CS 61A Fall 2022

Structure and Interpretation of Computer Programs

Final Solutions

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

- ◯ You must choose either this option
- ◯ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

- ☐ You could select this choice.
- ☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

You can complete and submit these questions before the exam starts.

(a) What is your full name?

(b) What is your student ID number?

(c) What is your @berkeley.edu email address?

1. (15.0 points) What Would Python Do?

Assume the code below has been executed.

```
triple = lambda z: 3 * z
dec = lambda z: z - 1

def alt(f, g):
    def h(x):
        print(fns[0](fns[1](x)))
        fns[0] = fns[1]
        fns[1] = fns[0]
        return h
    fns = [f, g]
    return h

class Sub:
    fns = [lambda z: str(z) + str(z),
           lambda z: 2 * z]

    def __init__(self, f, g):
        self.fns = [f, g]

    def h(self, x):
        print(Sub.fns[0](Sub.fns[1](x)))
        Sub.fns = self.fns
        return self

    def __repr__(self):
        return str(self.fns[0]('ha'))

class Pub(Sub):
    fns = [lambda z: [z],
           lambda z: z]

    def __init__(self, f, g):
        fns = [f, g]  # Careful!
```

What is the output **displayed by the interactive Python interpreter** after evaluating each of the following expressions.

Expressions are evaluated in order, and **earlier expressions may affect later ones**.

All of the expressions for this question appear here (so that you can work out the answer without flipping pages), but **answer the question using the multiple choice prompts on the following pages**.

```
>>> print(20, (lambda x: print)(20)(22))

>>> (print(4) or 3+2) or 1/0

>>> alt(triple, dec)(8)(8)

>>> Sub(triple, dec).h(8).h(8)

>>> [f(8) for f in Pub(dec, triple).fns]

>>> Pub(dec, triple).h(8).h(8)
```

(a) (1.0 pt) What is the **first** line displayed for the **first** expression: `print(20, (lambda x: print)(20)(22))`

- ◯ 20
- ● 22
- ◯ 20 None
- ◯ 22 None
- ◯ A function value

(b) (1.0 pt) What is the **second** line displayed for the **first** expression: `print(20, (lambda x: print)(20)(22))`

- ◯ 20
- ◯ 22
- ● 20 None
- ◯ 22 None
- ◯ A function value

(c) (1.0 pt) What is the **first** line displayed for the **second** expression: `(print(4) or 3+2) or 1/0`

- ● 4
- ◯ 5
- ◯ 4 5
- ◯ None 5
- ◯ There is no first line because an error occurs

(d) (1.0 pt) What is the **second** line displayed for the **second** expression: `(print(4) or 3+2) or 1/0`

- ◯ 4
- ● 5
- ◯ 4 5
- ◯ None 5
- ◯ There is no second line because an error occurs

(e) (1.0 pt) What is the **first** line displayed for the **third** expression: `alt(triple, dec)(8)(8)`

- ◯ 6
- ● 21
- ◯ 23
- ◯ 72

(f) (1.0 pt) What is the **second** line displayed for the **third** expression: `alt(triple, dec)(8)(8)`

- ● 6
- ◯ 21
- ◯ 23
- ◯ 72

(g) (1.0 pt) What is the **third** line displayed for the **third** expression: `alt(triple, dec)(8)(8)`

- (●) A function
- ( ) `None`
- ( ) `'h'`
- ( ) `h`
- ( ) There is no third line

(h) (1.0 pt) What is the **first** line displayed for the **fourth** expression: `Sub(triple, dec).h(8).h(8)`

- ( ) 8
- ( ) 12
- ( ) 16
- ( ) 21
- (●) 1616
- ( ) `'8888'`
- ( ) There is no first line because an error occurs

(i) (1.0 pt) What is the **second** line displayed for the **fourth** expression: `Sub(triple, dec).h(8).h(8)`

- ( ) 8
- ( ) 12
- ( ) 16
- (●) 21
- ( ) 1616
- ( ) There is no second line because an error occurs

(j) (1.0 pt) What is the **third** line displayed for the **fourth** expression: `Sub(triple, dec).h(8).h(8)`

