Directions; PLEASE READ

This exam comprises a mixture of multiple-choice, short-answer and problem-solving questions. Values are indicated for each; they total 58 points. Question 20 is worth 6 extra credit points. It isn’t part of the 58, and it won’t factor into your exam score, but it may help to raise your letter grade at the end of the semester.

This is a closed-book exam: you must put away all books, cellphones, and notes. Please confine your answers to the space provided. For multiple choice questions, unless otherwise instructed, darken the circle next to the single best answer. Be sure to read all candidate answers before choosing. No partial credit will be given on the multiple-choice questions.

In the interest of fairness, I will decline to answer questions during the exam. If you are unsure what a question is asking, make a reasonable assumption and state it as part of your answer.

You must complete the exam in class. I will collect any remaining exams promptly at 1:45 pm. Good luck!

1. (3 points) Put your name on every page (so if I lose a staple I won’t lose your answers).

Multiple Choice (2 points each)

2. What is the defining difference between a compiler and a preprocessor?
   - a. A compiler performs code improvement (optimization); a preprocessor does not.
   - b. A compiler performs full syntactic and semantic analysis; a preprocessor does not.
   - c. A compiler produces machine language; a preprocessor produces source code.
   - d. none of the above

3. Which of the following is a checked dynamic semantic error in C (one the language implementation is required to catch at run time)?
   - a. use of an uninitialized variable
   - b. access off the end of an array
   - c. wrong number of parameters passed to a function
   - d. divide by zero

4. What is a sentential form?
   - a. One line of a derivation.
   - b. An arbitrary string of grammar symbols.
   - c. The content of the LL(1) parse stack.
   - d. The string of symbols between the current point in the parse and the root of the parse tree.
5. Why, in the general case, must local variables in OCaml be allocated in the heap instead of on the stack?

□ a. Because we don’t know their sizes at compile time.
× b. Because local variables in OCaml can have unlimited extent.
□ c. Because OCaml subroutines don’t follow strict LIFO calling conventions.
□ d. Because the OCaml garbage collector is unable to manipulate addresses in the stack.

6. Language implementations based on interpretation are almost always slower than implementations based on compilation. Why, then, do people still use interpreters?

□ a. Interpreters tend to produce more helpful error messages.
□ b. Interpreter-based implementations tend to require less memory than compiler-based implementations.
□ c. Interpreters accommodate late binding.
× d. All of the above.

7. What is the significance of L-attributed attribute grammars?

□ a. Their attribute flow is strictly bottom-up.
□ b. They are the most general class of attribute grammars that can be evaluated in linear time.
× c. They can be evaluated in the course of an LL parse.
□ d. They work on syntax trees rather than parse trees.

8. The grammars used in many production compilers contain error productions, which allow the parser to accept certain input programs that are, according to the manual, syntactically invalid. What purpose do these error productions serve?

□ a. They improve compilation speed by catching common errors more quickly than a general-purpose error recovery scheme would.
□ b. They support language features that can be parsed bottom-up, but not top-down.
× c. They provide more helpful error messages than a general-purpose error recovery scheme would.
□ d. none of the above

9. The static link of a subroutine is

□ a. the address to which the subroutine should return
× b. the frame pointer of the lexically surrounding subroutine
□ c. the frame pointer of the calling subroutine
□ d. the base address of global variables
Recall, for the following three questions, that OCaml, like most modern languages, uses static scope and deep binding.

10. What is the value of the following?

```ocaml
let x = 2 in
let f y = x + y in
let x = 3 in
f 4;;
```

- a. 5
- b. 6
- c. 7
- d. none of the above: the program is incorrect

11. How about this one?

```ocaml
let x = 2 in
let f y = x + y in
let x = 3 in
f x;;
```

- a. 4
- b. 5
- c. 6
- d. none of the above: the program is incorrect

12. And this one?

```ocaml
let rec f p n =
  let g () = n in
  if n = 0 then p ()
  else if n > 2 then f g (n - 1)
  else f p (n - 1) in
f (fun () -> 0) 3;;
```

- a. 3
- b. 2
- c. 1
- d. 0
Short Answer (3 points each)

13. Recall that a pass of compilation is a phase or set of phases that completes before the next phase begins (rather than being interleaved [pipelined] with subsequent phases). Why might a phase (or set of phases) be implemented as a separate pass?

**Answer:** Historically: to minimize the memory footprint of the compiler (space for the code of one pass can be reused by the next). Today: to facilitate the construction of compiler *families*, in which front ends for different languages can be coupled, “mix-and-match,” with back ends for different machines.

14. Absent a specific declaration to the contrary, Python adopts the convention that a variable is local to a scope if and only if it is written in that scope. Explain the rationale for this convention: why is this a good way to distinguish between local and global variables?

**Answer:** In general, Python programmers do not declare their variables, so some sort of convention is necessary. A local variable that was read-only would be useless: it would never be given a value. Statistically, most writes are to local variables, rather than globals, so this makes for a useful indicator. A non-local variable that is written locally requires a special declaration.

15. Briefly describe two advantages of embedded server-side web scripting relative to CGI scripting (which is of course also server-side).

