

All questions based on 12th Edition of textbook

Chapter 2

Question 2

Source code can be found in in QuickSort.java

Sample output:

```
C:\Users\phunm\Downloads\test>java QuickSort 0
Running non-recursive quicksort
Sorted arrayay:
5 19 54 56 67 86 94 438 1220 1230 9213
C:\Users\phunm\Downloads\test>java QuickSort
Running recursive quicksort
Sorted arrayay:
5 19 54 56 67 86 94 438 1220 1230 9213
```

The code for this question implements a quicksort algorithm both with and without recursion. The code is implemented in Java and separates the sort into 2 separate functions, which can be ran individually via launch args. The recursive quicksort function takes a provided array, and sorts the values of the array using a pivot point and partitioning. The pivot point is determined by taking the highest value of the array. The function partitions the array into two halves, with every value lower than the pivot point value being placed behind the pivot in the array, and every value above the pivot being placed in front. Once the function iterates through the array, it will call itself again to repeat the process, but this time passing in the partitioned arrays (array of values less than pivot and greater than). This process will repeat until all arrays are of size 1 (nothing left to partition), which at that point the array will be properly sorted. The non-recursive (iterative) version of the function follows the same logic, but instead of calling the

same function again to sort the partitions, it utilizes a stack to keep track of the high and low points for each partition, and pushes/pops from the stack as it iterates through each iteration. The function will continuously sort the array until the stack is empty, which indicates there are no more partitions to make (meaning the array is sorted).

Question 3

Source code can be found in **MatrixMath.java**

Sample output (codingground has trouble formatting the matrices)

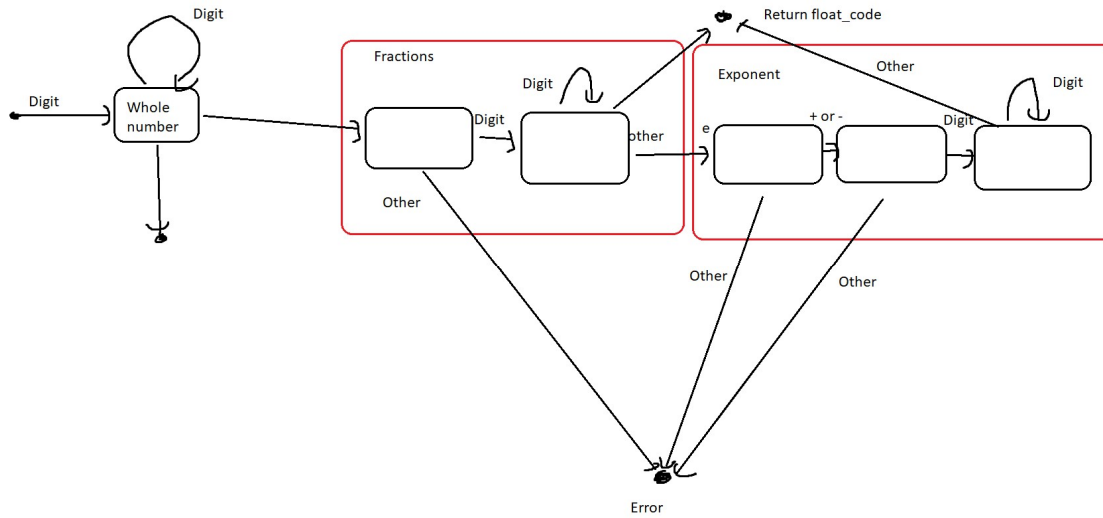
```
Array 1:
3 0 3 0
6 6 7 8
1 4 3 9
1 8 5 0 Array 2:
6 3 3 8
3 9 5 8 5 9 4 0 8 0 8 2
Final Array!
33 36 21 24 153 135 140 112
105 66 107 58 55 120 63 72
```

The code for this question implements 2 matrix multiplication functions in Java, following the convention rules of matrix multiplication (2 $N \times M$ matrices, where M_1 must equal N_2). Each function first checks if the matrix multiplication is valid, and throws an exception if it is not. Otherwise, it generates the product matrix by taking rows of the first matrix and the columns of the second matrix to define the final dimensions. From there, the function iterates through the rows and columns of the final matrix, and within the nested loops performs a third loop iterating through the length of the inner dimensions of the provided arrays. Within each

iteration, it multiplies the values of both matrices to perform the dot product of the current row and column, and once the 3rd iteration is complete the final dot product is saved onto the final result. This process is repeated until all rows and columns are done. The main difference between the two functions is one function utilizes for loops for it's iteration, while the other function uses while loops. The provided code hard codes 2 arrays of 4x4 dimensions with a fixed seed for random number generation.