- ( ) A function
- ( ) `ha`
- ( ) `haha`
- (●) `hahaha`
- ( ) `['ha']`
- ( ) `['haha']`
- ( ) `['hahaha']`
- ( ) There is no third line (for any reason, including an error)

(k) **(2.0 pt)** What is displayed for the **fifth** expression: `[f(8) for f in Pub(dec, triple).fns]`

- ◯ `[7, 21]`
- ◯ `[7, [21]]`
- ◯ `['88', 16]`
- ◯ `['88', [16]]`
- ● `[[8], 8]`
- ◯ `[[8], [8]]`
- ◯ None of these

(l) **(1.0 pt)** What is the **first** line displayed for the **sixth** expression: `Pub(dec, triple).h(8).h(8)`

- ◯ 8
- ◯ `[8]`
- ● 21
- ◯ 23
- ◯ 1616
- ◯ `'8888'`

(m) **(1.0 pt)** What is the **second** line displayed for the **sixth** expression: `Pub(dec, triple).h(8).h(8)`

- ◯ 8
- ● `[8]`
- ◯ 21
- ◯ 23
- ◯ 1616
- ◯ `'8888'`

(n) **(1.0 pt)** What is the **third** line displayed for the **sixth** expression: `Pub(dec, triple).h(8).h(8)`

- ◯ A function
- ◯ `ha`
- ◯ `haha`
- ◯ `hahaha`
- ● `['ha']`
- ◯ `['haha']`
- ◯ `['hahaha']`
- ◯ There is no third line (for any reason, including an error)

2. (4.0 points) What About Scheme?

Choose the output **displayed by the interactive Scheme interpreter** when each expression below is evaluated.

To help distinguish between the backtick (`) and apostrophe (') characters: The first character in subpart 3's line of code is a backtick. All remaining ticks are apostrophes (the first character of line 2, the 8th non-space character of line 3, and three characters in line 4).

(a) (1.0 pt) `(+ (* 3 2) (+ 3 2))`

- (●) `11`
- ( ) `(+ (* 3 2) (+ 3 2))`
- ( ) `(+ 6 (+ 3 2))`
- ( ) `(+ (* 3 2) 5)`
- ( ) `(+ 6 5)`

(b) (1.0 pt) `'(+ (* 3 2) (+ 3 2))`

- ( ) `11`
- (●) `(+ (* 3 2) (+ 3 2))`
- ( ) `(+ 6 (+ 3 2))`
- ( ) `(+ (* 3 2) 5)`
- ( ) `(+ 6 5)`

(c) (1.0 pt) `` `(+ ,(* 3 '2) (+ 3 2)) ``

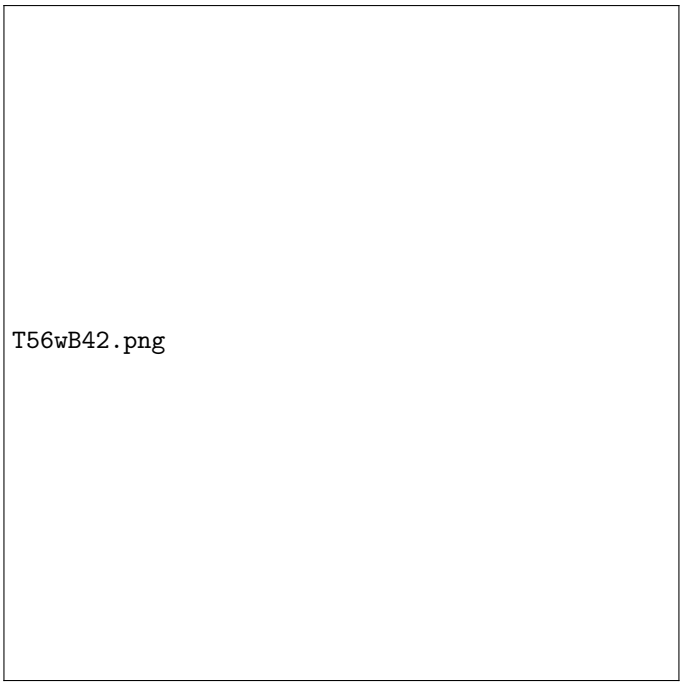
- ( ) `11`
- ( ) `(+ (* 3 2) (+ 3 2))`
- (●) `(+ 6 (+ 3 2))`
- ( ) `(+ (* 3 2) 5)`
- ( ) `(+ 6 5)`
- ( ) Error

