Andrew Parsons COMP 320 A. Brady 3 April 2018

PROBLEM SET 1

Chapters 1 and 2. Due Monday, 9 April 2018.

1. *Read over questions 6, 7, and 8 from the Problem Set questions at the end of Chapter 1. Choose one of them to answer. (3 pts)*

- 7: *Java uses a right brace to mark the end of all compound statements. What are the arguments for and against this design?*This is covered in syntax design addressed on page 12. In that section the author asserts that simplicity and readability are sometimes at odds with one another.The C family of languages uses `{ ... }` to encapsulate many statements. This is fairly simple, but readability is both improved and damaged in using a right brace to mark the end of a compound statement.A primary but weak supporting argument for using the right brace instead of an `end` or similar convention is an acknowledgment of legacy. Because the C family of languages uses this brace syntax, readability is improved for new Java programmers with C-family familiarity. This however is only advantageous when a new Java programmer is comfortable with this syntax in the first place.Ignoring this gentle nod to predecessors and relatives, Java's ending brace adds another character, and depending on convention, another line to code. This makes the code challenging to read because exactly which compound statement the right brace ends is obscured. Consider the following code snipet:`// some Java here`

```
      }
    }
// some more here
  }
}
}
```

  Identifying exactly which right brace corresponds with is--from experience--quite taxing when reading and writing many nested compound statements. Another point to consider is that different types of compound statements all use the same encapsulation syntax in Java. This means that a conditional `if` and a loop `while` will both terminate with `}`. This is advantageous when considering writability, as a programmer needs only to remember that all compounds end uniformly instead of trying to recall the compound-specific endings. A simple trick for deeply nested code blocks is using comments to denote for which each ending brace stands. `// some Java here`

```
      } // end if-statement
    } // end for-loop
// some more here
  }
} // end while-loop
}
```

This however is laborious and just adds text to a code block, decreasing legibility. A more elegant solution is noted on page 12, where Fortran 95 and Ada use `end if` and `end loop`. Such an implementation would look something like the following: `// some code here`

```
      END IF
    END FOR
// some more here
  END WHILE
END
```

This is clearly more legibile, however rather laborious to write. Additionally, if Java were to implement the above, the compiler would need to reserve quite a few more words.

1. *Answer Problem Set question 10 from the end of Chapter 1. (4 pts)*

- What are the arguments for writing efficient programs even though hardware is relatively inexpensive?

Though hardware is relatively inexpensive, there is an upper boundary for the amount of hardware that can be allocated to solve a given problem. To lessen the need for immense amounts of hardware, programs ought to be efficient. Hardware, no matter how cheap, is not free. We as humans largely value speed because it leads to increased production, and so why write slower programs when there is an opportunity to write faster ones?

I think efficiency is also very related to the aforementioned criteria, readability, writability, and reliability. An inefficiently written program will be hard to read and maintain (though it may be easier to write).

1. *Choose one of Problem Set questions 11, 12, 15, 17, or 18 from the end of Chapter 1 and answer it. (3 pts)*

#15. *How do type declaration statements for simple variables affect the readability of a language, considering that some languages do not require them?*

Type declaration statements for simple variables improve readability as they make clear of what datatype variables are.

I think the worth of type declarations really shines when declaring strings: quotation marks are easy to miss, but when the variable type is explicity written, it is clear that what follows is not an integer, boolean, etc., but most definitely a string.

Considering the following statements (ignoring the awful variable names):

```
int a = 6
String b  = "6"

a = 6
b = "6"
```

From the first two statements it is clear that `a` is an integer with a value of six and `b` is a string holding the character `6`. This is not immediately clear with the second two statements. A reader might, while skimming, assume that both `a` and `b` are integers since their values are set to `6`. Wile it is easy to discern the quotation marks denoting the string, those might be less than visible should these statements be burried and deeply nested among dozens of other lines. Sure, modern text editors will recognize a variable's type and provide the correct coloring, thereby improving readability, but that is a crutch and hardly a case for removing explicit type declarations.

An argument against using type declarations is that they clutter code and type declaration can be handled during compilation or interpretation. However, I think the ease of variable recognition by the reader outweighs a desire to declutter the code.

1. *Answer Problem Set question 16 from the end of Chapter 1. (12 pts)*

- *Write an evaluation of some programming language you know, using the criteria described in this chapter.*

The feel most comfortable with Java.

Readability

- Sebesta gives an example of where Java fails to be simple, and therefore is difficult to read, on page 8. The author points to feature multiplicity as a complicating characteristic of Java, as shown below.`count = count + 1`

  `count += 1`

  `count++`

  `++count`
- Java's operators are fairly readable considering there is only one overloaded operator (that I know of), `+`: it handles all numbers (floats, ints) and string concatenation. This forces writers to use many statements to preform complex arithmetic operations, but this is easier for readers to understand, since they can follow steps.
- There are a limited number of control statements.
  - if-then, if-then-else, switch
  - for, while, do-while

    - break, continue, return
- Readers must know only about ten control flow statements and mustn't learn several dozen. Lastly, because Java requires statically and strongly typed type declarations, and has an appropriate number of data types available (like boolean), programmers can be clear in what they write. This improves readability. However, Java's program structure and syntax can make it difficult to read. First, programs or classes start with a number of declarations using keywords which, while important, can confuse beginners. This problem is exasperated by the braces used to group compound statements, as discussed in question 7. Waterfalls of `}` make readability difficult, since readers must keep track of brace pairs. I can't speak to orthogonality.

Writability

Java requires some syntactical overhead. Programs open with

```
public class ClassName {

  public static void main(String args[]) {

  }
}
```

before anything else is written. This is an oversimplification, but enough to mention.

Because Java has a finite number of building blocks, programmers must often build their own constructs to achieve a goal. This gives programmers great flexibility while keeping the language relatively simple.

As mentioned on page 13, Java includes `for` loops, which are fairly intuitive and from my experience simpler to write than a `while`.

Reliability

Java is reliable because it is translated to the portable byte code which can run on JVMs on nearly every platform.

Java also features extensive type checkking and exception handling unlike the first version of C. Java's type checker runs at compile-time, removing the chance that there will be type errors during run time. Programmers can have a higher degree of confidence that their program will reliably run as expected if it properly compiles.

1. *Answer questions 1, 6, and 7 from the Problem Set at the end of Chapter 2. (3 pts each, except #6, which is worth 2 pts)*

#1. *What features of Plankalkül do you think would have had the greatest influence on Fortran 0 if the Fortran designers had been familiar with Plankalkül?*

Had the designers of Fortran been able to refer to Plankalkül's design, I think they would have tried to implement the following features:

- data typing: that Fortran initially didn't have data typing seems quite limiting, though it must have been a deliberate decision given the constraints. However, Zuse's Z4 was also faced with constraints yet Plankalkül somehow had data types.
- records/structs: the original versions of Fortran didn't have the records that Zuse included with Plankalkül. These were added in later versions of Fortran.
- Zuse's programs, specifically for syntax checking: though the programs aren't language features, analyzing the algorithms may have proven useful for language design.

#6. *Make an educated guess as to the most common syntax error in Lisp programs.*

One too many, one too few, or incorrectly placed parentheses.

#7. *Lisp began as a pure functional language but gradually acquired more and more imperative features. Why?*

An inherent feature of functional languages is that they have no state. Though Lisp is functional in that it is organized around expressions instead of statements, some functions have side effects like storing data in variables or array positions, meaning Lisp modifies a state. These features were necessary as they allowed for computation and the output of that computation.

The development of Common Lisp merged the various dialects of Lisp, of which several had developed additional imperative features.