Final Exam: Part 1 of 2

COMP 3010 – Organization of Programming Languages; May 8, 2018 – Dr. Wilkes

Note: *This exam is closed book and notes, except for two 8.5x11" sheets of paper with handwritten notes (no photocopies) & the printed OCaml Language reference sheet.*

Multiple Choice Questions – 2 points each: Circle the correct answer.

1. Which stage of a compiler traverses an abstract syntax tree, and produces output in either a low-level or high-level language that can be executed (perhaps after further processing)?
   - a. Code generator
   - b. Code improvement stage
   - c. Parser
   - d. Scanner
   - e. Semantic analysis stage

   code improvement phase of compilation performs a variety of transformations on the control flow graph

   a semantic analyzer,which could then create a syntax tree and populate a symbol table, and then pass it all on to a code generator

   The semantic analyzer traverses the tree, performing all static semantic checks and initializing various at-tributes(mainly symbol table pointers and indications of the need for dynamic checks) of use to the back end

   The machine-independent code improvement phase of compilation performs a variety of transformations on the control flowgraph

2. Which stage of a compiler reads a stream of tokens, and produces a tree based on productions in the language's grammar specification?
   - a. Code generator
   - b. Code improvement stage
   - c. Parser
   - d. Scanner
   - e. Semantic analysis stage

3. **(CIRCLE ALL THAT APPLY)** If a grammar can be implemented using a bottom-up parser, what type of grammar can it be?
   - a. LL
   - b. LR
   - c. LALR
   - d. SLR
   - e. None of the above

4. **(CIRCLE ALL THAT APPLY)** If a grammar contains right-recursive productions, what type of grammar can it be?
   - a. LL
   - b. LR
   - c. LALR
   - d. SLR
   - e. None of the above

5. **(CIRCLE ALL THAT APPLY)** If a grammar is ambiguous, what type of grammar can it be?
   - a. LL
   - b. LR
   - c. LALR
   - d. SLR
   - e. None of the above

Name: _____

6. **(CIRCLE ALL THAT APPLY)** Phases that are commonly associated with the "back end" of a compiler include:
   a. Machine-independent code generator
   b. Machine-independent code improvement stage
   c. Machine-dependent code generator
   d. Machine-dependent code improvement stage
   e. Parser
   f. Scanner
   g. Semantic analysis stage
7. **(CIRCLE ALL THAT APPLY)** Which of the following is an example of a "low-level" or "medium-level" intermediate form?
   a. Abstract syntax tree
   b. Assembler language
   c. Register Transfer Language (RTL)
   d. Stack machine instructions (e.g., Pascal P-code or Java bytecode)
   e. Three-address instructions (quadruples)
8. In OCaml, an expression such as `(3.14159 + 5)` is an example of:
   a. Type compatibility
   b. Type equivalence
   c. Type incompatibility
   d. Type inference
   e. None of the above
9. In C, the "dangling `else`" problem was addressed by:
   a. Adding an "end marker" for the `if` statement to the grammar
   b. Including a semantic rule in the language reference manual stating that an `else` clause matches the closest unmatched `if` clause
   c. Using indentation to indicate which `else` clause matches which `if` clause
   d. None of the above
10. In OCaml, a function definition such as `let plus a b = a+b;;` is an example of:
    a. Type compatibility
    b. Type equivalence
    c. Type incompatibility
    d. Type inference
    e. None of the above
11. In C++, an expression such as `reinterpret_cast<int>(95.7)` is an example of:
    a. Non-converting type cast
    b. Type coercion
    c. Type conversion
    d. None of the above

12. In OCaml, the ability to declare a record with pair or list elements, or vice versa, is an example of:

Lists in ML are homogeneous: every element of the list must have the sametype.
Both ML and Lisp provide a wealth of built-in polymorphic functions to manipulate arbitrary lists.

a. Dynamic typing
b. Orthogonality
c. Polymorphism
d. Static typing
e. Strong typing
f. None of the above

13. In C, the function definition

```
int min(int a, int b) {return (a<b)?a:b;}
```

is an example of:

https://www.sitepoint.com/typing-versus-dynamic-typing/
-Dynamic typing, in fact, does save you from writing a few "extra" lines of code, which, in turn, means less time spent writing code.
-Static typing does not imply that you have to declare all the variables first, before you use them
-Strong typing: Programming languages in which variables have specific data types are strong typed. This implies that in strong typed languages, variables are necessarily bound to a particular data type.
-Orthogonality is the property that means "Changing A does not change B". An example of an orthogonal system would be a radio, where changing the station does not change the volume and vice-versa. A non-orthogonal system would be like a helicopter where changing the speed can change the direction. In programming languages this means that when you execute an instruction, nothing but that instruction happens (very important for debugging).

a. Dynamic typing
b. Orthogonality
c. Polymorphism
d. Static typing
e. Strong typing
f. None of the above

