Midterm 01 Exam Review Sheet

Logistics

Review Tips

In order to have a successful exam, the student should do the following:

• Review lecture slides and class notes [by Prof. Mohan] in google drive folder and the student’s class notes.
• Clear browser cache and get new slides before the exam. There might be some changes that will not show if you do not clear cache and download a new copy of the slides.
• Go through the reading assignments at the end of each slide, and read the textbook chapters. You can skip topics that were not discussed in class.
• Class activity exercises
• Lab assignments
• Quizzes

Exam Format

• Paper based
• Questions will be aimed at basic understanding of concepts and the ability to apply them in concrete examples.
• Question types will include:
  1. Short answer (may require writing programming statements, or analyze a given code, and writing descriptive answers through diagram illustration.)
  2. True/false
  3. Multiple choice
• You will not be asked to write whole programs; however, you have been exposed to a number of concepts through languages other than Java (in labs, in class) and you will be expected to recognize features of such languages when they were highlighted in class (e.g., used of postfix notation in PostScript, or syntax for different scoping rules in JavaScript).

Additional Details

• Exam is on: 10/02/2018 2:30 PM - 4:30 PM at Alden 101. The submissions after 4:30pm will be penalized for lateness (unless you have special arrangements). If you plan to be late to the exam starting time, you must let me know in advance.
There would be three booklets given to you during the exam. The first is a question booklet, that contains the list of questions in the exam. The second is an answer booklet, where the students are expected to write their answers to the questions given in the exam. The third is an outline document, that provides an outline to the attributes of right answers. At the end of the exam, it is the responsibility of the student, to combine all the three sections of the student’s exam using a paper clip provided and return back the exam to Prof. Mohan. After grading completion, the graded answer booklets would be returned back to the student.

The exam will be closed [notes,lecture slides, textbook, other teaching materials, and NO internet].

I highly recommend you use a black pen (preferably Pilot G -2) for writing your answers during the exam. Please note, it is very difficult for me to read the pencil writing and hence there might be a chance for inaccuracy while grading.

It is better to give part of an answer than to leave a question blank. No partial credit can be given for wrong answers if there is no accompanying work. If you leave a question blank, then there will be no points awarded to the question.

Add necessary justification to your answer, if your understanding of the question deviate from the actual question. This rule also applies to multiple choice questions. I may give you partial credit or even full credit, based on how good you have justified your answer.

I would strongly encourage you to make use of my office hours, to discuss and/or clarify any topic related to the finals.

Topics covered

The exam will cover all material up through Monday, 1st October, 2018. The main topics we have covered so far are:

- Compilation and interpretation (chapter 1)
- Binding, names, and scope (chapter 3)
- Control structures (loops, conditionals, recursion, chapter 6)
- Data types—basic types; type systems; polymorphism; strong typing; static typing; composite types (chapter 7)
- Arrays; indexing (chapter 7)

Exam Prep Guide

Here are examples of the kinds of questions that might be asked. This is not intended to be a sample exam; the topics covered below are not intended to be an exhaustive review. In particular, knowing the answers to all the questions below will not guarantee a good grade on the exam!

1. List, in order, the steps involved in compiling a program. A diagram illustration may be required.

2. What is the difference between “block level scoping” and “function level scoping”? Give an example of a language that uses function level scoping.

3. Explain the difference between prefix, infix, and postfix notation, and give examples of each in languages that we have studied (identify the language and which notation is being used).

4. Give an example of a “ternary operator” (show the operator and explain its use with an example).
5. Given the following grammar (start symbol $S$, terminal symbols $a$, $+$, and $b$):

$$
S \rightarrow S + V | V \\
V \rightarrow a | b | + S
$$

(a) [True/False] The following is a valid parse tree for the string “$+a++b$”:

(b) Is it possible to draw a parse tree for the string “$a+b+$”? If so, draw one, if not, illustrate the problem.

(c) Is it possible to draw a parse tree for the string “$+++a$”? If so, draw one, if not, illustrate the problem.