Chapter 4

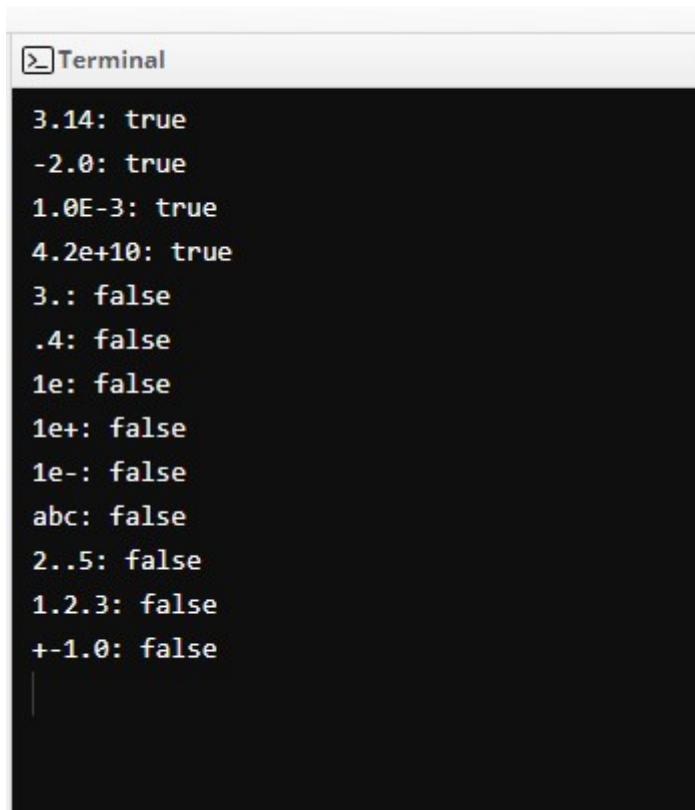
Question 2



Question 4

Source code can be found in `StateDiagram.java`

Sample output



```
Terminal
3.14: true
-2.0: true
1.0E-3: true
4.2e+10: true
3.: false
.4: false
1e: false
1e+: false
1e-: false
abc: false
2..5: false
1.2.3: false
+-1.0: false
```

The provided java code takes an array of various strings representing different possible “floats”. For each string, it parses each character in the string following the above state diagram, checking to see if the currently inspected character is valid or not. If it is, it will continue along the string until an invalid character is found or the entire string has been iterated. If an invalid character is found, it is marked as an error, which fails the comparison (in turn suggesting the string is not a valid float). If it can iterate through the whole string successfully, then the provided string is a valid float.

Question 6

Sample code can be found in **LexicalAnalyzer.java**

Incomplete, small error with conversion of EOF to java equivalent

Chapter 5

Question 1

Source code can be found in PerlScope.pl

Sample output

```
Global variable: 10

Static scoping:
Local variable: 5
Global variable: 20

Dynamic scoping:
Use of uninitialized value $local_var in concatenation (.) or string at main.pl line 27.
Local variable:
Global variable: 30

After function calls:
Global variable: 30
|
```

In the provided code, the sample code uses two variables and two functions to showcase static scoping and dynamic scoping (global_var and local_var, and static_scoping and dynamic_scoping respectively). In the static_scoping function, global_var is modified and its new value is shown in the function (5), and a new local variable is declared and printed. The local variable is defined with “local” keyword for perl to indicate it is a dynamic scope. In the dynamic_scoping function, the values of global_var and local_var are printed again. global_var is modified in this function, and its new value is shown properly. However, local_var fails to print because it is a dynamically scoped variable. This showcases how dynamic scoping is different in perl, as despite the variable being initialized earlier, it is not available in the scope of

dynamic_scoping. In Perl, dynamic scoped variables are still limited to the functions within their block, meaning once the program exits the static_scoping function, access to local_var is no longer available. The program ends by exiting the dynamic_scoping function and printing the value of global_var one more time, which maintains the value that was modified in dynamic_scope.

Question 3

Sample code can be found in Subprogram.js

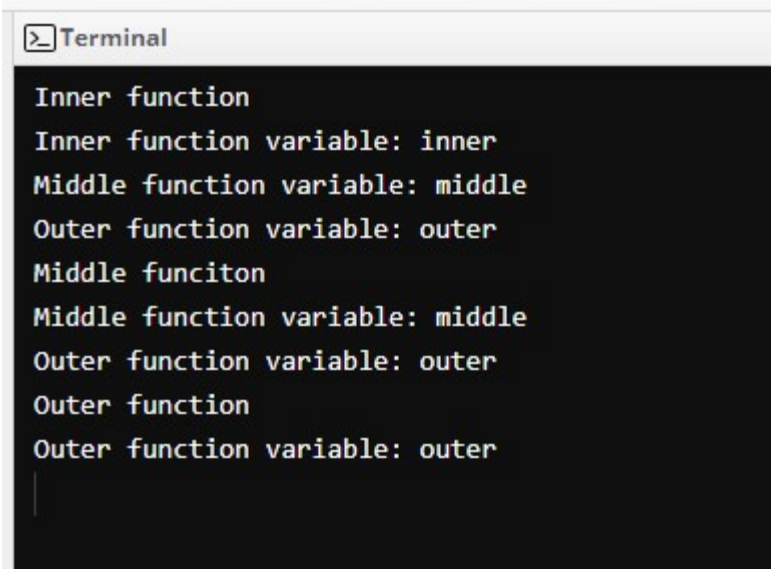
Sample output:

```
Outer function: outerVar = 1
Middle function: outerVar = 1, middleVar = 2
Inner function: outerVar = 1, middleVar = 2, innerVar = 3
|
```

This program simply creates three functions, with each function defining one variable and calling another function inside of it. outer defines one variable and calls middle, which defines another variable and calls inner, which defines a third variable. inner prints all 3 variables, showing how it can access the outer 2 variables even though it wasn't defined inside of inner.

