B1 Compilers (25 pts) (Answer all questions.)

You have been hired (for real money) to write a compiler for a new programming language. Assume that you will write a compiler consisting of separate phases for lexical analysis, parsing, semantic analysis/intermediate code generation, symbol table management, and final code generation which generates assembly code. Answer the following questions concerning your new job.

a) [12 pts]

(1) What information do you need to obtain from your boss in order to write the compiler?

(2) For each piece of information, list the compiler phases for which you will need this information in order to implement the compiler phase.

(3) For each phase listed, explain why the phase needs this information.

Answer this question with the goal that when you are finished, you would be able to begin designing each phase without having to bother your boss with more questions.

b) [13 pts]

(1) For each compiler phase, give a list of design decisions that you will face in developing that phase.

(2) For each design decision, describe some of the common approaches to dealing with that issue and which features of the language may affect the approach taken.

B2 Compilers (25 pts) (Answer all questions.)
a) [10 pts]
(1) Give a regular expression that recognizes legal single-line comments of the form

/* any comment goes here*/

where the comment text does not include /* or */.

(2) Now, discuss how to adjust/extend your approach to comment handling in the scanner when your compiler is also required to accept the comment in each line below, *but also emit warnings for each.*
In each case, the comment encompasses all of the characters between the first /* on the line and the last */ on the line.
Please write down any assumptions you make about how your scanner deals with multiple regular expressions matching the same string, and a particular regular expression matching more than one substring of a given string.

```
/*  /*  any comment */
/********/*/
/*/*****/
/*  /* looks nested */  still in comment*/
```

and emit an error for:

```
/* neverending
```

b) [15 pts]
(1) Construct the LR(1) DFA for the following grammar.

S := T
T := xAu A | A | B
A := z | Gw
B := tE
E := GuT | GuE
G := z

(2) Show that the grammar in (1) is LR(1), but not LALR(1).

(3) Explain what it means to have a shift-reduce conflict in state X in terms of the LR items in the underlying LR DFA. (Don't just say "We could shift and go to the next state or reduce!")

(4) As the compiler writer, what can you do when you find out you have a shift-reduce conflict?

B3 Compilers (25 pts) (Answer all questions.)

a) [15 pts] The task of error detection and recovery is usually distributed among the various phases of the compiler, and reflects the capabilities and limitations of each phase.
For each error below, (i) state the earliest phase at which the error could be detected by the compiler, (ii) how the compiler writer could design the phase to detect such errors without writing lots of extra error-detection code, and (iii) a good way of recovering from the error to continue compilation/execution.

(1) character operand for an integer operator
(2) misspelled keyword
(3) missing operand to an expression
(4) variable use before declaration

b) [10 pts] Run-time storage management is an important task of a code generator. Storage management is "more interesting" when a language allows nested subroutine declarations and variable declarations inside any subroutine.

(1) Describe one method for generating code for handling nonlocal variable accesses when a language has STATIC SCOPING. In particular, explain the actions upon a variable reference, subroutine call, and subroutine return. State the run-time space and time requirements for your described method.

B4 Compilers (25 points) (Answer all questions.)

a) [9 pts] Symbol table management is crucial to efficiently compiling statically scoped, block-structured languages with nested subroutines and other features which create additional scoping environments. Describe how you would handle symbol table management and semantic checking (in particular, *multideclared identifiers* and *use before declaration* errors) when a language allows calls to subroutines declared later in the same file, as long as the subroutine is declared somewhere within the same scope as the call. (assume no function prototypes)

b) [8 pts] Describe how *live variable* data flow information can be used to improve register allocation.

c) [8 pts] Characterize when static, stack, and heap storage allocation methods are most appropriate, and when each is not appropriate. For each allocation method, give one example of a language feature for which it is most appropriate, and one for which it is not appropriate. In the latter feature, explain why the method is not appropriate.