Final

Name:

General Instructions

> *This exam was intended to be taken in a regular, non-pandemic classroom, where students did not have access to computers. So much for that!*
>
> *You are welcome to use your computers, but I do not expect you to compile and execute any code you write. You can, if you want, but your solutions do not need to be that polished. A good solution demonstrates that you understand the basic concepts.*

You have 120 minutes to complete this test. You may write on the test, or attach additional paper. Put your name on each sheet of paper. You may refer to your notes, coursework, and textbook, but no networked equipment. You may neither observe nor consult your neighbors. Read the problems carefully. If you have a question, ask during the test.

Problem	*Description*	*Points Awarded*	*Points Possible*
1	Grammars: Parsing		10
2	Languages: Declarative and Functional		20
3	Code Generation: Stack Frames		20
4	Grammars: Design		15
	Total		65

Name:

Problem #1

10

Consider this abridged grammar, similar to that of Interpreter Assignment #2:

```
prog     : block
block    : stmt ';' block
         | stmt
stmt     : assn
         | 'wr' expr
         | 'if' boolexpr 'then' stmt
         | 'if' boolexpr 'then' stmt 'else' stmt
expr     : term addop expr
         | term
term     : fact mulop term
         | fact
fact     : id
         | num
         | '(' expr ')'
         | '-' fact
boolexpr : expr relop expr
relop    : '<'
         | '<='
         | '>'
         | '>='
         | '<>'
         | '=='
```

Show a parse tree for the following:

```
if x<y then
    if a<b then
        wr x
else
    wr y
```

Explain any assumptions you make. You may use the next page.

Name:

Name:

Problem #2

20

The following Prolog program defines a predicate named `foo`, which performs a common list manipulation:

```
1 foo([],          [],      []).
2 foo([H1|L],      [H1|T1], [])      :- foo(L, T1, []).
3 foo([H2|L],      [],      [H2|T2]) :- foo(L, [], T2).
4 foo([H1|[H2|L]], [H1|T1], [H2|T2]) :- foo(L, T1, T2).
```

Concisely and precisely, in English, from a user's perspective, explain what `foo` does. Then, rewrite the program in Scheme, renaming `foo` to something mnemonic.

Name:

Problem #3

20

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.

```
1  #include <stdio.h>
2
3  void f(int i) {
4    int x,y;
5
6    int g(int i) {
7      int x;
8      x=i+1;              // x=3,5
9      y=x+1;              // y=4,6
10     if (y==4)
11       return g(y);      // recursion: g(4)
12     else
13       return y+1;       // return 7, YOU ARE HERE
14   }
15
16   x=i+1;                // x=1
17   y=x+1;                // y=2
18   printf("%d\n",g(y)); // g(2)
19 }
20
21 int main() {
22   f(0);
23 }
24
```

Name:

Name:

Problem #4

15

Consider this abridged grammar, similar to that of Interpreter Assignment #2:

```
stmt: ···
      | 'if' boolexpr 'then' stmt
      | 'if' boolexpr 'then' stmt 'else' stmt
      ⋮
```

Some languages (e.g., Ada, BASIC, Bash, FORTRAN, PHP, Perl, and Python) have an optional `else if`, `elseif`, `elsif`, or `elif` clause for their `if` statements. Different languages use different keywords, but the semantics are all the same. For example:

```
if x==a then
   x=d
elsif x==b then
   x=e
elsif x==c then
   x=f
else
   x=g
```

Pick one keyword, and extend the grammar fragment to allow for this feature.