Question 4

Sample code can be found in Subprogram.py



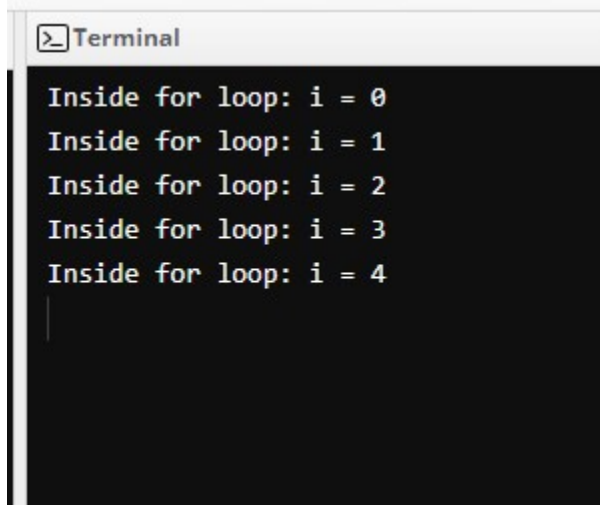
```
Terminal
Inner function
Inner function variable: inner
Middle function variable: middle
Outer function variable: outer
Middle function
Middle function variable: middle
Outer function variable: outer
Outer function
Outer function variable: outer
```

This program functions identical to the one above, but implemented in Python instead of JavaScript. It creates 3 functions, with each function calling a subsequent function and defining a variable, with the inner-most function printing the values of all 3 functions while the outer function only prints one value.

Question 6

Sample code can be found in **ScopeForLoop.java**, **ScopeForLoop.cs**, and **ScopeForLoop.cpp**

Sample output (java version shown only):

A terminal window titled "Terminal" with a dark background and light-colored text. It displays five lines of text, each representing an iteration of a for loop. The text is: "Inside for loop: i = 0", "Inside for loop: i = 1", "Inside for loop: i = 2", "Inside for loop: i = 3", and "Inside for loop: i = 4". There is a vertical cursor line on the left side of the terminal, positioned at the start of the fifth line.

```
Terminal
Inside for loop: i = 0
Inside for loop: i = 1
Inside for loop: i = 2
Inside for loop: i = 3
Inside for loop: i = 4
```

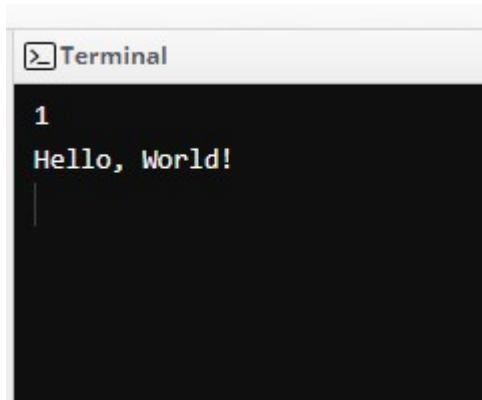
The following 3 code snippets all do the same thing: create a simple for-loop that prints the counter variable. The main thing of each program is that the counter variable, `i`, is attempted to be printed outside of the for-loop. For each program, the problematic line is commented out so the program may run, but if it is uncommented, the program will fail once it tries to print the value of `i` as it tries to access a variable that it can no longer use.

Chapter 6

Question 1

Sample code can be found in `CTest.c`

Sample output

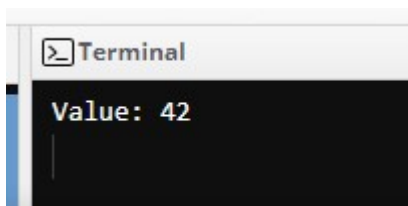
A terminal window titled "Terminal" with a dark background. It displays the output of a program: the number "1" on the first line and the text "Hello, World!" on the second line. A vertical cursor is visible on the third line.

```
1
Hello, World!
```

The provided code runs a set of simple tests to do things that are unideal for the C compiler, such as assigning a float to an int data type. Note that the online programming environment that is used for most of these questions do not show warnings the C compiler would generate, therefore it cannot be seen what the specific warnings are. However, the program still runs successfully and does not crash.

Question 2

Sample code can be found in `CFree.c`

A terminal window titled "Terminal" with a dark background. It displays the output of a program: the text "Value: 42" on the first line. A vertical cursor is visible on the second line.

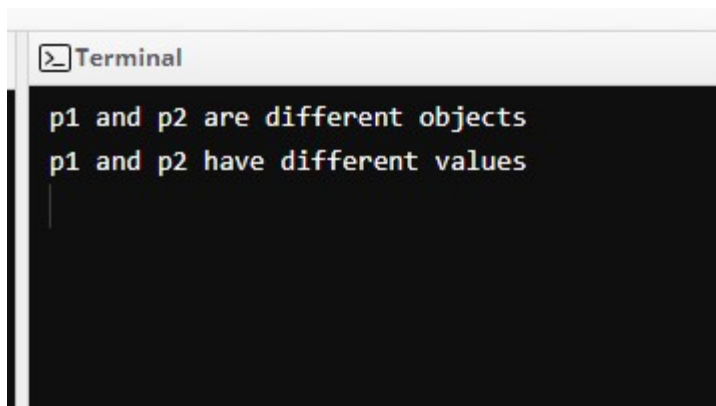
```
Value: 42
```

The provided code simply declares a pointer, assigns it a value of 42, prints the value, then frees the pointer. In this online programming environment, the `free()` function works fine, as the sample output shows the program ran without issue.

