Sample Midterm Exam Student Name: ________________

I) Answers to T/F (**3 point each**): **True answers are marked "T", False are left unmarked, 5pts for T/F on left.**

1) The number of HLLs' compilers/interpreters decides the number of different machine languages in your computer.
2) The execution of *micro* level instructions invokes *Assembly machine code* level routine from the computer RAM.
3) The lower the level of a programming language, the more it abstracts the machine hardware details to its users.
4) The Pascal *by-reference* & *by-value* are more *powerful* and *secure* than that of Algol *by-name*.
5) The "*stack* model of computation" forces *static* binding of variables' names to their memory locations (elaboration).
6) The compiler uses the names' synthesized and inherited types in its *lexical-analysis* phase.
7) The addition of enumerated types to Pascal made it more secure than both FORTRAN & Algol. T
8) The "type" section in Pascal (similar to the "typedef" in C) binds names to their types and allocates their appropriate memory spaces.
9) Some of the dynamic *type* checking might still be resolved at compile time.
10) Generally speaking, compiled HLLs execute much slower than interpreted HLLs.
11) The *symbol table* is used in the compiler's code generation phase. **T**
12) In Algol and Pascal, binding of names to their memory locations is done at compile time.
13) In Algol, the interpreting environment of a "local" name is strictly a "complex" set of nested scopes.
14) In scoped languages, the CPU will access any non-local names, simply by following the contour diagram rules.
15) In early FORTRAN, there was no "recursion" facility in the language, for efficiency. T
16) *Operator overloading* in Pascal is a form of secure ***polymorphic*** power. **T**
17) Programming at the system *micro* level is a tradeoff between ***execution speed*** (***efficiency***) and ***security***. **T**
    At the micro-level languages, there is no security protection as in higher level languages.
18) FORTRAN introduced very early *true* built-in abstract data types (ADTs). T
19) Early FORTRAN is an example of sacrificing efficiency for power and security.
20) All ***security loopholes*** in Algol are typing system related.
21) An HLL's *translation* may map that HLL's code into *machine* code. **T**
    It may map to machine code, but not to micro code, what about if the machine is hardwired?
22) Definitely, there is no way we can write a secure program code in C or FORTRAN, they are insecure HLL's.
23) *Dynamically* scoped HLLs are less *powerful* than *statically* scoped HLLs.
24) Aside from pass *by-name*, Algol provides a more *secure* programming environment than Pascal, C, and FORTRAN.
    Still the lack of enumerated types will force "overworking" of integer with other types!
25) A powerful HLL provides the programmer with *secure* programming environment.
26) "Missing Parentheses" is an error message to be generated by the compiler's ***lexical analyzer***. **T**
    The compiler Lexical analyzer is concerned with the syntax issue of matching parentheses!
27) Pascal "pointers" are more secure than the notion of integer "addresses" in C. T
28) Alogl was the first HLL to attempt a very naïve mechanism for passing functions/procedures as parameters. T
29) At compile time, the CPU will always find and access memory locations of all names declared in the program.
30) In general, a HLL compiler is able to decide if a variable is *initialized* or not.
31) FORTRAN utilize stack model of computation as in Pascal.
32) In a dynamically scoped HLL program's ***contour diagram***, a box is drawn for every declared module (function/procedure/block), yet it defines the scope of visibility of any locally declared names.
33) Algol and Pascal are more ***secure*** and ***powerful*** than FORTRAN. T
34) In an AR, the DL and SL might be the same, only when the definer and the caller of the *callee* are the same. T
35) All names of all defined procedures and functions are automatically "local" names in their defining modules. T
36) In stack model of computation, the activation record of any *callee* procedure/function will be ***popped*** out of the run-time stack. T
37) The environment of any program construct (e.g., statement/expression) is the set of all declared scopes in the program.
38) In statically scoped HLLs, not every ***declared*** name is ***accessible***. T
39) In general, HLL power contradicts run-time execution efficiency and sometimes code readability. T

II) The following 35 questions are multiple choices; select (circle) the **BEST** answer (**5 pts each**):