(d) (1.0 pt) `'(+ (* '3 2) '(+ 3 2))`

- ( ) `11`
- ( ) `(+ (* 3 2) (+ 3 2))`
- ( ) `(+ (* 3 2) (quote (+ 3 2)))`
- ( ) `(+ (* (quote 3) 2) (+ 3 2))`
- (●) `(+ (* (quote 3) 2) (quote (+ 3 2)))`
- ( ) `(+ (* 3 2) (quasiquote (+ 3 2)))`
- ( ) `(+ (* (quasiquote 3) 2) (+ 3 2))`
- ( ) `(+ (* (quasiquote 3) 2) (quasiquote (+ 3 2)))`
- ( ) Error

3. (6.0 points) Sum Total

Executing the code on the left produced this environment diagram. Complete the diagram, then answer questions about it. The diagram itself will not be scored; only the questions. Blanks that do not have a letter label will not be scored.



(a) (1.0 pt) *Blank (a)*: **How many** elements are in the list bound to `result` in the Global frame?

- ○ 0
- ● 1
- ○ 2

(b) (2.0 pt) *Blank (b)*: What value is bound to `totals` in the Global frame?

- ○ An empty list
- ○ A list `[3]`
- ● A list `[6]`
- ○ None of the above

(c) (1.0 pt) *Blank (c)*: What value is bound to `s` in frame `f2`?

- ○ An empty list
- ● A list `[1]`
- ○ A list `[4]`
- ○ A list `[1, 2, 4]`

(d) (2.0 pt) *Blank (d)*: What is the return value of frame `f2`?

6.1

4. (16.0 points) Advent of Code

The choice of programming language can affect the difficulty of implementation. This problem was adapted from Day 1 of the 2020 Advent of Code, a set of 25 programming challenges released annually in December. Complete the following Python, Scheme, and SQL solutions to this problem:

Given a list of numbers, find the two numbers in the list that sum to 2020. You are guaranteed that there is **exactly one pair of numbers that sum to 2020** and that the list does not contain 1010.

For example, given `[979, 1721, 366, 299, 675, 1456]`, output `[1721, 299]`, since 1721+299 = 2020.

(a) (5.0 points) Python

```
def find_pair(s):
    '''Return the two elements of s that sum to 2020, in the order that they appear in s.

    >>> find_pair([979, 1721, 366, 299, 675, 1456])
    [1721, 299]
    '''
    for i in __(a)__:
        for j in __(b)__:
            __(c)__:
                return __(d)__
```

i. (1.0 pt) Fill in Blank (a)

- (●) `s`
- ( ) `[s]`
- ( ) `s[:len(s)-1]`
- ( ) `zip(s, s)`
- ( ) `range(s)`

ii. (1.0 pt) Fill in Blank (b)

- ( ) `i`
- ( ) `[i]`
- (●) `s`
- ( ) `s[1:]`
- ( ) `range(s)`

iii. (2.0 pt) Fill in Blank (c)

```
if i + j == 2020:
```

iv. (1.0 pt) Select all expressions that could fill in Blank (d).

- [■] `[i, j]`
- [ ] `list(i, j)`
- [ ] `[list(i, j)]`
- [ ] `[next(i), next(j)]`

(b) (6.0 points) Scheme

Implement `find-pair` in Scheme. You may call `contains?`, which takes a Scheme list of numbers `s` and a number `n`. It returns whether `s` contains `n`.

```
;;; > (contains? '(1 2 3) 3)
;;; #t
;;; > (contains? '(1 2 3) 4)
;;; #f
(define (contains? s n) ...)  ; Assume contains? has been implemented correctly for you.
;;; > (find-pair '(979 1721 366 299 675 1456))
;;; (1721 299)
(define (find-pair s)
  (if __(a)__ ( __(b)__ (car s) (- 2020 (car s))) __(c)__ ))
```

i. (3.0 pt) Fill in Blank (a)

```
(contains? (cdr s) (- 2020 (car s)))
```

ii. (1.0 pt) Fill in Blank (b)

- ○ append
- ○ car
- ○ cdr
- ○ cons
- ● list

iii. (2.0 pt) Fill in Blank (c).

