Andrew Vo
Professor Chinua Umoja
Test 2
December 13th, 2020

Final Exam Written

Question 1 (5 points) For what types of A and B is the simple assignment statement A = B legal in C++ but not Java?

A = B is legal in C++ but it is illegal in Java for any numeric type value since it can be assigned to any numeric type variable and conversion is automatic. For example in java an int to float, in C++ A = B is perfectly legal while in Java it is illegal.

Question 2 (5 points) What are the benefits and drawbacks of replacing of all implicit type conversion with explicit type conversion in a programming language? Logically support your argument with examples. Discuss the benefits and drawbacks using the evaluation criteria of programming languages discussed in this class.

The advantages and drawbacks of implicit type conversion with explicit type conversion in a programming language is that for implicit type conversion it does automatic casting while the downside would be losing data during a conversion. For explicit type the data loss will not necessarily occur, on the other hand you must manually cast inorder to get the type you wanted.

Question 3 (18 points) The following problem list the operators line by line, where each line is a higher precedence than the operators underneath.

(postfix) ++, (postfix) --
++ (prefix), -- (prefix)
***, >=, &**
+, -, <=
-(unary), +(unary), %
>, <
&&, /, !
||, ~| (this symbol meant to be a tilde followed by a pipe)
=, /=

This problem will be using a left associative property on every operand that defaults to always doing the highest level operation from left to right.
Show the order of evaluation of the following expressions by parenthesizing all sub-expressions and placing a superscript on the right parenthesis to indicate order. Like in the example from the chapter 7 problem set.

1. **a * b - 1 + c**
2. **a * (b - 1) / c % d**
3. **(a - b) / c & (d * e / a - 3)**
4. **(a + b <= c) * (d > b - e)**
5. **-a || c = d && e**
6. **a > b ~| c || d <= 17**
7. **-a + b**
8. **a + b * c + d**
9. **E = ++(a++)**

1. ((a * b)1 - (1 + c)2)3
2. ((a * (b - 1)1)2 / (c % d)3)4
3. ((a - b)1 / (c & ((d * e)2 / (a - 3)3)4)5)6
4. ((((a + b)1 <= c)2) * ((d > (b - e)3) 4) 5
5. ((-(a)1 || c)3 = (d && e)2)4
6. ((a > b)2 ~| (c || (d <= 17)1)3)4
7. (- (a + b)1)2
8. ((a + (b * c)1)2 + d)3
9. (E = (++(a++)1)2)3

Question 4 (9 points) Solve the problems for the above expressions (from problem 3) and show your work for the values (assume 5 bits are used, two's complement notation, ~ | and & represents the logical bitwise XOR and AND operations respectively, ! Represents boolean NOT, and there is implicit type conversion between BOOLEANS and INTEGERS)

A = 5
B = 7
C = 11
D = -13
E = -2

1. ((a * b)1 – (1 + c)2)3
   ((5 * 7)1 – (1 + 11)2)3
   ((35)1 – (1 + 11)2)3
   ((35)1 – (12)2)3
   23

2. ((a * ((b - 1))1)2 / (c % d)3)4
   ((5 * ((7 - 1))1)2 / (11 % -13)3)4
   ((5 * ((6)))2 / (11 % -13)3)4
   (30 / (11 % -13)3)4
   01011 % 01101 10010+1 = 10011
   01011 % 01101 = 01110
   01110-1 = 01101 un2 = 10010 = -2

(30 / (-2))4
-15

3. (a - b) / c & (d * e / a - 3)
   (((a - b)1) / (c & ((d * e)2 / (a – 3)3)4)5)6
   (((5 - 7)1) / (11 & ((-13 * -2)2 / (5– 3)3)4)5)6
   ((-2) / (11 & ((-13 * -2)2 / (5– 3)3)4)5)6
   ((-2) / (11 & ((26) / (5– 3)3)4)5)6
   ((-2) / (11 & ((26) / (2))4)5)6
   ((-2) / (11 & (13)5)6
   01011 & 01101 01001 = 9
   (-2) / 9)
   0
4. ((((a + b)1 <= c)2) * ((d > (b – e)3)4))5
   ((((5 + 7)1 <= 11)2) * ((-13 > (7 – -2)3)4))5
   (((12 <= 11)2) * ((-13 > (7 – -2)3)4))5
   ((00000) * ((-13 > (7 – -2)3)4))5
   ((00000) * ((-13 > 9)4))5
   ((00000) * (00000))5
   0

