I, Cole Bardin, agree to adhere to the Drexel Academic code of conduct during this exam; other than the instructor/TA/RCF and the materials provided by the instructor, no other resources were utilized.

Summer CS270: Midterm	**July 28th 2023**

Directions: you have 110 minutes to complete this midterm. Time is tight and you will likely not have much to spare to access outside resources, so ideally you are familiar enough with the content to draw from your own brain -- Late submissions are NOT accepted. You may use your own handwritten notes, the provided lecture powerpoints/videos, the DrRacket software, and the instructor/TA as resources. No other references (talking to another human, going to an external website, or reading other files/book) are permitted. Submit your edited PDF (or picture images) into bbLearn –be sure to tag which pages go with which answers! The Racket file with your implementations for questions10and11should be turned in as a separate bbLearn submission.

This test totals 100 points (+1 bonus point if 70%+ students did the anonymous midcourse survey on bbLearn, and +1 for an accurate estimate of your midterm score at end)

#1 [5pts]. Write a Racket expression using only parentheses together with L, first and rest which would give out the number 6 where L is the list (5 (((0 6) 3) 2) 4 7)

```
(first (rest (first (first (first (rest L))))))
```

#2 [5pts]. Without using any quote symbols, write a racket expression using only parentheses, cons, integers and null which would give out the list ((4 7) 1)

```
(cons (rest (rest L)) (cons 1 null))
```

#3 [5pts]. Fill in the boxes with 0/1 in following truthtable to find all possible outputs for the expression below. Be sure to go in the standard order of inputs that have been done in the labs/videos/homeworks (that is, the top row is all trues and the bottom row is all falses). Put a circle around the column that represents the final evaluation for the entire expression. (The input columns have been separated out for you for readability).

A	B
1	1
1	0
0	1
0	0

$$(A \wedge B) \rightarrow \neg B$$

1
0
0
0

0	0
1	1
1	0
1	1

#4 [10pts]. Write a Boolean expression that would have the following truth table. You must use actual logic symbols and not ascii approximations or English words. You do not need to simplify your answer.

P	Q	R	expr
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	0
0	0	0	1

$(P\wedge Q\wedge\neg R)\vee(\neg P\wedge Q\wedge R)\vee(\neg P\wedge\neg Q\wedge\neg R)$

#5. For this problem, consider the expression $(A \wedge \neg C) \rightarrow B$

[a] 10 pts. Use the rules of Boolean Algebra linked in bbLearn to convert the expression into CNF. Do not skip any steps, and be sure to state the rule number being applied for each step! (the rule sheet with numbers for every Boolean rule is on bbLearn). For full credit, you must use legitimate logic symbols, not English/ascii substitutions.

(A∧¬C)⇒B	Premise
¬(A∧¬C)∨B	Def of Implies (rule 20)
(¬A∨¬¬C)∨B	DeMorgan 1 (rule 18)
(¬A∨C)∨B	Double Negative (rule 17)
¬A∨C∨B	Associativity OR (rule 2)

[b] 5 pts Using A as 1, B as 2 etc, denote the above expression in DIMACS notation (Note: depending on your simplification, it might take more than one line)

```
p cnf 3 1
-1 2 3 0
```

#6 [10pts] . Implement a non-recursive function (with no separate defined helpers) according to the specs below
; Input contract: n is a nonnegative integer. Note: there is only ONE input to this function, not two!
; Output contract: (decList n) is a *function* which takes a list L of integers and returns that same list but with
; each member decreased by n.
; Example: ((decList 3) '(7 9 6)) would return '(4 6 3) and ((decList 2) ‘(8 13 1)) would return ‘(6 11 -1)

```
(define (decList n)
     (lambda (L) (map (lambda (a) (- a n)) L))
)
```

#7 [10pts]

Suppose a,b are integers and A is the proposition that a is even, and B is the proposition that b is even.

Using only two variables, express in DIMACS that a+b is even. (hint: it may take multiple clauses)

Note: be sure to show your work so that a reader can follow your thought process. Also, include a comment defining your variables

for integers a,b, let
$A \Leftrightarrow$ (a is even)
$B \Leftrightarrow$ (b is even)
$S \Leftrightarrow$ ((a+b) is even)

S would be even if both a and b are even or if a and b are both odd
Meaning S is true if $(A \land B) \lor (\neg A \land \neg B)$

This can be recognized to be the inverse of XOR
$\neg((A \text{ xor } B))$
Using the definition of XOR
$\neg((A \land \neg B) \lor (A \land \neg B))$
Using DeMorgan's law
$\neg(A \land \neg B) \land \neg(\neg A \land B)$
Using DeMorgan's again
$(\neg A \lor \neg\neg B) \land (\neg\neg A \lor \neg B)$
Using Double negation
$(\neg A \lor B) \land (A \lor \neg B)$ is equivalent to S.
Using A=1 and B=2, the following DIMACS expression can be written

```
p cnf 2 2
-1 2 0
1 -2 0
```

#8 [15pts] . Construct an equational reasoning proof showing (myst ‘(x y)) results in ‘(y x). Do not skip any steps and be sure to provide the justification for each step. Our definition for the function myst is the following:

```
(define (myst L [M null]) (if (null? L) M (myst (rest L) (cons (first L) M))))
```

Note: this question is a bit long, but you can save time by copy/pasting from the previous step and just modifying the appropriate portion (Please number your lines. When done correctly, there should be 16 total).