6. Here is an excerpt from a valid Java program (line numbers are included as references). Assume “Something” is the name of another class.

```
10 // INSTANCE VARIABLES:
11 private int i, j;
12
13 // METHODS:
14 public void a(int i, int j) {
15    double x, y;
16    Something z = new Something();
17    ...
18 }
19 public void b(int p, int q) {
20    int x, y;
21    ...
22 }
```

(a) Using line numbers to refer to portions of the program and naming the affected variables in the program, illustrate an example of “hole in scope”

(b) In line 14, what kind of memory allocation is used for parameters $i$ and $j$?

(c) In line 15, what kind of memory allocation is used for local variables $x$ and $y$?

(d) In line 16, what kind of memory allocation is used for the new object created for variable $z$?

(e) If method $a$ calls method $b$ and method $b$ makes a reference to a variable named $i$, which “$i$” does it refer to—the one declared in line 11 or the one declared in line 14?

7. What kinds of data might be stored in static memory?

8. Suppose an imaginary language contains the following function and variable definitions:
int i = 10; // global
function f() {
    int i = 20; // local
    g(); // call function g
}
function g() {
    print(i);
}

What will $g$ print if this language uses lexical scoping? What will it print if dynamic scoping is used?

9. [Multiple choice.] During which phase of computation does a compiler detect errors such as missing semicolons or unmatched parentheses?

   (a) lexical analysis  
   (b) semantic analysis  
   (c) syntactic analysis  
   (d) none of the above

10. Given a logical expression, how does a bitwise operator [And, OR] and shift operator [left, right] work?

   (a) For example 01:
       
       11 & 5 → 1011 & 0101
       1011 & 0101 → 0001 [1]
       Hence 11 & 5 → 1
       
       Note: Here we translate decimal [11 and 5] into its binary form. We had detailed discussion in class about this conversion using truth table. Refer class notes if you need additional details about truth table. Logical "AND" [\&] states that both inputs need to be 1, for the output to be 1. If any one of the input is 0 then the output is 0.

   (b) For example 02:
       
       12 | 3 → 1100 | 0011
       1100 | 0011 → 1111 [15]
       Hence 12 | 3 → 15
       
       Note: Here we translate decimal [12 and 3] into its binary form. We had detailed discussion in class about this conversion using truth table. Refer class notes if you need additional details about truth table. Logical "OR" [\|] states that both inputs need to be 0, for the output to be 0. If any one of the input is 1 then the output is 1.

   (c) For example 03:
       
       13 << 2 → 1101 << 2
       1101 << 2 → 110100 → 2^5 + 2^4 + 2^2 → [52]
       Hence 13 << 2 → 52
       
       Note: Here we translate decimal [13] into its binary form. We had detailed discussion in class about this conversion using truth table. Refer class notes if you need additional details about truth table. Logical "SHIFT LEFT" [\<<] indicates that the the bits in binary form are shifted towards left by X bits. X is always defined within the shift operation. In this example, X = 2. All the bits are shifted left by 2 bits, which in a sense makes the output to be 6 bits with the trailing 0's added. One can use binary arithmetic to find the decimal equivalent, using the powers of 2 representation as shown above. The output is 52.

   (d) For example 04:
       
       14 >>= 3 → 1110 >> 3
1110 >> 3 \rightarrow 0001 \rightarrow 2^0 \rightarrow [1]
Hence 14 >> 3 \rightarrow 1

Note: Here we translate decimal [14] into its binary form. We had detailed discussion in class about this conversion using truth table. Refer class notes if you need additional details about truth table. Logical "SHIFT RIGHT" [\gg] indicates that the the bits in binary form are shifted towards right by X bits. X is always defined within the shift operation. In this example, X = 3. All the bits are shifted right by 3 bits, which in a sense makes the output to be 4 bits with the leading 0's added. One can use binary arithmetic to find the decimal equivalent, using the powers of 2 representation as shown above. The output is 1.