Question 9

Sample code can be found in `Equivalence.java`

Sample output

A screenshot of a terminal window with a title bar that says "Terminal". The terminal has a black background with white text. It displays two lines of output: "p1 and p2 are different objects" followed by "p1 and p2 have different values". There is a vertical line of white text on the left side of the terminal window, likely representing a scrollbar or a list of files.

```
p1 and p2 are different objects
p1 and p2 have different values
```

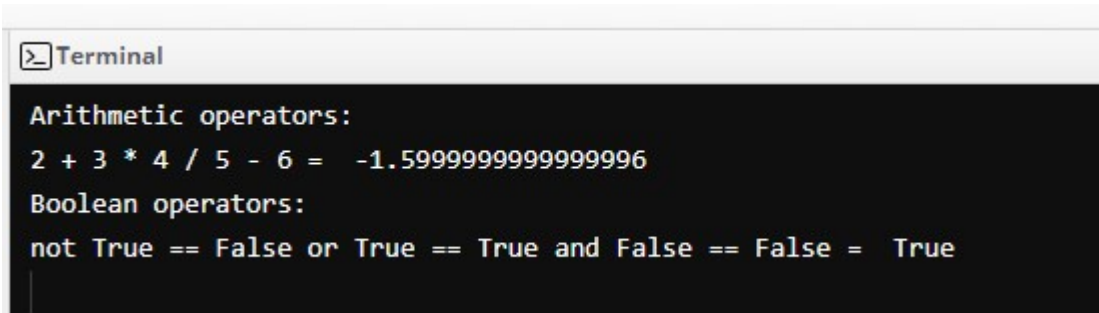
The provided Java code defines a `Point` class that simply contains two ints, and proceeds to create two objects of that class. It then proceeds to compare the two objects, first with the `'=='` operator and then with the `equals()` method in Java. In both tests, the two objects are considered different, which shows that this piece of Java code utilizes name equivalence.

Chapter 7

Question 3

Source code can be found in `Order.py`

Sample output:



```
Terminal
Arithmetic operators:
2 + 3 * 4 / 5 - 6 = -1.5999999999999996
Boolean operators:
not True == False or True == True and False == False = True
```

The following code shows the order of operations in Python for arithmetic and logic. For arithmetic, the following is done:

1. $3 * 4$, which equals 12
2. $12 / 5$, which equals 2.4
3. $2 + 2.4$, which gives 4.4
4. $4.4 - 6$, which gives -1.6

For logic, the following is done:

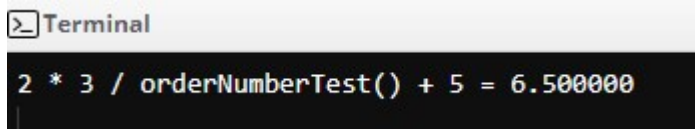
1. `not True` is evaluated to `False`
2. `False == False or True == True` is evaluated, which becomes `True`
3. `True and False == False` is evaluated, which becomes `True`

Note that in python, the associativity for both arithmetic and logic is to go left-to-right

Question 4

Sample code can be found in **Order.java**

Sample output:

A terminal window with a title bar that says "Terminal". The window has a dark background and shows the output of a Java program: `2 * 3 / orderNumberTest() + 5 = 6.500000`.

```
Terminal
2 * 3 / orderNumberTest() + 5 = 6.500000
```

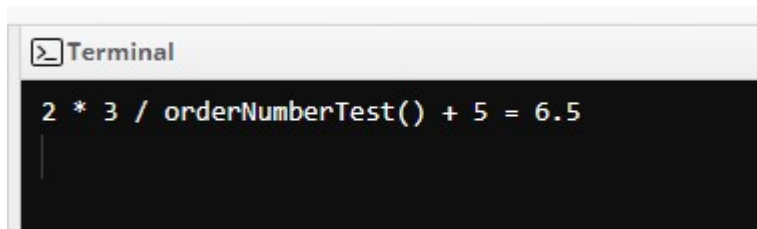
The following java code performs a simple operation, with the special thing of note being `orderNumberTest()`, which simply returns the number 4. The order of operations is as follows:

1. $2 * 3$, which is 6
2. $6 / 4$, which is 1.5
3. $1.5 + 5$, which is 6.5

Question 5

Sample code can be found in **Order.cpp**

Sample output:

A terminal window with a title bar that says "Terminal". The window has a dark background and shows the output of a C++ program: `2 * 3 / orderNumberTest() + 5 = 6.5`.

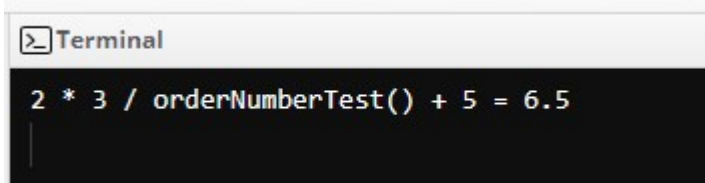
```
Terminal
2 * 3 / orderNumberTest() + 5 = 6.5
```

The following c++ code performs identical to the java code. The order of operations is identical as well.

Question 6

Sample code can be found in Order.cs

Sample output:

A terminal window with a title bar that says "Terminal". The window has a dark background and shows the text "2 * 3 / orderNumberTest() + 5 = 6.5" on the first line, followed by a vertical cursor on the second line.

