1. [4 points] Pick the appropriate data structure for each purpose:
   - storage of information about identifiers: a. heap
   - input to code generation: b. stack
   - storage of local variables at run time: c. queue
e. parse tree
   - storage of objects at run time: d. hash table
   f. doubly linked list
   g. abstract syntax tree

2. [3pts] In a pure functional language there are no local variables. Is a stack still needed? Why or why not?

3. [15pts] Sometimes you really need something to run fast. One way to do it is to interleaver bits of hand-optimized assembler code in the source program. (This is not common, but it can be done.)
   Suppose you are chosen to modify the Triangle compiler so that it can accept Triangle programs which include sequences of TAM code. These will be delimited with `begintam` and `endtam`.

   Briefly explain how you would modify the various parts of the Triangle compiler to allow this. Also explain the implications for the ASTs.
4. [5pts] In addition to JUMP, which will "jump to code address (d + register r)", TAM also has a JUMPI instruction, which will "pop a code address from the stack, then jump to that address". Which of the following would this be useful for, and why?

- loops
- recursive functions
- switch statements
- conditionals
- nested conditionals
- array access
- method invocation

5. [5pts] Recall that we saw how to build a parse table from the rules of the grammar, and how to use this for parsing. Now briefly describe a way to go the other way, that is, to generate random programs (for example, for testing a compiler) using either a parse table or the raw set of grammar rules.

6. [4] Group the following types of languages into equivalence classes

- languages described by regular expressions
- languages handled by lex
- languages described by BNF
- context-free languages
- context-sensitive languages
- languages recognized by finite automata
- languages recognized by push-down automata
- languages generated by Turing Machines

7. [3] Process switches are handled by the Operating System. Thread switches are handled by the Runtime Environment. Which is faster? Why?

8. [3] Explain why identifier names are not needed at run time.
9. [6] Give two dissimilar examples of type coercion and explain how a compiler would deal with each.


11. [3] Given a statement like object1.field3 = 0, what computation is required at run-time to compute the address?


13. [12pts] If you use an optimizing compiler you can expect some things to be different. For each of the below, circle the right answer and explain briefly why.
   
   (a) slower compile time [always, sometimes, never]

   (b) faster run time [always, sometimes, never]

   (c) smaller object code [always, sometimes, never]

   (d) more confusing run-time error reports [always, sometimes, never]

   (e) more confusing compile-time error reports [always, sometimes, never]

   (f) more accurate computations [always, sometimes, never]
14. [20] Briefly define or explain:

(a) garbage collection

(b) handle

(c) scope

(d) static link

(e) backpatching

(f) code template

(g) binding

(h) code movement

(i) register allocation

(j) scanner
(k) Yacc/Bison

(l) translator

(m) cross-compilation

(n) emulator

(o) bootstrapping (in general)

(p) grep

(q) pipe (in Unix)

(r) basic block

(s) ambiguous grammar

(t) peephole optimization
15. [5pts] cc, the old C compiler, often produced the message “Syntax error at line x”, without specifying anything more. What functions would you change in cc to make it able to say specifically what the problem was? State any assumptions you make.

16. [6pts] Your boss calls you in and says “I’ve downloaded this great freeware program, bluedog, which does exactly what we want. Unfortunately it’s too slow and it uses too much memory. Oh, and bluedog is written in this new language BP, but I downloaded a freeware compiler for that too. Get to work.”


[4] Briefly say how you’d go about making bluedog run faster and use less memory.

17. [15] The Roboland language needs a break command, to allow termination of loops. 1. Design a code template showing how code generation will handle break. 2. Also design an extension to the AST objects to handle break. 3. Redesign some module of the compiler to allow detection of breaks that are not within loops.

For full credit, your designs should work for break commands in nested loops, and inside conditionals inside loops.