14. In C++, the function definition

```
template<typename T>
T min(T a, T b) {return (a<b)?a:b;}
```

is an example of:

Explicit parametric polymorphism, also known as generics (or templates in C++), typically appears in statically typed languages, and is usually implemented at compile time. The implicit version can also be implemented at compile time—specifically, in ML-family languages; more commonly, it is paired with dynamic typing, and the checking occurs at run time.

a. Dynamic typing
b. Orthogonality
c. Polymorphism
d. Static typing
e. Strong typing
f. None of the above

15. (**CIRCLE ALL THAT APPLY**) Which of the following best describe the type system in C++?

a. Dynamically typed using name equivalence
b. Dynamically typed using structural equivalence
c. Statically typed using name equivalence
d. Statically typed using structural equivalence
e. Strongly typed using name equivalence
f. Strongly typed using structural equivalence

https://docs.microsoft.com/en-us/cpp/cpp/cpp-type-system-modern-cpp?view=vs-2019

16. In OCaml, the expression "`(1, 2, 3)`" represents:

a. An array with the three integer elements 1, 2, & 3. [, ,]
b. A list with the three integer elements 1, 2, & 3. []
c. A record with the three integer elements 1, 2, & 3. {}
d. A tuple with the three integer elements 1, 2, & 3. (, ,)
e. A variant with the three integer elements 1, 2, & 3. |
f. None of the above.

Name: _____

17. Given the following definition of function `f`, what does the expression "`f [1; 2; 3];;`" return?

```
let rec f list1 =
   match list1 with
   | [] -> 1
   | head::rest -> head * f rest;;
```

a. 0
b. 6
c. 120
d. 123
e. 456
f. [3; 2; 1]
g. [4; 5; 6]
h. [6; 5; 4]
i. Error message
j. None of the above

18. Which of the following is the correct meaning of the C declaration "`double(*a[n])();`"?

Pointer to an Array:
data_type (*var_name)[size_of_array]; Example: int (*ptr)[10];

Declare an array of five pointers-to-integers:
int *array_of_pointers[5];

a. `a` is an array of `n` pointers to `doubles`
b. `a` is a pointer to an array of `n` `doubles`
c. `a` is an array of `n` pointers to functions returning `doubles`
d. `a` is a function returning a pointer to an array of `n` `doubles`
e. None of the above

https://stackoverflow.com/questions/2192620/array-of-n-pointers-to-functions-returning-pointers-to-functions?fbclid=IwAR18yXecXH6g4UP7XmHWKqHlc4qas7aKCBth8-s5MgtlBbm43Wvox15moxs

19. (**CIRCLE ALL THAT APPLY**) For a language supporting both static and dynamic array bounds, in which of the following array allocation situations would the compiler **NOT** allocate the array on the stack?

a. A global array variable for which the bounds are static and known at compile time
b. A local array variable for which the bounds are static and known at compile time
c. A local array variable for which the bounds are static and known at run time
d. A local array variable for which the bounds are dynamic and known at run time
e. None of the above

20. The "mark-and-sweep" algorithm for garbage collection (used in the runtime environment for languages such as OCaml and Java) uses which of the following techniques to minimize the need for additional storage during execution of the algorithm?

→ a. Locks and keys
b. Pointer reversal
c. Reference counts
d. Tombstones
e. None of the above

So we need to clear heap memory by releasing memory for all those objects which are no longer referenced by the program (or the unreachable objects) so that the space is made available for subsequent new objects. This memory can be released by the programmer itself but it seems to be an overhead for the programmer, here garbage collection comes to our rescue, and it automatically releases the heap memory for all the unreferenced objects.

Name: _____

True/False Questions – 1 point each: Write T or F beside each question.

A regular expression followed by a Kleene star, meaning the concatenation of zero or more strings generated by the expression in front of the star

T 1. Kleene closure (such as `A*`) indicates one or more repetitions of the pattern `A`.

T 2. Operator precedence can be specified in a grammar by incorporating productions which cause the higher-precedence operators to appear "lower" (closer to the leaf nodes) in the parse tree of an expression.

by <= n

F 3. An LR(*n*) or LL(*n*) parser must sometimes "look ahead" by *n* characters.

https://www.csee.umbc.edu/courses/331/fall13/03/notes/04/04cparsing.ppt.pdf

T 4. Epsilon productions are more common in LL grammars than in LR grammars.

T 5. An *L-attributed* grammar may contain both synthesized and inherited attributes.

F 6. A bottom-up parser may incorporate semantic attributes via action routines.

https://pdfs.semanticscholar.org/1dd4/6f1f6929be6d7cc67e3fb2c357472490fe87.pdf

T 7. Most front ends for the GCC compiler use an AST-based intermediate form.

T 8. A machine-independent code generation phase can assume that an unlimited number of registers are available.

T 9. The OCaml type signature "`val f : 'a -> 'a = <fun>`" indicates that function `f` has a parameter and return value that can be any type, as long as the types of `f`'s parameter and return value are the same.