5. (-5 || 11) = (-13) && (-2)
   1 = 0
   0 False

6. ((a > b)2 ~| (c || (d <= 17)1)3)4
   ((5 > 7)2 ~| (11 || (-13 <= 17)1)3)4
   ((00000)2 ~| (11 || (00001)1)3)4
   11 = 01011
   ((00000)2 ~| (01011 || (00001)1)3)4
   01011 OR 00001 == 01011
   ((00000) ~| (01011))4
   00000 XOR 01011 == 01011 == 11
   11

7. (-(a + b)1)2
   (-(5 + 11)1)2
   -16 = 10000
   two’s complement = 01111+1 = 11111

8. ((a + (b * c)1)2 + d)
   ((5 + (7 * 11)1)2 + -13)
   5+ 7 * 11 + (-13)
   5 + 77 - 13

82 - 13
69

9. (E = (++((a++)1))2)3
   (E = (++((5++)1))2)3
   (E = (++((6)1))2)3
   (E = (7 1))2)3
   7

Question 5 (9 points) Write a formally defined CFG for the above problem (from problem 3) and assume and imply that:

1) **only variables can have the increment or decrement operation;**
2) **~ | and & represents the logical bitwise XOR and AND operations respectively,**
3) **! Represents boolean NOT**

G=(V,E,R,S) where
V={<stmt>, <expr>, <mathexpr>, <logicexpr>, <assignexpr>, <fix>, <var>, <let>, <num>, <fixops>, <mathops>, <bitops>, <booleanops>, <unaryops>, <assignops>}
E={“” , “+” , “-” , “%” , “/”, “&” , “&&” , “|” , “||” , “~|” , “>=” , “<=” , “>” , “<” , “!”, “=” , “/=”, “++”, “--”, a-z, A-Z, 0-9} R=[
<stmt> → <expr>
<stmt> → <assignexpr>
<expr> → <fix>
<expr> → <mathexpr>
<expr> → <logicexpr>
<expr> → <var>
<mathexpr> → <var> <mathops> <var> <mathexpr> → <var> <mathops> <expr> <mathexpr> → “(“ <var> “)”
<mathexpr> → <unaryops> <var> <logicexpr> → <var> <bitops> <var> <logicexpr> → <var> <booleanops> <var> <logicexpr> → <var> <bitops> <expr> <logicexpr> → <var> <booleanops> <expr> <assignexpr> → <var> <assignops> <expr> <fix> → <fixops> “(” <let> “)”
<fix> → “(” <let> “)” <fixops>
<mathops> → “”
<mathops> → “+”
<mathops> → “-”
<mathops> → “%”
<mathops> → “/”
<bitops> → “&”
<bitops> → “&&”
<bitops> → “|”
<bitops> → “||”
<bitops> → “~|”
<booleanops> → “>=”

<booleanops> → “<=”
<booleanops> → “>”
<booleanops> → “<”
<booleanops> → “!”
<unaryops> → “-”
<unaryops> → “+”
<assignops> → “=”
<assignops> → “/=”
<fixops> → “++”
<fixops> → “--”
<var> → <let>
<var> → <num>
<let> → a-z
<let> → A-Z
<num> → 0-9
<num> → 0-9 <num>
] S=<stmt>

Question 6 (18 points) Rewrite each expression from problem 3 as function calls as if it where an object oriented programming language (like how is practically done in Ruby)? Is there a need to express this using a symbol to break precedence? Why or why not?

gt → greater than (>)
lt → less than (<)
gte → greater than equal to (>=)
lte → less than equal to (<=)
mul → multiply (*)
div → divide (/)
add → add (+)
sub → subtract (-)
preinc → increment (++var)
predec → decrement (--var)
postinc → increment (var++)
postdec → decrement (var--)
mod → modulo (%)
assign → assignment (=)
booland → boolean and (&&)
bitand → bitwise and (&)

1. sub(mul(a,b), add(1,c))

2. div(mul(),mod(c,d))

3. div(sub(a, b), bitand(c, div(mul(d, e), sub(a, 3))))
4. mul(lte(add(a, b), c), gt(d, sub(b, e)))
5. assign(or(neg(a), c) , booland(d, e))
6. nor(gt(a,b), or(c, lte(d,17)))
7. add(negate(a), b)
8. add(add(a, mul(b, c)), d)
9. assign(E, preinc(postinc(a)))