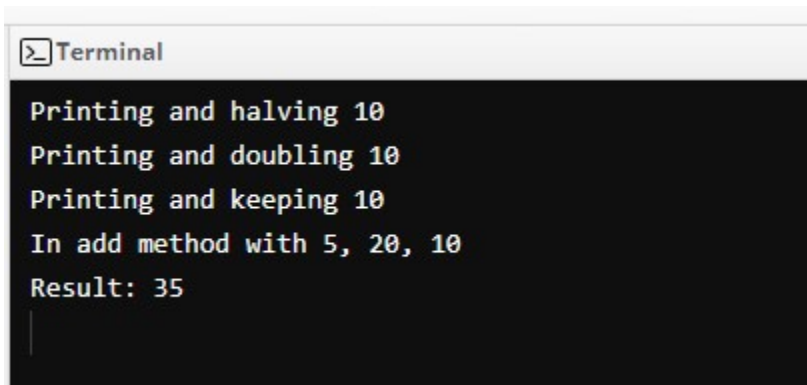
```
Terminal
2 * 3 / orderNumberTest() + 5 = 6.5
|
```

The following c# code performs identical to the c++/java code, and maintains the same order.

Question 7

Sample code can be found in Order2.java

Sample output

A terminal window with a title bar that says "Terminal". The window has a dark background and shows several lines of text: "Printing and halving 10", "Printing and doubling 10", "Printing and keeping 10", "In add method with 5, 20, 10", and "Result: 35". There is a vertical cursor on the line following "Result: 35".

```
Terminal
Printing and halving 10
Printing and doubling 10
Printing and keeping 10
In add method with 5, 20, 10
Result: 35
|
```

The following code calls an add() function which takes in 3 values: a return from printAndHalf (), a return from printAndDouble(), and a return from printAndKeep(). Each function takes in the value 10 and performs an operation based on the function name. As the

above output shows, the order of operation for expressions passed into a function is left-to-right, as `printAndHalf()` is ran first since it is the first argument in the `add()` function, and so on.

Chapter 8

Question 1

a.

```
// Setup k with provided math
int k = (j + 13) / 27;

// Begin while loop
while (true) {
    // Break out of loop when this expression is true
    if (k > 10) {
        break;
    }

    // Do more math with k and variable i
    k = k + 1;

    int i = 3 * k - 1;
}

// continue with the rest of the program after the "out" label
```

The following is a code snippet that implements the pseudocode in Java.

b.

```
k = (j + 13) // 27
while True:
    if k > 10:
        break
    k += 1
    i = 3 * k - 1
```

The following is a code snippet that implements the pseudocode in Python

c.

```
k = (j + 13) / 27
```

```
loop do
```

```
  break if k > 10
```

```
  k += 1
```

```
  i = 3 * k - 1
```

```
end
```

The following is a code snippet in Ruby

In this example, the code with the best readability would be Java, as it provides the clearest indication of what the code is doing with the variable types being labeled. For writeability, the Ruby code is superior as it offers the simplest syntax of the 3, making for the quickest and easiest to write. Python offers a middle ground between the two, with good writeability and some readability of the program.

Question 2

```

float j = 5;
float k = (j + 13) / 27;
while (true) {
    if (k > 10) {
        break;
    }
    k = k + 1.2f;
    float i = 3 * k - 1;
}
// rest of the code

```

```

k = (j + 13) / 27
while True:
    if k > 10:
        break
    k = k + 1.2
    i = 3 * k - 1
# rest of the code

```

```

k = (j + 13) / 27
loop do
    break if k > 10
    k = k + 1.2
    i = 3 * k - 1
end
# rest of the code

```

The following is Java, Python, and Ruby code (respectively) that rewrites the provided pseudocode into code snippets, but with the catch of converting variables into floats instead of

integers. In this example, Java still maintains the best readability, while Ruby and Python are similar in writeability.

Question 3

```
if (k == 1 || k == 2) {  
    j = 2 * k - 1;  
} else if (k == 3 || k == 5) {  
    j = 3 * k + 1;  
} else if (k == 4) {  
    j = 4 * k - 1;  
} else if (k == 6 || k == 7 || k == 8) {  
    j = k - 2;  
}
```

```
if k == 1 or k == 2:  
    j = 2 * k - 1  
elif k == 3 or k == 5:  
    j = 3 * k + 1  
elif k == 4:  
    j = 4 * k - 1  
elif k == 6 or k == 7 or k == 8:  
    j = k - 2
```

```
if k == 1 || k == 2
    j = 2 * k - 1
elseif k == 3 || k == 5
    j = 3 * k + 1
elseif k == 4
    j = 4 * k - 1
elseif k == 6 || k == 7 || k == 8
    j = k - 2
end
```