- ○ `nil`
- ○ `(cdr s)`
- ○ `(map (lambda (x) (- 2020 x)) (cdr s))`
- ○ `(filter (lambda (x) (= 2020 (+ (car s) x))) (cdr s))`
- ● `(find-pair (cdr s))`
- ○ `(find-pair (map (lambda (x) (- 2020 x)) (cdr s)))`
- ○ `(find-pair (filter (lambda (x) (= 2020 (+ (car s) x))) (cdr s)))`

(c) (5.0 points) SQL

Implement a SQL query that creates a **one-row, two-column table** of numbers where the single row contains the two numbers in table `s` that sum to 2020. The order that the numbers appear will not be graded.

```
CREATE TABLE s AS
  SELECT 979 as num UNION SELECT 1721 UNION SELECT 366 UNION
  SELECT 299        UNION SELECT 675  UNION SELECT 1456;
SELECT __(a)__ FROM __(b)__ WHERE __(c)__;
```

i. (1.0 pt) Fill in Blank (a)

- ○ `s, 2020 - s`
- ○ `num, 2020 - num`
- ○ `a, b`
- ○ `s.a, s.b`
- ● `a.num, b.num`

ii. (2.0 pt) Fill in Blank (b)

- ○ `s`
- ○ `a, b`
- ○ `s, s`
- ● `s AS a, s AS b`
- ○ `a AS s, b AS s`

iii. (2.0 pt) Fill in Blank (c)

- ○ `a + b = 2020`
- ○ `a.num + b.num = 2020`
- ○ `a < b AND a + b = 2020`
- ● `a.num < b.num AND a.num + b.num = 2020`
- ○ `a = b AND a + b = 2020`
- ○ `a.num = b.num AND a.num + b.num = 2020`
- ○ `(2020 - num) IN s`
- ○ `CONTAINS(2020 - num, s)`

5. (5.0 points) Two Cents

One of the most useful aspects of programming is that code can often be reused to solve similar problems. The `count_change` function takes an integer `amount` of cents and returns the number of ways to make change for that `amount` in US currency.

```
coins = (1, 5, 10, 25, 50)

def count_change(amount):
    '''Return the number of ways to make change for amount.

    >>> count_change(10) # 10, 5-5, 5-1-1-1-1-1, 1-1-1-1-1-1-1-1-1-1
    4
    >>> count_change(20) # 10-10, 10-5-5, 10-5-1-1-1-1-1, 10-1-1-1-1-1-1-1-1-1, 5-5-5-5, ...
    9
    >>> count_change(100)
    292
    '''
    def helper(remaining, coin_index):
        if coin_index == len(coins) or remaining < 0:
            return 0
        if remaining == 0:
            return 1
        return (helper(remaining - coins[coin_index], coin_index    ) +
                helper(remaining                    , coin_index + 1))
    return helper(amount, 0)
```

Modify this code to solve the following new problem:

Your cash register only has `k` of each type of coin. Implement `count_change_register`, which counts the number of ways to make change for `amount` using at most `k` coins of each type.

```
def count_change_register(amount, k):
    '''Return the number of ways to make change for amount using at most k of each coin type.

    >>> count_change_register(20, 10) # Excludes 20 pennies and excludes 1 nickel + 15 pennies
    7
    >>> count_change_register(20, 2) # 10-10, 10-5-5
    2
    >>> count_change_register(100, 10)
    84
    >>> count_change_register(100, 100)
    292
    '''
    def helper(remaining, coin_index, n):
        if coin_index == len(coins) or remaining < 0 or __(a)__:
            return 0
        if remaining == 0:
            return 1
        return (helper(remaining - coins[coin_index], coin_index    , __(b)__ ) +
                helper(remaining                    , coin_index + 1, __(c)__ ))
    return helper(amount, 0, 0)
```