Question 7 (11 points) Write an RDA for the above mentioned problem 6?

```
public static boolean mathExpression() {
    if (nextToken != variable) {
        return false
    }else{
        if(nextToken != math_op){
            if(nextToken != log_op){
                return false
            }else{
                if(nextToken != variable){
                    return false
                }else{
                    if(nextToken!= semi_colon){
                        return false
                    }else{
                        return true
                    }
                }
            }
        }else{
            if(nextToken != variable){
                if(nextToken != math_expr){
                    return false
                }else{
                    return true
                }
            }else{
                if(nextToken != semi_colon){
```

```
                    return false
                }else{
                    return true
                }
            }
        }
    }else{
        if(nextToken != assign_op){
            return False
        }else{
            if(nextToken != variable || nextToken != number || nextToken !=
function){
                return false
            }else{
                if(nextToken != semi_colon){
                    return false
                }else{
                    return true
                }
            }
        }
    }
}
```

Question 8 (15 points) Write code to describe how you would analyze the semantic meaning of the languages of the expressions highlighted above? How would it track variable value or type? Does value or type matter? If the do matter what are your rules around the operations that they matter for?

I would apply regex in order to search the input that is going to be parsed. Implement a pointer system rather than a stack or queue as it is more flexible and versatile in this use case. From there after identifying the tokens, the mathematical operators and boolean operators I would apply the above CFG in order to find a solution.

Question 9 (5 points) Evaluate a > b > c in math logic. Evaluate the same inequality in C. Explain the difference between the two.

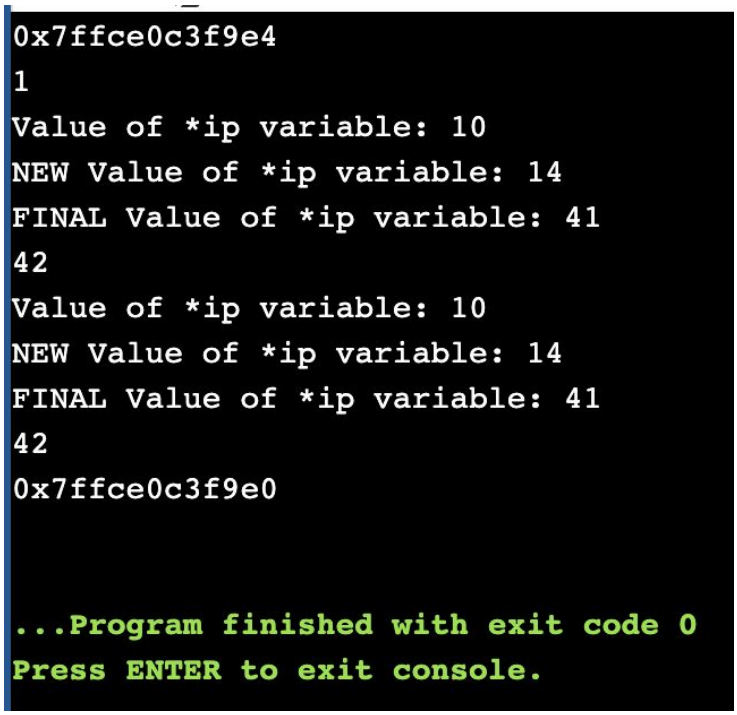
We can assume that a = 100, b = 10, and c = 1. Then the following math logic would be true. But in C it would be false since it would be evaluated from left to right and the results are used for further comparisons. Since a > b would be true, it would return a 1, and 1 > c would return false.

Question 10 (5 points) Let the function fun be defined as

```
int fun(int *k) {
        *k += 4;
return 3 * (*k) - 1;
}

int main() {
        int i = 10, j = 10, sum1, sum2;
sum1 = (i / j) + fun(&j);
 sum2 = fun(&i) + (i / j);
return 0;
}
```

Run the code in on some system that supports C and edit it to determine the values of sum1 and sum2. Explain the results



Running the code shows that fun(&j) will pass the address of j and the parameter of fun is just pointing to a location of memory. The address that fun(&j) points to will contain 10. Afterwards K is updated to reflect k = 10 + 4 which is 14. This will only then update with the contents in address location 14 which is equal to 41 and since 1 + 41 = 42, sum1 is now equal to 42 while j is now equal to 14. As for sum2 = fun(&i) + (i/j); fun(&i) passes the address of i, and the parameters of fun is a pointer to a value in a location of memory which contains 10 for the value

of i. This pointer is pointing to the address location which contains 14. Since i = 14 and j = 14 and i/j = 1. Then 41 + 1 = 42, and sum2 is now equal to 42.