**Answer:** From the textbook:

Though widely used, CGI scripts have several disadvantages:

- The web server must launch each script as a separate program, with potentially significant overhead (though a CGI script compiled to native code can be very fast once running).
- Because the server has little control over the behavior of a script, scripts must generally be installed in a trusted directory by trusted system administrators; they cannot reside in arbitrary locations as ordinary pages do.
- The name of the script appears in the URI, typically prefixed with the name of the trusted directory, so static and dynamic pages look different to end users.
- Each script must generate not only dynamic content, but also the HTML tags that are needed to format and display it. This extra “boilerplate” makes scripts more difficult to write.

Problem Solving

16. (5 points) Here is an OCaml function to calculate the length of a list:

```ocaml
let rec len l =
  match l with
  | [] -> 0
  | h::t -> 1 + len t;;
```

This code is correct, but suboptimal. Explain why, and describe what you might do to improve it.
The code is not tail recursive, but can easily be modified to be so. The OCaml implementation will then reuse the current stack frame for each recursive call, reducing both time and space overhead.

```ocaml
let len l =
  let rec helper l n =
    match l with
    | [] -> n
    | h::t -> helper t (n + 1) in
  helper l 0;;
```

Questions 17 through 19 refer to the following context-free grammar for arithmetic expressions in postfix notation.

```
E → num
   → E un_op
   → E E bin_op
un_op → ~ | \nbin_op → + | - | * | /
```

Here the convention is that a function name follows its argument list, rather than preceding it. The conventional infix equivalent of $3 \, 2 \, 5 \, + \, * \, 2$ - is $(3 \times (2 + 5)) - 2$. You may think of ~ as denoting additive inverse (what is usually marked +/- on a calculator); you may think of \ as denoting multiplicative inverse (what is usually marked $1/x$ on a calculator).

17. (5 points) Give a parse tree for the string $3 \, \sim \, 4 \, 2 \, + \, *$.

**Answer:**

```
E
  E
    E
      E un_op
      |
      num(3) ~
    E
      |
      num(4)
  E bin_op
    |
    num(2) *
```

18. (6 points) Give a canonical (rightmost) derivation of this same string.
   (Hint: the answer is 10 lines long.)

**Answer:**

```
E
E E bin_op
E E *
E E E bin_op *
E E E + *
E E num(2) + *
E num(4) num(2) + *
```
9. \( E \) \( \rightarrow \) \( \text{num}(4) \) \( \text{num}(2) + \ast \)
\( E \sim \) \( \text{num}(4) \) \( \text{num}(2) + \ast \)
\( \text{num}(3) \sim \) \( \text{num}(4) \) \( \text{num}(2) + \ast \)

19. (8 points) Suppose we wish to translate our postfix expressions into fully parenthesized infix notation. We can associate a synthesized attribute \( s \) of type \text{string} with each symbol in the grammar (and thus each node in the parse tree). Assume that the scanner initializes this attribute for \( \text{num} \). For \( \text{un} \_\text{op} \) and \( \text{bin} \_\text{op} \) we can define trivial attribute rules as follows.

\[
\begin{align*}
\text{un} \_\text{op} & \rightarrow \sim \quad \triangleright \quad \text{un} \_\text{op}.s = \sim \\
\text{un} \_\text{op} & \rightarrow \backslash \quad \triangleright \quad \text{un} \_\text{op}.s = \backslash \\
\text{bin} \_\text{op} & \rightarrow + \quad \triangleright \quad \text{bin} \_\text{op}.s = + \\
\text{bin} \_\text{op} & \rightarrow - \quad \triangleright \quad \text{bin} \_\text{op}.s = - \\
\text{bin} \_\text{op} & \rightarrow * \quad \triangleright \quad \text{bin} \_\text{op}.s = * \\
\text{bin} \_\text{op} & \rightarrow / \quad \triangleright \quad \text{bin} \_\text{op}.s = / \\
\end{align*}
\]

Complete the attribute grammar by writing attribute rules for \( E \). These should accumulate, into attribute \( s \) of the root of the parse tree, an infix representation of the overall expression. In your answer to question 17, for example, the root of the tree would be decorated with “((~3)*(4+2))”.

\textbf{Answer:}
\[
\begin{align*}
E & \rightarrow \text{num} \quad \triangleright \quad E.s = \text{num}.s \\
E_1 & \rightarrow E_2 \text{un} \_\text{op} \quad \triangleright \quad E_1.s = \left( \sim + \text{un} \_\text{op}.s + E_2.s + \right) \\
E_1 & \rightarrow E_2 E_3 \text{bin} \_\text{op} \quad \triangleright \quad E_1.s = \left( \sim + E_2.s + \text{bin} \_\text{op}.s + E_3.s + \right)
\end{align*}
\]

\textbf{Extra Credit}

20. (6 EC points max) The context-free grammar of questions 17–19 is clearly not LL(1). Give an equivalent LL(1) grammar for postfix expressions, or explain why no such grammar can exist.

\textbf{Answer:}
\[
\begin{align*}
E & \rightarrow \text{num} \text{ET} \\
ET & \rightarrow E \text{bin} \_\text{op} \text{ET} \\
& \quad \rightarrow \text{un} \_\text{op} \text{ET} \\
\text{un} \_\text{op} & \rightarrow \sim \mid \backslash \\
\text{bin} \_\text{op} & \rightarrow + \mid - \mid \ast \mid /
\end{align*}
\]