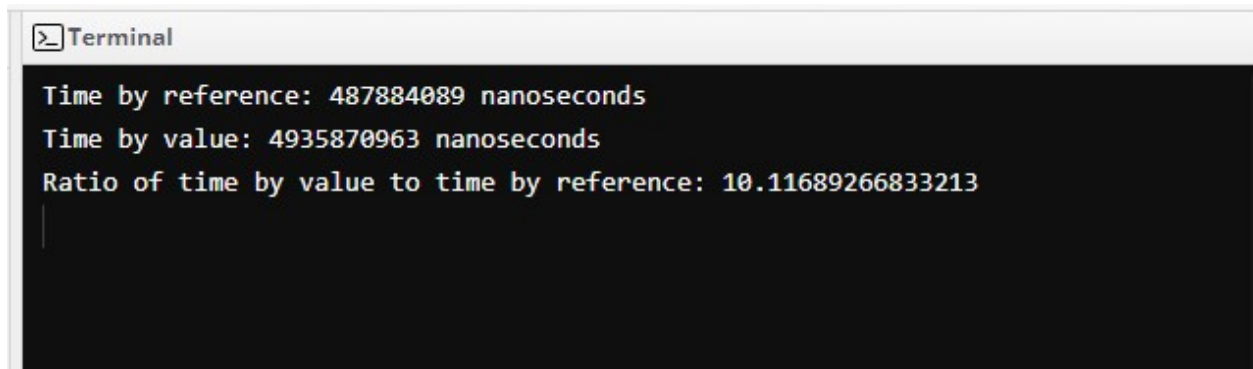
The following is a new code snippet that showcases multiple-selection statements in Java, Python, and Ruby, respectively. For this specific code, Python offers the best writeability as it has the shortest keywords needed for condition checks, but Ruby is not that far behind. Java offers the worse writeability of the 3, but the best readability as it is more clear what each statement is checking.

Chapter 9

Question 1

Sample code can be found in `ArrayTiming.java`

Sample output:

A terminal window titled "Terminal" with a dark background and light green text. It displays the following output:

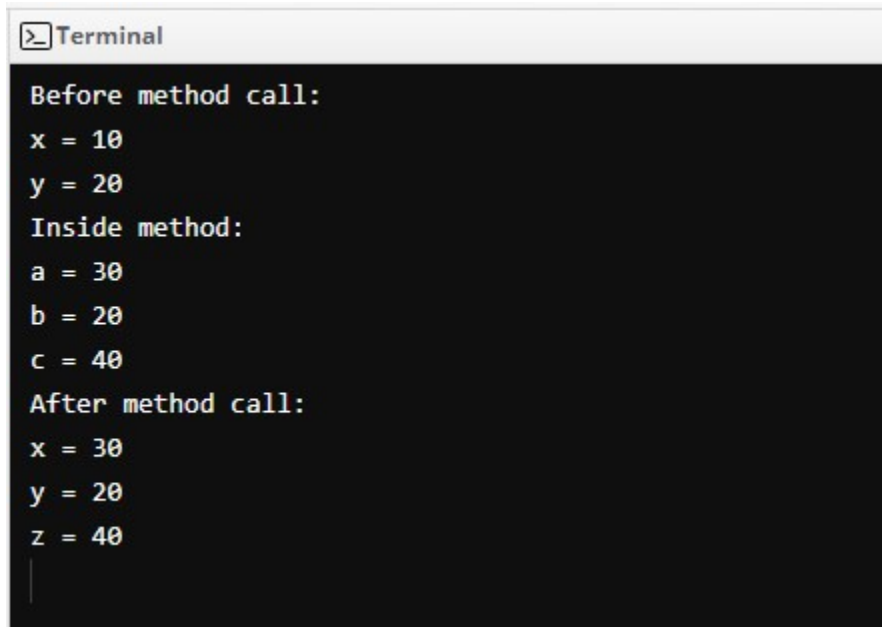
```
Time by reference: 487884089 nanoseconds  
Time by value: 4935870963 nanoseconds  
Ratio of time by value to time by reference: 10.11689266833213
```

The provided code creates a large array in Java filled with 1's, and runs a time experiment to see how long it would take to modify the array by passing it to a function that modifies it, both by reference and by value. The modifying function simply adds 1 to each number in the array. The pass by reference function simply takes the array, and passes the object to the modify function. The pass by value function creates a new array object, and copies the value of every element in the original array to the new one, and then passes the new array object into the modify value. Both functions time how long this process takes in nanoseconds, starting from the first element until the last element is done. Finally, the results of the time are displayed, along with the ratio of time from pass by value to time from pass by reference. In general, pass by value is significantly longer due to the need of needing to copy the values from the array in the first place before being able to modify them.

Question 2

Sample code can be found in `OutTest.cs`

Sample output:

A terminal window titled "Terminal" with a dark background and light-colored text. The output shows the state of variables before, inside, and after a method call. Before the call, x is 10 and y is 20. Inside the method, a is 30, b is 20, and c is 40. After the call, x is 30, y is 20, and z is 40. The cursor is at the end of the last line.

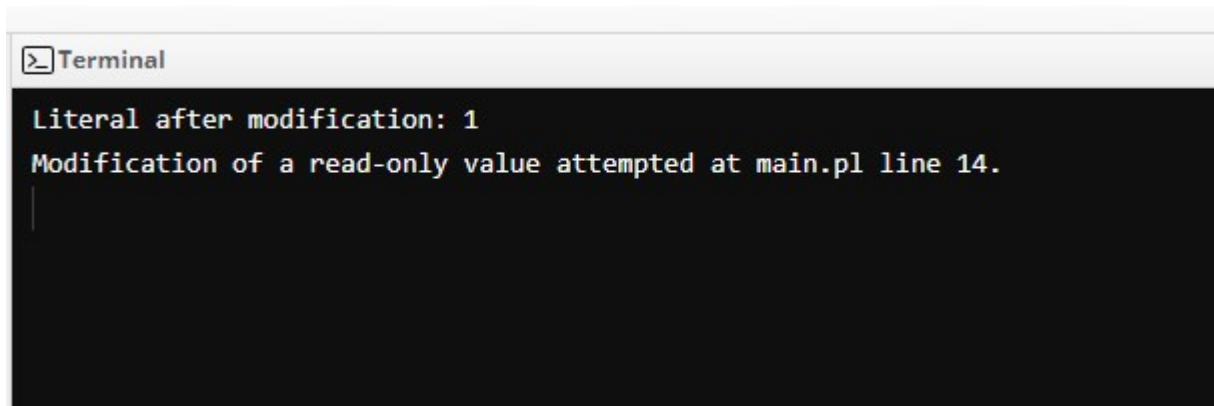
```
Terminal
Before method call:
x = 10
y = 20
Inside method:
a = 30
b = 20
c = 40
After method call:
x = 30
y = 20
z = 40
|
```