(a) (2.0 pt) Fill in Blank (a)

- ◯ `True`
- ◯ `False`
- ◯ `n > 0`
- ◯ `k > 0`
- ● `n > k`
- ◯ `n == k`
- ◯ `n > 10`
- ◯ `n == 10`

(b) (1.0 pt) Fill in Blank (b)

- ◯ `k`
- ◯ `k - 1`
- ◯ `k + 1`
- ◯ `n`
- ● `n + 1`
- ◯ `n - 1`
- ◯ `0`
- ◯ `10`

(c) (1.0 pt) Fill in Blank (c)

- ◯ `k`
- ◯ `k - 1`
- ◯ `k + 1`
- ◯ `n`
- ◯ `n + 1`
- ◯ `n - 1`
- ● `0`
- ◯ `10`

(d) (1.0 pt) What does `count_change_register(25, 2)` return?

- ◯ 0
- ◯ 1
- ● 2
- ◯ 3
- ◯ 4
- ◯ None of these

6. (6.0 points) Barking up the Wrong Tree

After years of using SQL, John the Dog Breeder decides to switch to a Python `Tree` to track his dogs' heredity.

Implement `tree_to_table`, which takes a `Tree` instance `t` representing dogs. Each node in the tree has a label that is the name of a dog and children corresponding to the children of that dog. The `tree_to_table` function returns a list of the parent-child relationships in `t` **in any order**. Each parent-child relationship is represented as a two-element tuple, where the first element is the name of the parent, and the second is the name of the child.

The `Tree` class is defined on page 2 of the Midterm 2 Study Guide with instance attributes `label` and `branches`. You may **not** use any functions from the study guide defined outside of the `Tree` class.

For `Tree('E', [Tree('F', [Tree('A', [Tree('B'), Tree('C')]), Tree('D', [Tree('H')]), Tree('G')])])`

One valid output is: `[('A', 'B'), ('A', 'C'), ('D', 'H'), ('F', 'A'), ('F', 'D'), ('F', 'G'), ('E', 'F')]`

```
def tree_to_table(t):
    relationships = __(a)__
    for i in __(b)__:
        relationships.__(c)__(__(d)__)
    return relationships
```

(a) (3.0 pt) Fill in Blank (a)

```
[(t.label, i.label) for i in t.branches]
```

(b) (1.0 pt) Fill in Blank (b)

- ○ `branches`
- ● `t.branches`
- ○ `Tree.branches`
- ○ `tree_to_table(branches)`
- ○ `tree_to_table(t.branches)`
- ○ `tree_to_table(Tree.branches)`
- ○ `relationships`
- ○ `[p for p, c in relationships]`

(c) (1.0 pt) Fill in Blank (c)

- ○ `append`
- ● `extend`

(d) (1.0 pt) Fill in Blank (d)

- ○ `i`
- ○ `i.branches`
- ○ `t.branches`
- ● `tree_to_table(i)`
- ○ `tree_to_table(i.branches)`
- ○ `tree_to_table(t.branches)`

7. (10.0 points) One Cent

In Penney's Game, two players each choose a different sequence of three coin flips (Heads or Tails). A coin is flipped repeatedly until one of those two sequences occurs; the winner is the player whose sequence is flipped. Here's an example:

Player 1 selects the sequence "HHH" and Player 2 selects the sequence "THH", then a coin is flipped repeatedly, yielding the following sequence of coin flips: `HTTHTHTTHTHTHH`. Since "THH" was flipped first (the last three flips), Player 2 wins.