1. (myst '(x y)) ; premise
2. (if (null? '(x y)) null (myst (rest '(x y)) (cons (first '(x y)) null))) ; apply definition of myst
3. (if (#f) null (myst (rest '(x y)) (cons (first '(x y)) null))) ; evaluate IF condition
4. (myst (rest '(x y)) (cons (first '(x y)) null)) ; evaluate IF statement
5. (myst (rest '(x y)) (cons 'x null)) ; evaluate first
6. (myst (rest '(x y)) '(x)) ; evaluate cons
7. (myst '(y) '(x)) ; evaluate rest
8. (if (null? '(y)) '(x) (myst (rest '(y)) (cons (first '(y)) '(x)))) ; apply definition of myst
9. (if (#f) '(x) (myst (rest '(y)) (cons (first '(y)) '(x)))) ; evaluate IF condition
10. (myst (rest '(y)) (cons (first '(y)) '(x))) ; evaluate IF statement
11. (myst (rest '(y)) (cons 'y '(x))) ; evaluate first
12. (myst (rest '(y)) '(y x)) ; evaluate cons
13. (myst '() '(y x)) ; evaluate rest
14. (if (null? '()) '(y x) (myst (rest '()) (cons (first '()) '(y x)))) ; apply definition of myst
15. (if (#t) '(y x) (myst (rest '()) (cons (first '()) '(y x)))) ; evaluate IF condition
16. '(y x) ; evaluate IF statement

#9 [5 pts] . A client wants a function that matches the specifications below.

Input contract = n is a nonnegative integer

Output contract = a positive integer according to the following rules

(a) if the number is even, then it adds 2 to the tally and keeps going, running the function again on half the value
(b) if the number is odd and also divisible by 3, then it adds 3 to tally and runs the function again on a third of the value
(c) the function stops recursing whenever the input hits 0 and in this base case outputs a 4
(d) if none of the above conditions are met, it adds 6 to the tally and runs the function on one less than the input value.

Example: suppose the user inputs n=42. The client would like a 26 to be returned, according to the following reasoning:

Since 42 is even, the tally will be 2 + (f 21). Since 21 is odd and divisible by 3, we do 2 + 3 + (f 7). since 7 is neither even nor divisible by 3, we get 2 + 3 + 6 + (f 6). Since 6 is even, we get 2 + 3 + 6 + 2 + (f 3). since 3 is divisible by 3, 2 + 3 + 6 + 2 + 3 +(f 1). since 1 is neither even nor divisible by 3, we get 2 + 3 + 6 + 2 + 3 + 6 + (f 0). And finally, since this is now our base case, our computation ends and our tally stands at 2 + 3 + 6 + 2 + 3 + 6 + 4 = 26.

Your novice intern writes the following implementation to attempt to do the client's bidding

```
(define (f n)
  (cond
    [(zero? (remainder n 2)) (+ 2 (f (quotient n 2)))]
    [(zero? (remainder n 3)) (+ 3 (f (quotient n 3)))]
                      ; note: we already know that the input must be odd to have hit the case above
    [(zero? n) 4]
    [else (+ 6 (f (- n 1)))]))
```

Will this work? If not, why not, and what minor change could you do to fix it?
[Note: it is not necessary to use DrRacket to figure this out, unless you really want to. You can just manually describe the changes you'd make]

This would not work because base case is not checked first. imagining the function calls for when n is equal to zero, the first case (checking if remainder of n/2 is zero) would be true. this would cause an infinite loop as f would be called on zero again. to fix this, the true base case that checks whether or not n is zero should be placed at the top of the cond statement. this would stop the infinite loop to occur when (f 0) gets called.

This means the new order would be:
check if n=0
check if n is divisible by 2
check if n is divisible by 3
else add 6 + (f (n-1))

Questions #10 and #11 [each is 10pts] appear in the Racket file in bbLearn. Before submitting this pdf into Gradescope, please be sure to answer the Self-Assessment portion below (it is worth extra credit!)

Not counting any extra credit, what **numerical** score do you think you earned on this midterm: 90
(if you get within +5pts of your actual score, you will earn a bonus point for accurate self-assessment)