T 10. C supports both the *denotational* and *abstraction-based* view of types. C++ => T

F 11. Programming languages that are highly orthogonal tend to be more "flexible" than languages that are less orthogonal.

T 12. The real number type is an example of a scalar type that is not a discrete type.

https://www.adaic.org/resources/add_content/standards/05rm/html/RM-3-5.html

T 13. The support in C for a large number of coercions between different types makes it easier for the reader to understand programs that rely on these coercions.

T 14. "Pure" functional languages do not allow side effects. https://stackoverflow.com/questions/1916692/are-side-effects-possible-in-pure-functional-programming

book p.570: lazy evaluation is not transparent in the presence of side effects

F 15. Logic languages such as Prolog rely on specification of constraints ("what" the program should do) rather than algorithms ("how" the program should do it).

T 16. "Lambda expressions" in C++11 and later provide some of the capabilities incorporated in functional languages. book p.538: C++11 and Java8 provide lambda expressions, but without unlimited extent
book p.685: Futures are also available in C++, where they are designed to interoperate with lambda expressions,

F 17. Ada is widely used as a programming language for rapid prototyping, but is seldom used for development of critical systems due to its lack of type safety. book p. 12

T 18. Most functional languages rely on the programmer to explicitly allocate and deallocate items stored in the heap. book p. 124

T 19. The C array declaration "`char *my_array[10];`" will result in a contiguous allocation of 10 C-strings.

F 20. If a multidimensional array is allocated using the contiguous allocation technique, access to an individual element of the array requires a complicated address calculation involving multiple multiplications and additions. https://www.sciencedirect.com/topics/computer-science/address-calculation

Final Exam: Part 2 of 2

COMP.3010 – Organization of Programming Languages; May 8, 2018 – Dr. Wilkes

Note: *This exam is closed book and notes, except for two 8.5x11" sheets of paper with handwritten notes (no photocopies) & the printed OCaml Language reference sheet.*

Short Answer Questions – 10 points each: Write your answer in the space provided.

1. Suppose we are compiling for a machine with 1-byte characters, 2-byte shorts, 4-byte integers, and 8-byte reals. The alignment rules for this machine require the beginning address of a real to be a multiple of 8 bytes, and all of the other primitive data types to be a multiple of 4 bytes. Suppose further that the compiler is not permitted to reorder fields. **How much space will be required by the following array? Explain.**

```
A: array [0..9] of record
  s: short;  2
  c: char;   1
  t: short;  2
  d: char;   1
  r: real;   8
  i: integer 4
end
```

https://stackoverflow.com/questions/4374276/alignment-rules

- Structural equivalence is based on the content of type definitions: roughly speaking, two types are the same if they consist of the same components, put together in the same way.
- Name equivalence is based on the lexical occurrence of type definitions: roughly speaking, each definition introduces a new type.

2. Consider the following pseudocode:

```
type T = array [1..10] of integer
type S = T => type S = array [1..10] of integer
A : T
B : T
C : S
D : array [1..10] of integer
```

   a. Which of the variables will a compiler consider to have compatible types under **structural equivalence**?

   S and T
   variables A, B because they have the same structure T

   b. Which of the variables will a compiler consider to have compatible types under **name equivalence**?

   A, B, C, D because they are in lexical order.

Name: _____

3. Consider the following grammar:

stmt ⟶ *assignment*
⟶ *subr_call*
assignment ⟶ `id := expr`
subr_call ⟶ `id ( arg_list )`
expr ⟶ *primary expr_tail*
expr_tail ⟶ *op expr*
⟶ ϵ

primary ⟶ `id`
⟶ *subr_call*
⟶ `( expr )`
op ⟶ `+ | - | * | /`
arg_list ⟶ *expr args_tail*
args_tail ⟶ `,` *arg_list*
⟶ ϵ

Draw the parse tree for the statement "`foo(a,b)`".

https://www.chegg.com/homework-help/questions-and-answers/consider-following-grammar-stmt-rightarrow-assignment-rightarrow-subrcall-assignment-right-q21930679

4. Both interpretation and code generation can be performed by traversal of an abstract syntax tree (AST). Compare these two kinds of traversals. In what ways are they similar, and in what ways are they different?

https://www.chegg.com/homework-help/Programming-Language-Pragmatics-4th-edition-chapter-1-problem-6E-solution-9780124104099?fbclid=IwAR3J-NEAcha0QurdzgZR4BobgWjcZYBd9efeEBfPxsxu9kf6QmvACk1wXdE

https://modeling-languages.com/executable-models-vs-code-generation-vs-model-interpretation-2/
- The code-generation strategy involves using a model compiler (many times defined as a model-to-text transformation) to generate a lower-level representation of the model using existing programming languages and platforms (e.g. Java). Instead, the model interpretation strategy relies on the existence of a virtual machine able to direclty read and run the model (example of a proposal for a UML virtual machine).