1) We study HLLs mainly in order to:
a) speed up code execution b) find the cost when purchasing their compilers/interpreters.
c) improve existing HLLs and/or design future new HLL
d) c & have the best language choice to solve a problem
e) d & make them powerful f) none of the above

2) The following factors make for a "**good**" HLL, regardless of the environment of its usage:
a) compiled or interpreted translation b) how easy and direct to program the hardware components
c) b & how expensive is its translator d) a and c above e) none of the above

3) The following language mechanisms will add ***power*** to their hosting (providing) HLL:
a) recursion, dynamic scoping, and pass by-name
b) a & dynamic arrays c) b & dynamic type checking d) a & code reusability
e) c & functions as first class f) e & code sharing (inheritance) g) f & pass by-value

4) These HLL mechanisms are a tradeoff between ***security*** (gain) and *execution* ***efficiency*** (loss):
a) Algol's *by-name* parameter passing mechanism b) name *aliasing* (*by-ref* and *global* names)
c) a & b d) dynamic type checking f) none of the above

5) All data types are all *inherently* ***true*** abstract data types (***ADTs***) in the following HLL category/domains:
a) block-structured b) Hybrid of functional and imperative c) functional d) logic
e) pure Object Oriented f) c & d g) a, b, & d h) none of the above

6) The most ***abstract*** HLL paradigm is the ***pure***:
a) imperative b) block-structured c) a and b d) object oriented e) d and functional
h) none of the above

7) The following Pascal program compilation's phases are arranged in the right order:
a) "syntactic-analyzer"→scanner→"semantics-analyzer"→optimization→"code-generation"
b) scanner→" optimization"→" semantics -analyzer"→" syntactic –analyzer" →"code-generation"
c) scanner→"syntactic-analyzer"→optimization→"code-generation"→"semantics-analyzer"
d) scanner→"syntactic-analyzer"→"semantics-analyzer"→ "code-generation" →" optimization"
e) none of the above

8) Early FORTRAN is an example of a HLL that is very:
a) platform independent b) secure c) efficient d) abstract e) general purpose
f) none of the above

9) In addition to ***overworking*** the *integer* type with *label* type, the following caused a potential security loophole in FORTRAN:
a) operator overloading b) implicit name declaration c) global name declaration
d) pass *by-value*-result e) syntax similarity of totally different semantics constructs f) b & e
g) none of the above

10) Some efficient feature(s) of the COMMON and EQUIVALENCE mechanisms in FORTRAN is(are):
a) security of name access b) implicit typing c) alleviating the lack of global name access
d) sharing memory e) c and d f) a and e g) none of the above

11) When procedure ***Q*** calls procedure ***P***, and just before the execution of ***P***'s code starts, the activation record (AR) of ***P*** will contain the following:
a) a pointer to the AR of the caller of ***Q*** b) ***P***'s static nesting level c) a pointer to ***P***'s AR
d) the actual return address into the code of ***P*** e) a pointer to ***Q***'s AR f) none of the above

12) In Pascal, if ***X*** is a name <u>encountered</u> in procedure ***Q*** the compiler will look it up ***<u>first</u>*** in the:
a) environment of definer of ***Q*** b) environment of caller of ***Q*** c) AR of the main-program
d) the locally declared names in ***Q*** e) actual parameter of the ***Q*** call statement
f) none of the above

13) Some of the <u>major feature(s)</u> that Algol and early FORTRAN have shared is(are):
a) recursion, dynamic arrays, pass by-name, blocks, and free-format, stack model of computation
b) global variable declarations, nesting of scopes, and compound statements
c) powerful structuring constructs (e.g., the *for*, *switch*, and *if* statements)
d) all of the above e) d & the *contour diagram* f) d & dynamic and static scoping
h) none of the above

14) The ***<u>aliasing</u>*** of more than one name into the same memory location is a side effect of the following language feature(s):
a) COMMON and EQUIVALENCE in FORTRAN b) Algol pass ***by-name***
c) pass *by-reference* and *global* name visibility d) "***<u>union</u>***" structures in ***<u>C</u>*** (Pascal's ***variant*** records)
e) d & dynamic arrays f) a, c, & d g) f & operator overloading

III Short Answers Questions are directly from the Lectures' notes, textbook, HW's, and Quizzes.