CSE 320 Review

Sample final review

[1] Some languages (e.g. Smalltalk) allow you to redefine keywords. This allows you to change

the behaviour of keywords like WHILE and operators like \*. This can result in shorter programs

for special cases, enhancing writeability. Does this feature enhance readability? Explain.

No it does not increase readability

1. All the references and syntax you have learned about the language can be changed. For example a for loop can be modified to take a fractional index which will require you to change what you know about for loops
2. Any other programmer will need to figure out how the original programmer wrote the program. In fact it may not even compile without the original modifications.

[2] In general, shorter programs take less time to write, so a very concise programming language

tends to increase programmer productivity. What disadvantage would you expect to find in such

a language?

It would be hard to read. The exact usage of the language would require a new programmer to know everything about the language just to read the program.

As well it would be less expressive then a language like C++; restrict how programs can be made.

[3] C++ has many ways of expressing any algorithm, Pascal restricts the possible was to code anything.

Which language is safer? Which language is more expressive? Which style do you prefer?

Pascal is safer, because it restricts the programmer to its style of programming. C++ is more expressive because you have freedom to write how the program how you see fit without the language restricting you. I prefer the freedom of C++; allow me to make my own mistakes.

[4] The following are two different Fibonacci number functions, both in C++.

Assume we are calculating Fibonacci number 45.

int fib1(int N) // returns the Nth Fibonacci number {

if( N <= 2 ) return 1;

else return fib1(N-1)+fib1(N-2); }

int fib2(int N) // returns the Nth Fibonacci number

{int a,b; // previous 2 numbers

int i; // loop index

int pt=0; // if 0, update a else update b with latest fib. calculated

int fib; // current Fibonacci value

a=1; b=1; // initial values for fib(1) and fib(2)

for(i=3; i<=N; i++){

fib = a + b;

if( pt==0 ) { a=fib; pt=1; }

else { b=fib; pt=0; } } return fib; }

1. Which uses more memory - fib1 or fib2? Why?

Fib1 uses more memory because it puts more onto the stack because of the nature of recursion.

b) Which do you find easier to understand? Why?

Fib1 is easier to understand, as long as, you are comfortable with recursion. Because it does not require a for loop and a complicated if statement to work.

c. which program runs faster?

Fib2 would run faster because it is able to run in linear time. While fib1 would be running in exponential time instead.

[5] How is a stack used in calling subprograms?

A stack is the memory for the ordering of the subprograms. For example in a recursion program, each iteration of a recursive call would create another subprogram on the stack. It will continue to add onto the stack until the last one is returned, then it would work down the stack until it returns the original call.

You would put the return line of the code at the bottom of the stack, then you would add the parameters you are passing, and any work space needed for the program to function. Once it returns it would return to that next line of code at the bottom of the stack with the require returned parameters.

[6] Do you think Prolog is Turing-complete? (Note that this was left open in class,

I will accept either alternative as correct). Give 2 reasons why you

think it is or isn't.

Prolog is turing complete because prolog includes arithmetic and first order logic.

Or

Prolog is not turing complete because it does not include all first order logic. As well as arithmetic is not completed in the language either.

[Extra]

Values in the predicate calculus are either T or F, and we have logical operators

'and', 'or', 'not' and 'implies'. 'not' is unary (has a single argument), all the

others are binary operators. How could we show:

[a] No additional binary or unary operators are necessary.

Unary is not needed because it would only flip a bit. If you don’t need to flip the bit then do nothing. If you need to flip it then you use not.

Binary is not needed because you can prove that using the existing operators, you would be able to create any possible output from those original operators.

[b] No higher-order operators (that is, operators that combine 3 or more values)

are necessary.

If the operators would not be needed to simultaneously return a value then no higher order operators are needed. However if the system can not have the binary operators calculating things in steps then yes higher order operators are needed.

**Properties of languages**

Syntax - is the set of rules that defines the combinations of symbols that are considered to be a correctly structured document or fragment in that language.

Semantics - of language constructs, as opposed to their form ([syntax](https://en.wikipedia.org/wiki/Syntax_(logic))). According to Euzenat, semantics "provides the rules for interpreting the syntax which do not provide the meaning directly but constrains the possible interpretations of what is declared

Type Systems - defines how a programming language classifies values and expressions into *types*, how it can manipulate those types and how they interact. The goal of a type system is to verify and usually enforce a certain level of correctness in programs written in that language by detecting certain incorrect operations.

**Types of languages**

**Imperative** – An imperative language uses a sequence of statements to determine how to reach a certain goal. These statements are said to change the state of the program as each one is executed in turn.

**OO** – is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) based on the concept of "[objects](https://en.wikipedia.org/wiki/Object_(computer_science))", which may contain [data](https://en.wikipedia.org/wiki/Data), in the form of [fields](https://en.wikipedia.org/wiki/Field_(computer_science)), often known as *attributes;* and code, in the form of procedures, often known as [*methods*](https://en.wikipedia.org/wiki/Method_(computer_science))*.* A feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "[this](https://en.wikipedia.org/wiki/This_(computer_programming))" or "self").

**Functional** – is a [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm)—a style of building the structure and elements of [computer programs](https://en.wikipedia.org/wiki/Computer_program)—that treats [computation](https://en.wikipedia.org/wiki/Computation) as the evaluation of [mathematical functions](https://en.wikipedia.org/wiki/Function_(mathematics)) and avoids changing-[state](https://en.wikipedia.org/wiki/Program_state) and [mutable](https://en.wikipedia.org/wiki/Immutable_object) data. It is a [declarative programming](https://en.wikipedia.org/wiki/Declarative_programming) paradigm, which means programming is done with [expressions](https://en.wikipedia.org/wiki/Expression_(computer_science)) or declarations[[1]](https://en.wikipedia.org/wiki/Functional_programming#cite_note-expression_style-1) instead of [statements](https://en.wikipedia.org/wiki/Statement_(computer_science))

**Logic** - is a type of [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm) which is largely based on [formal logic](https://en.wikipedia.org/wiki/Formal_logic). Any program written in a logic [programming language](https://en.wikipedia.org/wiki/Programming_language) is a set of sentences in logical form, expressing facts and rules about some problem domain.

**Readability -** is high correlation between an idea’s visual intensity in code and its marginal importance to the understanding of what the code does.

In highly-readable languages, key concepts are more visually intense in the code, while less important ones don’t even show.

**Writability -** that it is easy and fast to create programs in that language because the language construct has minimal symbols which does not require many statements and focuses on simplification of code (concise).

**Expressivity** – is the breadth of ideas that can be represented and communicated in that language. The more expressive a language is, the greater the variety and quantity of ideas it can be used to represent.