In the provided C# code, a set of variables is declared and their values are printed out. Then, a function called `TestMethod()` is called, which declares two parameters as `out` and takes in one value. The two parameters are then immediately defined with values, and the values are printed inside the function, before the function is completely and the final values are printed in main. As the above output shows, the updated values are retained leaving `TestMethod()` and display in main. The addresses of the `out` variables are determined at the time of the function call, since the values are changed inside of the function and those changes can be seen in the outside call after the function runs (the arguments to the function use the same addresses as the variables in the function).

Question 3

Sample code can be found in **PBR.pl**

Sample output:

A terminal window titled "Terminal" with a dark background. It displays two lines of text in a monospaced font: "Literal after modification: 1" and "Modification of a read-only value attempted at main.pl line 14." followed by a vertical cursor line.

```
Terminal  
Literal after modification: 1  
Modification of a read-only value attempted at main.pl line 14.  
|
```

The provided Perl code defines two subprograms, both of which take in a literal and attempt to modify it. The first subprogram modifies a numeric literal, while the second modifies a string literal. In the first subprogram, the modification result is printed, which only ends up being the modification itself (an attempt to add 1 to the number, but we only see one printed). In the second subprogram, we see an error pop up as it fails to modify the literal. This is because Perl always passes by reference, meaning modifying literal values can be inconsistent as whenever a literal is defined (or edited), Perl creates a new literal variable, meaning the original literal is still unchanged.

Question 4

Sample code can be found in **PBR.cs**

Sample output:

Terminal

```
main.cs(14,25): error CS1510: A ref or out argument must be an assignable variable  
Compilation failed: 1 error(s), 0 warnings
```

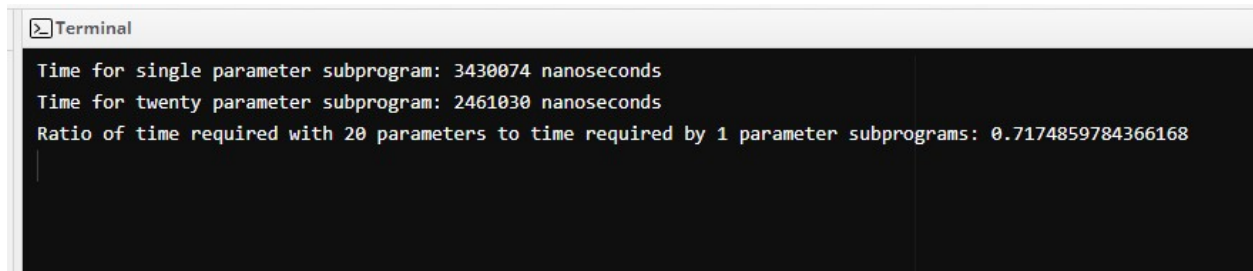
The provided C# code passes by reference a literal numeric of 5 to a function, before attempting to modify it and printing the result in the function. However, this fails to run as C# doesn't allow literals to be assigned by reference, as the literal doesn't have a memory place to reference (it's not a variable, so what's there to reference).

Chapter 10

Question 1

Sample code can be found in `ParameterTest.java`

Sample output:

A terminal window with a title bar that says "Terminal". The window has a dark background with light-colored text. The text inside the terminal shows the results of a performance test.

```
Time for single parameter subprogram: 3430074 nanoseconds  
Time for twenty parameter subprogram: 2461030 nanoseconds  
Ratio of time required with 20 parameters to time required by 1 parameter subprograms: 0.7174859784366168
```

In the following Java code, a test is performed in which two functions are called to perform an operation. The first function takes in 1 random number and applies a small operation to it (doubles it). The second function takes in 20 random numbers and adds them all together. The time for both functions to run is tracked, and the final result is shown in the output above. In this implementation, the single parameter function takes longer than the twenty-parameter function.

Chapter 11

Question 6

Sample code can be found in Queue10Name.java

The provided class creates an object that represents a queue that takes in 10-character strings represented as a character array. The class consists of a Node private object, which in turn is a custom class that tracks elements inside our queue. The node class is what tracks the 10-character names, and also has a variable to track what's next to it. In the Queue10Name class, it contains two variables which track the front and back of the queue (two Node variables). enqueue() is the first function and takes in a character array, and creates a new node and adds it to the back of the queue, and updates the rear variable to point to the newly created node. dequeue() is the second function and removes the front-most element from the queue, and updates the front variable to be the new front node. Finally, empty() clears the entire queue, by nulling both front and rear.