(a) (6.0 points)

First, implement the generator function `three_flips`. It takes as input a (potentially infinite) iterator `coin`, each element of which is either `'H'` or `'T'` (denoting coin flips of Heads and Tails, respectively). It yields a sequence of triples of coin flip outcomes from `coin`.

```
def three_flips(coin):
    '''
    >>> a = three_flips(iter("HTTHHT"))
    >>> next(a)  # Coin flips 0, 1, and 2
    'HTT'
    >>> next(a)  # Coin flips 1, 2, and 3
    'TTH'
    >>> next(a)  # Coin flips 2, 3, and 4
    'THH'
    >>> next(a)  # Coin flips 3, 4, and 5
    'HHT'
    >>> next(a)  # A StopIteration exception is raised
    StopIteration
    '''
    start = __(a)__
    __(b)__:
        yield start
        start = __(c)__ + __(d)__
```

i. (2.0 pt) Fill in Blank (a)

```
next(coin) + next(coin) + next(coin)
```

ii. (1.0 pt) Fill in Blank (b)

- (●) `while True`
- ( ) `while next(coin) is not None`
- ( ) `for x in coin`

iii. (2.0 pt)

```
start[1:]
```

iv. (1.0 pt) Fill in Blank (d)

- ◯ `coin`
- ◯ `coin.pop()`
- ● `next(coin)`

(b) (4.0 points)

Implement the function `penney`. It receives as input an iterator `coin` over `'H'` and `'T'` and two sequences `p1` and `p2`. It returns 1 if Player 1 wins Penney's Game and 2 if Player 2 wins. Assume that one player wins before the `coin` runs out of elements, and that `p1` and `p2` are distinct three-element sequences of 'H' and 'T'.

```
def penney(coin, p1, p2):
    '''Return the winner of Penney's game, where Player 1 chooses p1 and Player 2 chooses p2.

    >>> penney(iter("HTTHTHTTHTHTHH"),"THH","HHH")
    1
    >>> penney(iter("HTTHTHTTHTHTHH"),"HHH","THH")
    2
    >>> penney(iter("HTTHTHTTHTHTHH"),"HTT","THH")  # HTT happens first
    1
    '''
    it = __(a)__(lambda x: __(b)__, __(c)__)
    if next(it) == p1:
        return 1
    return 2
```

i. (1.0 pt) Fill in Blank (a)

```
filter
```

ii. (2.0 pt) Fill in Blank (b)

```
x in [p1, p2]
```

iii. (1.0 pt) Fill in Blank (c)

- ○ `it`
- ○ `coin`
- ○ `(p1, p2)`
- ● `three_flips(coin)`

8. (6.0 points) A Parentheses Scheme

In a fit of Scheme-induced rage, you've decided that all internal parentheses must be eliminated! Implement the procedure `remove-parens` that takes as input a Scheme list and returns that list with all internal parentheses removed.

```
;;; > (remove-parens '(((1) 2 3) 4 5 (6 (7)) (8 10)))
;;; (1 2 3 4 5 6 7 8 10)
;;; > (remove-parens '(((a) b (c) ()) (d) e (f (((g)))) (h i)))
;;; (a b c d e f g h i)
(define (remove-parens s)
  (cond
    ( (null? s) nil    )
    ( __(a)__   __(b)__)
    ( else      __(c)__)))
```

(a) (1.0 pt) Fill in Blank (a)

- ○ `(list? s)`
- ● `(list? (car s))`
- ○ `(list? (cdr s))`
- ○ `(null? (cdr s))`
- ○ `(not (number? (car s)))`

(b) (3.0 pt) Fill in Blank (b)

```
(append (remove-parens (car s)) (remove-parens (cdr s)))
```

(c) (2.0 pt) Select all of the expressions below that could fill in Blank (c).

- ☐ `(remove-parens (cdr s))`
- ☐ `(remove-parens (cons (car s) (cdr s)))`
- ☐ `(remove-parens (list (car s) (cdr s)))`
- ■ `(cons (car s) (remove-parens (cdr s)))`
- ☐ `(cons (remove-parens (car s)) (remove-parens (cdr s)))`
- ☐ `(list (car s) (remove-parens (cdr s)))`
- ☐ `(list (remove-parens (car s)) (remove-parens (cdr s)))`

9. (7.0 points) Pokemon SQarLet

You've been hired as the newest Gym Leader of the Paldea region! Now you need to decide what team to make. All your available Pokemon (for this question, Pokemon stats have been simplified) are listed in a SQL table `Pokemon`:

