# HW3

Graded

**Student**

Adam Fenjiro

**Total Points**

**54 / 56 pts**

**Question 1**

## Q1

**5** / 5 pts

✔ **– 0 pts** Correct

**– 1 pt** Missing / incorrect example / example not explained

**– 2 pts** Missing reworded claim

**– 1 pt** Reworded claims does not mention operator arity

**– 5 pts** Incorrect

**Question 2**

## Q2

**5** / 5 pts

✔ **– 0 pts** Correct

**– 1 pt** Mentions invalidity with no mention of lvalue/rvalue in reasoning

**– 5 pts** Incorrect

**Question 3**

## Q3

**5** / 5 pts

✔ **– 0 pts** Correct

**– 1 pt** Minor mistakes

**– 2.5 pts** Missing / incorrect short-circuit for AND

**– 2.5 pts** Missing / incorrect short-circuit for OR

**Question 4**

## Q4

**7** / 7 pts

✔ **– 0 pts** Correct

**– 1 pt** Missing / incorrect expression evaluation

**– 2 pts** Missing short-circuit explanation

**– 2 pts** Missing / incorrect reason against redesign

**Question 5**

**Q5** **7** / 7 pts

✔ **– 0 pts** Correct

**– 2 pts** Incorrect program

**– 2 pts** No comparison provided

**Question 6**

**Q6** **5** / 7 pts

**– 0 pts** Correct

**– 2 pts** No valid alternative provided / code is incorrect

✔ **– 2 pts** No explanation provided or explanation is incorrect

**Question 7**

**Q7** **12** / 12 pts

✔ **– 0 pts** Correct

**– 1 pt** Partially correct code shape for (a)

**– 1 pt** Partially correct x86 code for (a)

**– 1 pt** Partially correct code shape for (b)

**– 1 pt** Partially correct x86 code for (b)

**– 1 pt** Missing / incorrect code shape for (a)

**– 2 pts** Missing / incorrect x86 code for (a)

**– 1 pt** Missing / incorrect code shape for (b)

**– 2 pts** Missing / incorrect x86 code for (b)

**– 12 pts** Incorrect

**Question 8**

**Q8** **8** / 8 pts

✔ **– 0 pts** Correct

**– 1 pt** Partially correct code shape

**– 2 pts** Missing / incorrect code shape

**– 2 pts** Missing / incorrect jump table

**– 2 pts** No fall-through handling

**– 2 pts** Partially correct x86 code

**– 4 pts** Missing / incorrect x86 code

**– 8 pts** No answer

Questions assigned to the following page: 1 and 2

## Problem 1:

Given this example: * - 10 4 6 2  , that could have 2 expressions:

(* (- 10 4) 6 2) => (* 6 6 2) = 72

(* (- 10 4 6) 2) => (* 0 2) = 0

The correct expression can only be when proper parentheses is used or else it will be ambiguous. Thus, the claim can be reworded as “The issues of precedence and associativity is not with prefix /postfix notation if the number of operands for each operator is fixed”

## Problem 2:

No, I think that &(&i) is not valid in C because I don’t think you can take the address of an r-value, and &i is an r-value which contradicts. The & operator requires an lvalue as its operand as well.

Questions assigned to the following page: 3 and 4

**Problem 3:**

This is how to achieve Short-Circuit for AND, where this ensures that conditionB is evaluated only if conditionA is true. If conditionA is false, then the expression short-circuits and returns false without evaluating conditionB.

```
C/C++
if conditionA
    if conditionB
true else false
else
    false
```

This is how to achieve Short-Circuit for OR, where this structure ensures that conditionB is evaluated only if conditionA is false. If conditionA is true, the expression short-circuits and then returns true without evaluating conditionB.

```
C/C++
if conditionA
    true
else
    if conditionB
true else false
```

**Problem 4:**

- When a is zero: I think that the expression a/b becomes 0/b that evaluates to 0 if b is non zero, so 0 > 0 is false. The expression b/a results in division by zero that for sure is an undefined behavior in C meaning the program may crash, produce an error, or behave unpredictably.