Question 7

Sample code can be found in QueuePrimitive.java

The provided class creates an object that represents a queue that can take in primitive types. It works by using a generic operator <T> to allow for the acceptance of primitives. It contains a field that is an array of N generic types, and a value to keep track of the front-most index. The constructor takes in a size argument, which defines how big the array that the queue uses will be (and thus the max size of the queue). Like before, the queue has an enqueue(),

dequeue(), and empty() function which all serve the same purpose, with a slightly different implementation. The major distinction is that instead of Node objects, we use the front index value to shift elements in the array. If the index value is -1, this represents the array is empty.

Question 8

Sample code can be found in Queue20Name.java

The provided java code is like the one in Question 6. The main difference is that this queue can hold character arrays of size 20, and (more importantly) uses a priority system within the queue. The constructor has been updated to take in an integer value that represents the items priority. The higher the number, the higher the priority. The dequeue() function has been updated to handle this priority system, iterating through the whole queue in order to find the node with the highest priority in order to be removed. In the event of a tie, the standard rules of a queue apply (FIFO).

Chapter 12

Question 1

Sample code can be found in `LinkedListRewrite.java`

In terms of readability, the C++ is more explicit with what each class relation to one-another, and is especially easy to understand when a function that is called is actually a function that belongs to the parent of the class. In contrast, the Java code is easier to program as the simpler syntax requires less for the programmer to remember, and does not need to constantly type out the parent class.

Question 2

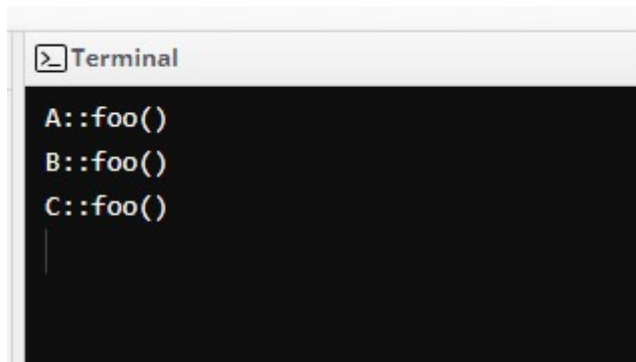
Sample code can be found in `LinkedListRewrite.rb`

In terms of readability and ease of programming, the Ruby code is worse than both the C++ and Java implementations. It is not clear at a glance the contents of the different classes, and it's syntax is more complicated to write than C++ or Java.

Question 3

Sample code can be found in `Inheritance.cpp`

Sample output:

A terminal window with a title bar that says "Terminal". The window has a black background with white text. The text inside the terminal is:

```
A::foo()
B::foo()
C::foo()
|
```

The vertical bar at the end of the third line indicates the cursor is positioned there.

The following C++ code creates 3 class (A, B, C) which all contain a `foo()` function that simply prints the class name and function. A 4th test class is created which runs the test for the 3 classes. The highlight is that in the test class, only one object is used; an object made with the A class. First, the `foo()` function is called with the A class, which represents static binding of `foo()` as A class using an A function is determined in compile time. Then, the object is deleted and redefined as an object of class B (we can do this since B extends A). From there, `foo()` is called again but this time the B class's implementation of `foo()` is used. This represents dynamic binding as `foo()` from class B is not determined until runtime, as the a object was originally defined as a class A object and wasn't considered class B until the redefinition. The same process is repeated with class C, showing dynamic binding yet again.

Chapter 13

Question 6

Sample code can be found in `ReaderWriter.java`

Sample output:

 Terminal

```
Note: ReaderWriter.java uses or overrides a deprecated API.  
Note: Recompile with -Xlint:deprecation for details.  
Reader 37 read shared data: 1  
Reader 43 read shared data: 1  
Reader 46 read shared data: 1  
Reader 32 read shared data: 1  
Writer 36 wrote to shared data: 1  
Reader 45 read shared data: 1  
Reader 56 read shared data: 1  
Reader 44 read shared data: 1  
Writer 34 wrote to shared data: 2  
Writer 33 wrote to shared data: 3  
Writer 35 wrote to shared data: 4  
Writer 38 wrote to shared data: 5  
Writer 39 wrote to shared data: 6  
Writer 40 wrote to shared data: 7  
Writer 41 wrote to shared data: 8  
Writer 42 wrote to shared data: 9  
Writer 47 wrote to shared data: 10  
Writer 48 wrote to shared data: 11  
Writer 49 wrote to shared data: 12  
Writer 50 wrote to shared data: 13  
Writer 51 wrote to shared data: 14  
Writer 52 wrote to shared data: 15  
Writer 53 wrote to shared data: 16  
Writer 54 wrote to shared data: 17  
Writer 55 wrote to shared data: 18
```

The provided Java code implements a basic producer-consumer (or reader-writer in this case) problem. In this case, a ReaderWriter class is created, which consists of a run function that randomly decides between becoming a reader or a writer. The class contains a static shareddata variable that all tasks use to reference for reading or writing, along with a semaphore used to restricting writing access. 25 tasks are made of the ReaderWriter class, and each task randomly decides to be a reader or writer. If a reader, the task simply reads the value of shareddata. If a writer, the task first attempts to acquire the writeSemaphore; until it can acquire it, it won't do anything. Once acquired, the task will write to the shareddata variable (add one) and then immediately releases the semaphore. There is only 1 semaphore in this program, meaning only one program may write at a given time. However, there is no restriction for the number of readers. This is why in the sample output all readers print the same value, as they all read at the same time so will read whatever value was available at that moment.