Name	Type1	Type2	Move1	Move2	BaseAttack	BaseDefense
Rattata	Normal	None	Endeavor	Quick Attack	0	0
Maushold	Normal	None	Population Bomb	Tidy Up	30	30
Cyclizard	Dragon	Normal	Shed Tail	Dragon Rush	100	40
Meoscadera	Grass	Dark	Flower Trick	Agility	80	40
Farigiraf	Normal	Psychic	Agility	Twin Beam	40	50

You also have a table `Moves` listing all Pokemon moves:

Name	MoveAttack	MoveDefense
Endeavor	250	0
Quick Attack	40	45
Shed Tail	0	150
Dragon Rush	80	0
Flower Trick	90	55
Agility	50	20
Twin Beam	80	30
Population Bomb	100	50
Tidy Up	30	150

Create a table `Attack`, which lists the `Name` and `TotalAttack` of each Pokemon. The total attack of a Pokemon is equal to the sum of a Pokemon's `BaseAttack` and the `MoveAttack` of both moves it knows. Order rows by total attack.

For example, Maushold has a base Attack of 30, and has two moves with `MoveAttack` values 100 and 0, respectively. Its total attack is 30+100+0 = 130. With the above tables, you should have the following result:

Name	TotalAttack
Maushold	130
Farigiraf	170
Cyclizar	180
Meowscadera	220
Rattata	290

```
CREATE TABLE Attack AS
  SELECT __(a)__ AS Name, __(b)__ AS TotalAttack FROM __(c)__ WHERE __(d)__ ORDER BY TotalAttack;
```

(a) (1.0 pt) Select all options that could fill in Blank (a).

- ☐ `Name`
- ■ `p.Name`
- ☐ `q.Name`
- ☐ `a.Name`
- ☐ `b.Name`

(b) (2.0 pt) Fill in Blank (b)

```
BaseAttack + a.MoveAttack + b.MoveAttack
```

(c) (2.0 pt) Select all options that could fill in Blank (c).

- ☐ `Pokemon, Moves`
- ☐ `Pokemon, Pokemon, Moves`
- ☐ `Pokemon, Moves, Moves`
- ☐ `Pokemon as p, Moves as a`
- ☐ `Pokemon as p, Pokemon as q, Moves as a`
- ■ `Pokemon as p, Moves as a, Moves as b`

(d) (2.0 pt) Fill in Blank (d)

```
p.Move1 = a.Name AND p.Move2 = b.Name
```

10. A+: Treeing up the Wrong Bark

This A+ question is not worth any points. It can only affect your course grade if you have a high A and might receive an A+. Finish the rest of the exam first!

Write a function `table_to_tree` that takes a list `parents` of parent-child relationships between dogs. Each element of this list is a two-element list of strings containing the name of the parent and the name of the child, as well as `ancestor`, the name of the very first dog. It returns a `Tree` instance with root label `ancestor` that has each dog's name as a label and represents all parent-child pairs in `parents` as parent-child relations between nodes. Assume that:

- All dogs have unique names and are descendents of `ancestor` (or `ancestor` itself).
- All dogs have exactly one parent (*John has some weird dogs*) except for `ancestor`, which has none.
- Any order of branches is acceptable.

For example: `table_to_tree([['A','B'],['A','C'],['D','H'],['F','A'],['F','D'],['F','G'],['E','F']],'E')` could return:

```
Tree('E', [Tree('F', [Tree('A', [Tree('B'), Tree('C')]), Tree('D', [Tree('H')]), Tree('G')])])
```

```
def table_to_tree(parents, ancestor):
    dogs = {}
    def add_if_new(dog):
        if __(a)__:
            dogs[dog] = __(b)__
    for i, j in __(c)__:
        add_if_new(i)
        add_if_new(j)
        __(d)__
    __(e)__
```

(a) Fill in Blank (a)

```
not dogs.get(dog)
```

(b) Fill in Blank (b)

```
Tree(dog)
```

(c) Fill in Blank (c)

```
parents
```

(d) Fill in Blank (d)

```
dogs[i].branches.append(dogs[j])
```

(e) Fill in Blank (e)

```
return dogs[ancestor]
```

11. A+: Another Parentheses Scheme

This A+ question is not worth any points. It can only affect your course grade if you have a high A and might receive an A+. Finish the rest of the exam first!

