will be in the scanner to accept new keywords

The parser will have to have some parseContinue/parseBreak methods in their parseStmt block to look for new keywords and make new parseloopBlock and parseloopStmt sections that implement the grammar

Lastly, you would have to create new node classes to implement the break and continue functionality in their eval()' methods.
- loopBlock and loopStmt would inherit their functionality from block and stmt in the code

Name: Cleverdon,Michael

Problem #4
10

The following Prolog rules define a predicate named `foo`, which performs a simple list manipulation:

```
foo([], []).
foo([H|T], [H|[H|U]]) :- foo(T, U).
```

Explain, from a user's perspective, what `foo` does. Then, show the definition of a Scheme function that does the same thing.

foo duplicates records. Take the list [1, 4, 7, 10, 5]. foo will have 1 be H and [4, 7, 10, 5] as T. The output will have [1, 1, foo(T)] as it's output from the first run,

```
(define (foo lis)
  (if (pair? lis)
    return (cons (cons (car lis) (car lis)) (foo (cdr lis)))
    lis)) ; not pair if null
```

Name: Cleverdon,Michael

Problem #5
10

While discussing Scheme, in lecture, we saw this Java class:

```
public class Pair {

    private Object car, cdr;

    private Pair(Object car, Object cdr)
        { this.car=car; this.cdr=cdr; }

    public static Pair cons(Object car, Object cdr)
        { return new Pair(car,cdr); }

    public Object car() { return car; }
    public Object cdr() { return cdr; }

}
```

Using this class, show Java code similar to this Scheme program:

```
(define (len v)
  (if (pair? v)
      (+ 1 (len (cdr v)))
      0))

(define v '(a b c))

(display (len v))
(newline)
```

You may use the next page.

```
Public int len (Object v){
    if( v.getClass() == Pair){
        return 1+ len(v.cdr());
    } else {
        return 0;
    }
}
Pair v = cons(cons(a,b),c);
System.Out.Println(len(v))
```

— all of this in main

Name: Cleverdon,Michael