CSCU9Y4 Spring 2018

Programming Language Paradigms

Tutorial 18/1

What's expected of students for CSCU9Y4 tutorials?

You should complete the work before the tutorial session: the tutorial class itself is **really to discuss your answers**. If you get stuck at a question, have a look at the course text books or do some searching on the Internet. If you're still stuck, move on to the next question; they are often independent. You might also consider working together in study groups to go through tutorial questions. Often in CSCU9Y4 there is no "right answer": instead we're looking for your ability to argue a particular position.

Solutions for tutorial questions are only available through attendance and participation in tutorial classes.

Reading for this tutorial: Chapter 1 of Bansal: Introduction to Programming Languages. Alternatively, Chapter 1 of Clark and Wilson: Comparative Programming Languages with Chapter 2 for further context, Chapter 1 and 3 (definite loops) of Watt: Programming Language Design Concepts, or Chapter 1, 2 and 7.4 of Sebesta: Concepts of Programming Languages. Y4_supplementary_notes_1 will also help.

If you understand the principles of languages you should be able to guess or work out the meaning of constructs in multiple languages. The first five questions explore different approaches to the design of a single language construct in a variety of languages. You will need to think about how programming constructs are implemented and how data is represented internally to get to the bottom of these examples.

1. Consider the following loop in C++:

```
sum = 0.0;
for (double i = 0.1; i <= 0.9; i += 0.2)
   sum = sum + i;
```

What is the effect of this code? How many times will you go round this loop? How might the answer be implementation dependent (i.e. different on different compilers or hardware)?

2. Pascal does not allow the control variable in a **for** loop to be a floating point variable and, with integers, the increment must be either +1 or –1. Are these two restrictions sensible? (Consider, when would you need to write a loop with a floating point variable, or with a different increment, and could you do this in a different way within the Pascal constraints?)

3. Consider these two loops (Java, or C++):

```
double total = 0.0;
for (double x = 1.0; total < 100.0; x += 1.0)
   total += x;

double total = 0.0; double x = 1.0;
while (total < 100.0) {
    total += x;
    x += 1.0;
}
```

continued over …

What result do you get with each loop? Are they the same? If so, which do you prefer? If not, how would you restrict the language to prevent misunderstandings?

4. What happens to the loop control variable after execution of the following Java loop? What is its value (if any)?

```
sum = 0;
for (int i = 1; i <=10; i++)
   sum += i;
```

5. In Pascal, after execution of the loop:

```
var i,sum: int;
sum := 0;
for i := 1 to 10 do
   sum := sum + i
```

the value of the loop control variable `i` may be used and its value is unknown. Why might that be? After the equivalent Basic program, the value of `i` would be 11. Which is preferable? Why?

6. Some languages use keywords to try to indicate meaning more clearly. Consider the following Ada loop (same as the Java loop of Q4) and the Algol loop (same as Q1).

```
sum := 0;
for i in 1 .. 10 loop
    sum := sum + i;
end loop;
```

```
sum := 0.0;
for i := 0.1 step 0.2 until 0.9 do
    sum := sum + i
```

Which form do you think is clearer (Java, Pascal, Basic, Ada)?

6. Two aims in the design of a programming language might be:

a) It should allow the programmer to express solutions in as convenient a way as possible.
b) It should be straightforward for the language developer to implement the compiler (or interpreter) on a computer using well-tried software technology.

In what ways are these aims in conflict? Which consideration, in your opinion, is the more important? What makes a "good" programming language?