11. Is Java a compiled language, an interpreted language, or something in between? Justify your answer!

12. What is “garbage”? Give a short code example that illustrates how garbage can be created.

13. What kinds of information go into a frame (activation record)? Try to be as specific as possible.

14. Explain short-circuit evaluation with a simple example involving the logical “or” operator “\|”.

15. Which of the following Java functions correctly finds the sum of the digits in a nonnegative integer named \texttt{eeek}? For instance, \texttt{sum(354)} = 3+5+4=12 and \texttt{sum(10223)} = 1+0+2+2+3= 8.

\begin{verbatim}
(a) int sum(int eeek) {
   int r = eeek % 10;
   int q = eeek / 10;
   return r + sum(q);
}
(b) int sum(int eeek) {
   if (eeek == 1)
      return 1;
   else
      return 1 + sum(eeek/10);
}
(c) int sum(int eeek) {
   if (eeek < 10)
      return eeek;
   else
      return sum(eeek/10);
}
(d) int sum(int eeek) {
   if (eeek == 0)
      return 0;
   else
      return eeek % 10 + sum(eeek/10);
}
\end{verbatim}

The correct answer is (d). There is an infinite recursion in (a) (no base case tested); (b) is just wrong (makes no use of the digits); (c) returns only the last digit, not the sum of the digits.

16. In Java, an object may be considered to be of more than one type; for instance, an instance of a subclass named \texttt{Kong} can also be considered as an instance of the parent class \texttt{Gorilla}. What is the term for the ability of a variable to be simultaneously considered as two different types?
Answer: When a variable can be simultaneously viewed as more than one type, we have \textit{polymorphism}.

17. Define “orthogonality” in a programming language.
Answer: Orthogonality is a property that measures how well different language features work together. During class discussion, detailed examples of Orthogonality were discussed. Refer to class notes for additional details.

18. In Scott’s book, the following phrase appears: “It is sometimes argued that iteration is more efficient than recursion. It is more accurate to say that \textit{naive implementation} of iteration is usually more efficient than naive implementation of recursion.” What type of recursion can be handled as efficiently as iteration?
Answer: Functions that rely on “tail recursion” can be efficiently implemented without actually performing any recursive calls.

19. What are the two principle purposes of “types” in a language (according to the textbook)?
Answer: Types provide an implicit context for certain operations, freeing the programmer from having to explicitly list all the assumptions and requirements necessary for an operation to take place. Types limit the set of operations that may be performed on values, preventing invalid or nonsensical operations from being carried out.
20. Give examples of primitive types; give examples of composite types.
   Answer: Primitive types include things like `int`, `char`, `boolean`. Composite types include things like
   arrays, records, lists, and sets.

21. What is an “enumerated type”? Give an example.
   Answer: An enumerated, or enumeration, type is just a collection of named elements, with no other
   properties (other than the ability to be compared with each other and tested for equality). In C we
   can write:

   ```
   typedef enum {michelangelo, donatello, raphael, leonardo} ninjaturtle;
   int main() {
       ninjaturtle x = raphael, y = leonardo;
       if (x != y) printf("different\n");
       else printf("same\n");
   }
   ```

   In Java we can write:

   ```
   enum Turtle {
       mike,don,ralph,leo;
   }
   public class Ninja {
       public static void main(String[] args) {
           Turtle a = Turtle.mike, b = Turtle.don;
       }
   }
   ```

22. If you have tried to track down precise definitions of the terms “strongly typed” and “weakly typed” you have
   probably come away frustrated. The textbook (Scott) defines a “strongly typed language” as one that “prohibits
   … the application of any operation to any object that is not intended to support that operation.” Give an
   example of such an operation and such an object, using only standard primitive types.
   Answer: For instance, `false += 1` is meaningless in Java; `10+'a'` is disallowed in Haskell; etc. Note that in both Java and C, the expression `10+'a'` is legal, so examples of this are not universally
   valid in every language.

The above is not a full review! (For example, I will ask at least one question about Java bytecode.)