Homework 3
Programming Languages
CS 320

Stephen Giang RedID: 823184070

Problem 1: What advantages does static type checking have over dynamic type checking? (Ch. 6)

The advantage of static type checking is that static type checking is more efficient. That is because it runs at compile time instead of runtime. Because of this compile time checking, static type checking can type check for all executions, thus it doesn't have to type check every time the program is executed. With dynamic type checking, type checking has at runtime which will actually slow down the program as it forces the machine to type check for every execution.

Problem 2: Consider the integer expression $A + B + C$. Suppose the values of A, B, and C are 20,000, 25,000, and -20,000, respectively, and the machine has a maximum integer value of 32,767. Explain difference of left-to-right evaluation and right-to-left evaluation. Show it, yes, but also include an explanation. (Ch. 7)

Left to right evaluation would read this expression as $A+B+C = (20,000+25,000)+-20,000 = 45,000 + -20,000$. This will lead to an error because by adding A and B first, we get an integer out of range. Notice how $45,000 > 32,767 = \text{maxInteger}$.

Right to left evaluation would read this expression as $C+B+A = (-20,000+25,000)+20,000 = 5,000+20,000 = 25,000$. This has no problem doing because all the values stay within the integer value range.

The difference between left to right evalution and right to left evalution is simply how the machine reads the operands.

A left to right evaluation machine will read this $A+B+C$ as $(A \to + \to B) \to + \to C = (A+B)+C$.

A right to left evaluation machine will read this $A + B + C$ as $(C \to + \to B) \to + \to A = (C + B) + A$.

Problem 3: What are arguments for and against a user program building additional definitions for existing operators, as in C++? (Ch. 9)

Arguments for a user program building additional definitions for existing operators is that it allows the user to dictate their own form of operators. It can be really useful for doing operations on user defined types. For example, a user can define a type city that holds its name and location. Then using an additional definition for the subtraction operator, $city_1 - city_2$ could equal the distance from each other.

Arguments against a user program building additional definitions for existing operators is that it reduces readability. For example, it isn't indicative that subtracting 2 cities will give u the distance between the two. It makes it harder for other programmers to read it and sometimes doesn't make all that much sense in the bigger picture.

Problem 4: 4. (Also Ch. 9) For each parameter passing method listed below, show all of the values of value and list after each of the three calls to swap. Assume each call begins with $val = 2$ and $list = \{1, 3, 5, 7, 9\}$.

```
void swap(int a, int b) {
  int temp;
  temp = a;
  a = b;
  b = temp;
}
```

Parameter passing method	Method call	Value of *val* after call	Value of *list* after call
pass by value	swap($value, list[0]$)	2	$\{1, 3, 5, 7, 9\}$
	swap($list[0], list[1]$)	1	$\{1, 3, 5, 7, 9\}$
	swap($value, list[value]$)	2	$\{1, 3, 5, 7, 9\}$
pass by reference	swap($value, list[0]$)	1	$\{2, 3, 5, 7, 9\}$
	swap($list[0], list[1]$)	2	$\{3, 1, 5, 7, 9\}$
	swap($value, list[value]$)	5	$\{1, 3, 2, 7, 9\}$
pass by value-result	swap($value, list[0]$)	1	$\{2, 3, 5, 7, 9\}$
	swap($list[0], list[1]$)	2	$\{3, 1, 5, 7, 9\}$
	swap($value, list[value]$)	5	$\{1, 3, 2, 7, 9\}$

Problem 5: What dangers are avoided in Java by having implicit garbage collection, relative to C++? (Ch. 11)

In Java, having implicit garbage collection is that an object will be deallocated and sent for garbage collection automatically once it has reached the end of its scope. When an object reaches the end of its scope, it is no longer needed, so it is sent to garbage collection, thus freeing up memory. Whereas in C++, once an object has reached its scope, it takes explicit destruction for the object to be sent to garbage collection. The danger avoided here is if this is not done, the program will have a lot of memory being unused. This will make ones programs more slow and less optimized.

Problem 6: Create a 1-dimensional Fortran array B of 300 real numbers, indexed from -149 to +150.

REAL, DIMENSION(-149:150) :: B