- When b is zero: I think that the expression a/b results in division by zero that for sure is an undefined behavior. The expression b/a becomes 0/a that evaluates to 0 if a is non zero, so 0 > 0 is false.

- Guaranteeing false: I think we need to implement checks before doing division operations to ensure that division by zero results in a false evaluation rather than undefined behavior. I think that this would require additional runtime checks that could impact performance and potentially might lead to confusion for programmers used to traditional behavior.

Questions assigned to the following page: 5 and 6

**Problem 5:**

```C/C++
//using while loop
line = read_line();
while (!all_blanks(line)) {
    consume_line(line);
    line = read_line();
}
```

The while loop version has read_line() function that is called before the loop and then at the end of each iteration which I think makes sure that the loop only continues if the line is not all blanks while the mid-test version avoids the duplication of the read_line() call. However, I think that while loop version is easier to understand since it looks like something commonly used.

**Problem 6:**

The provided example is convincing, be here is my other version below:

```C/C++
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        if (A[i][j] != 0) break;  // Exit if a non-zero element found
    }
    if (j == n) {  // Check if the loop completed without breaking
        first_zero_row = i;
        break;  // Exit since the first all-zero row found
    }
}
```

Question assigned to the following page: 7

**Problem 7:**

(a) The language uses short-circuit evaluation

```
Unset
L1: #loop body
    if (a > -20) go to L2
    go to L1
L2: if (a < -10) go to L3
    if (a > 20) goto L3
    go to L1
L3: #after loop
```

```
Unset
L1: #loop body
    movl -8(%rbp), %eax
    cmpl $-20, %eax
    jle L1
    movl -8(%rbp), %eax
    cmpl $-10, %eax
    jl L3
    movl -8(%rbp), %eax
    cmpl $20, %eax
    jle L1
L3: #after loop
```

Question assigned to the following page: 7

(b) The language does not support short-circuit evaluation

```C/C++
L1: #loop body
    t1 = a > -20
    t2 = a < -10
    t3 = a > 20
    t4 = t2 or t3
    t5 = t1 and t4
    if t5 go to L3
    go to L1
L3: #after loop
```

```Unset
L1: #loop body
    movl -8(%rbp), %eax
    cmpl $-20, %eax
    setg %r8b
    movl -8(%rbp), %eax
    cmpl $-10, %eax
    setl %r9b
    movl -8(%rbp), %eax
    cmpl $20, %eax
    setg %r10b
    orb %r10b, %r9b
    andb %r9b, %r8b
    testb %r8b, %r8b
    je L1
L3: #after loop
```

Question assigned to the following page: 8

**Problem 8:**

a) code shape

```
Unset
L1: #switch statement body
if t200 go to L200
if t202 go to L202
if t203 go to L203
if t205 go to L205
if tDefault go to LDefault
go to Lend

L200:
call foo1
go to Lend

L202:
call foo2
go to Lend

L203:
call foo3
go to Lend

L205:
call foo3
go to Lend

LDefault:
call foo4

Lend: #end of the switch statement
```

Question assigned to the following page: 8

(b) x86-64 code

```
Unset
    movl    -4(%rbp), %eax
    cmpl    $200, %eax
    je      .Lfoo1
    cmpl    $202, %eax
    je      .Lfoo2
    cmpl    $203, %eax
    je      .Lfoo3
    cmpl    $205, %eax
    je      .Lfoo3
    jmp     .Lfoo4

.Lfoo1:
    call    foo1
    jmp     .Lend
.Lfoo2:
    call    foo2
    jmp     .Lend
.Lfoo3:
    call    foo3
    jmp     .Lend
.Lfoo4:
    call    foo4

.Lend: #end code
```