Github Link: <https://github.com/jakdax99rb/Programming-Language-Concepts/tree/master/Test%202>

1. For the two functions(q1\_enum and q1\_int) in Test 2, in file test5.cpp I would argue that the enum version is more easily readable because the enum values cannot be changed, so at a glace you can always tell the outcome of a section of code without needing to check the rest to see if that variable was changed elsewhere. Code in Test5.cpp
2. Perl uses static scoping however it allows variables to be explicitly given a dynamic scope using the local operator. Code in Q2.pl
3. Heap took the longest to iterate through 200000 creations at .048 seconds, I think this is because of the use of the new operator causing more steps in the process. The other 2 took the same amount of time as each other at less than .0006 seconds. You cant do this in java because all arrays in java are fixed heap-dynamic array. Code in Test5.cpp
4. Q4 Is located in the test2Q4.java file. I only converted the specifically mentioned functions so there will be uninitialized variable errors.
5. Sum1 is 46 and sum2 is 48. The difference is caused by sum2 directly modifying the value at the memory address before the variable is used by the second half of the equation. The first sum modifies it after using the original value. If there were no precedence rules then for sum1 would end up with 10 / 2 = 5 + 41 = 46 which is the same as the original. Sum2 would be different though, 41 + 14 = 55 / 2 = 27.5. The 14 is there because fun() directly modifies whatever value is passed to it and increase it by 4.

6.

a.

static. Main(xyz) 1(a) 2(b) 3(w)

Dynamic. Main() 1(y) 2(bz) 3(axw)

b.

static. Main(xyz) 1(a) 2() 3(w)

dynamic Main() 1(yz) 2() 3(axw)

c.

static. Main(xyz) 1() 2(ab) 3(w)

dynamic Main() 1(ayz) 2(b) 3(xw)

d.

static. Main(xyz) 1() 2() 3(aw)

dynamic Main() 1(ayz) 2() 3(xw)

e.

static. Main(xyz) 1(a) 2(b) 3(w)

dynamic Main() 1(y) 2(abz) 3(xw)

f.

static. Main(xyz) 1() 2(b) 3(aw)

dynamic Main() 1(ayz) 2(b) 3(xw)

7. To make the statement true lets assign 7 to a, 6 to b, and 5 to c. This makes the equation 7>6>5 which is logically true, however in C what happens is the ‘7 > 6’ is evaluated first which returns a 1, leaving ‘1>5’ which is not true. This occurs because C doesn’t have a Boolean type and instead uses ints of 1 and 0 for true and false.

8.

a. ((a \* b)1 – (1 + c)2)3

-1 - -c + a \* b

b. ((((++a)1 \* (b - 1)2)3 / c)4 % d)5

cant be rewritten without parenthesis because it would cause ++a to be multiplied by b before b -1 can take place because \* is above – in the rules.

c. (a - b) / (c & (((d \* e)2 / (a – 3)3)4))1

cant be rewritten due to & being higher in the rules list than – causing the c to be & with something that is not equivalent to ( d \* e / a -3 ).

d. ((-a)1 or ((c = d)2 and e)3)4

e and d = c or -a

e. (((a > b)1 xor c)3 or (d <= 17)2)4

d <=17 or c xor a > b

9.

base : INT  
 | ID

| (‘-‘ | ‘++’ | ‘—’) base

| base (‘++‘ | ‘—’ )

| <assoc=left> base (‘ + ‘|’ \* ‘|’ &’ ) base

|<assoc=left> base (‘-’|’/’|’%’|’not’) base

|<assoc=left> base (‘<’|’<=’|’>=’|’>’|’!=’) base

| <assoc=left> base (‘=’|’+=’|’\*=’|’/=’) base

| <assoc=left> base (‘and’) base

| <assoc=left> base (‘or’|’xor’) base

;