After you finish writing your parentheses remover, you run it on all the Scheme code you have. You then realize that one of those programs was an ongoing assignment. Uh oh! Fortunately, your code for that assignment only contains two-argument procedures. Write a Scheme macro `eval-noparens`. This macro receives as input a list representing a line of Scheme code with all the parentheses removed (except the parentheses surrounding the entire list), and returns the result that would have been obtained by evaluating the original line of Scheme code. You are guaranteed that:

- All elements of the list are either numbers or symbols bound to two-input procedures.
- All original call expressions (before parentheses were removed) had exactly two operands.
- The list corresponds to exactly one valid line of Scheme code fulfilling the above conditions.

For credit, your solution must run in linear time in the length of its input.

Hint: Treat the `car` and `cdr` of the return value of `helper` as two independent outputs.

```
;;; > (eval-noparens (+ 1 * 2 3))
;;; 7
;;; > (eval-noparens (+ * 1 2 + 3 * 4 + + + 5 6 7 8))
;;; 109
;;; > (eval-noparens (cons 1 list 2 3))
;;; (1 2 3)
;;; > (let ((times *)) (eval-noparens (+ 1 times 2 3)))
;;; 7
(define-macro (eval-noparens expr)
  (define (helper expr)
    (if (number? (car expr)) expr
      (let ((x __(a)__ ))
        (let ((y __(b)__ ))
          __(c)__ ))))
  (car (helper expr)))
```

(a) Fill in Blank (a)

```
(helper (cdr expr))
```

(b) Fill in Blank (b)

```
(helper (cdr x))
```

(c) Fill in Blank (c)

```
(cons (list (car expr) (car x) (car y)) (cdr y))
```

12. A+: Pokemon Ancient SQarLet

This A+ question is not worth any points. It can only affect your course grade if you have a high A and might receive an A+. Finish the rest of the exam first!

As a gym leader, your team must be composed of Pokemon of a single type. You decide to count how many Pokemon of each type you have. Generate a table with two columns: All unique types in your collection and the number of Pokemon you have of each type, sorted in descending order by number of Pokemon.

Each Pokemon can have either one or two types, listed under columns `Type1` and `Type2`.

- If a Pokemon has one type, then `Type1` stores the type of the Pokemon, and `Type2` stores the string `"None"`.
- If a Pokemon has two types, then `Type1` stores the first type of the Pokemon, and `Type2` stores the second type of the Pokemon (which is distinct from the first type).
- A Pokemon with two types counts as both of its types.

The table `Pokemon` is reprinted here for convenience. **All Pokemon have unique names.**

Name	Type1	Type2	Move1	Move2	BaseAttack	BaseDefense
Rattata	Normal	None	Endeavor	Quick Attack	0	0
Maushold	Normal	None	Population Bomb	Tidy Up	30	30
Cyclizard	Dragon	Normal	Shed Tail	Dragon Rush	100	40
Meoscadera	Grass	Dark	Flower Trick	Agility	80	40
Farigiraf	Normal	Psychic	Agility	Twin Beam	40	50

For this table, the following would be the expected output (With the last four rows in any order):

Type	Count
Normal	4
Dragon	1
Grass	1
Dark	1
Psychic	1

```
CREATE TABLE helper AS SELECT __(a)__;
SELECT Type, COUNT(*) AS Count FROM helper __(b)__;
```

(a) Fill in Blank (a)

```
Name, Type1 AS Type FROM Pokemon UNION SELECT Name, Type2 FROM Pokemon
WHERE Type != "None"
```

(b) Fill in Blank (b)

```
GROUP BY Type ORDER BY count(*) DESC;
```

13. (0.0 points) OPTIONAL

The following questions will not affect your score in any way.

(a) (0.0 pt) Which of the Pokemon in the SQL question is holding an item, and what item is it?

Rattata is holding a Focus Sash.

(b) (0.0 pt) Free Space: Write/Draw anything you want!

No more questions.