COMP1811 – Scheme Project Report

Name	Brendon Shymanskyi	SID	001419181
Partner	Mykhaylo Zhyhan	SID	001398148

1. Software Design

Briefly describe the design of your coursework software and the strategy you took for solving it. – e.g. did you choose either recursion or high-order programming and why …

We were given two branches(lists) with names - maternal and paternal branches. The lists contained names and surnames and other information such as parents, date of birth and date of death. Looking at the specification, we could immediately understand that we should use either Higher-Order functions or recursion.

My partner and I decided to use Higher-Order functions, as it seemed more convenient and clearer, and, in our opinion, it would give us more options. One of the design decisions and part of Higher-Order functions was to create helper functions that we used in many of our functions. Our helper functions performed such actions as displaying a person's name, filtering a list, comparing dates, removing duplicates. They could make specific operations like sorting by surname or filtering in branches.

For displaying the output we decided to make a separate functions like we did in Python coursework. The function "display-menu" contains a text to display it to user, and "command-loop" contains options, condition under which you can enter the numbers to which the options belong, so that you can run the previously recorded functions, loop and a piece of code that do not allow the application to crash if input is not correct.

We have divided our code into sections to make it much clearer and easier to understand. Sections: Helper Functions, Main Functions, Command Functions.

We also found a better way for C1, C2, and C3 using our helper function 'display-name'. We implemented all three tasks in one just because our helper function can take any lists and more than one at a time.

Using "let" in functions – we knew that we hadn't learnt this function in the lecture, but when we found it during a research, we wanted to try it because it seemed like a pretty interesting experience. "let" is somewhat similar to lambda in some cases and can have a similar result depending on the structure of functions, but we use "let" mostly when we need to bind values for a block of code in a function without creating a new one.

2. Use of Functional Programming Techniques

For each of the techniques below, list where you have use it and why you used it. This is your chance to convince us that you understand the concepts. Give a clear exposition of your design strategy.

2.1 Appropriate Use of Lambda Functions

B1: lst (lambda (entry)

(and (not (null? (cadr entry)))

I used lambda here to check if an entry represents a child with at least one existing parent, it looks at the second element of each entry.

B2: (let ((alive-members (filter (lambda (entry) (null? (cadr (caddr entry)))) lst)))

(if (null? alive-members)

Lambda here filters out deceased members by checking if their date of death is null.

B2: (let ((oldest (car (sort alive-members (lambda (a b) (date<? (car (caddr a)) (car (caddr b))))))))

Sorts living members by date, the oldest comes first.

B3: (let ((deceased (filter (lambda (entry) (not (null? (cadr (caddr entry))))) lst)))

Filters out deceased members by checking date of death.

B3: (let ((total-age (apply + (map (lambda (entry) (- (caddr (cadr (caddr entry))) (caddr (car (caddr entry)))))) deceased))))

Here lambda calculates each persons age at death.

B4: (lambda (entry) (= (cadr (car (caddr entry))) month))

Checks if the person's birth month is matches given month.

B5: (let ((sorted (sort lst (lambda (a b) (string<? (symbol->string (caar a)) (symbol->string (caar b)))))))

Compares the first names of two members for sorting.

2.2 Appropriate Use Recursion

We have used recursion in our helper function:

```
(define (remove-duplicates lst)
  (if (null? lst)  ; base case: returns empty list if it's empty
      '()
```

```
    (cons (car lst)  ; includes the first emelent of the list in the result of the next line
        (remove-duplicates (filter (lambda (x) (not (equal? x (car lst)))) (cdr lst))))))  ; recursive case: process the rest
```

It’s recursively processes the list after filtering duplicates.

Another function with recursion is the “command-loop":

```
(define (command-loop)
  (let loop ()
    (display-menu)
    (let ((input-command (read)))
      (if (not (number? input-command))
          (begin
            (display "Invalid command! Try again.\n")
            (loop))
          (cond
            ((= input-command 1) (display-names Mb)) ; C1
            ((= input-command 2) (display-names Pb)) ; C2
            ((= input-command 3) (display-names (append Mb Pb))) ; C3
            ((= input-command 4) (parents Mb)) ; A1
            ((= input-command 5) (living-members Mb)) ; A2
            ((= input-command 6) (current-age (append Mb Pb))) ; A3
            ((= input-command 7) (same-birthday-month (append Mb Pb) 10)) ; A4
            ((= input-command 8) (sort-by-last (append Mb Pb))) ; A5
            ((= input-command 9) (change-name-to-Juan (append Mb Pb) 'John 'Juan)) ; A6
            ((= input-command 10) (children Pb)) ; B1
            ((= input-command 11) (oldest-living-member Pb)) ; B2
            ((= input-command 12) (average-age-on-death Pb)) ; B3
            ((= input-command 13) (birthday-month-same (append Mb Pb) 1)) ; B4
            ((= input-command 14) (sort-by-first Pb)) ; B5
            ((= input-command 15) (change-name-to-Maria (append Mb Pb) 'Mary 'Maria)) ; B6
            (else (display "Invalid command! Try again.\n"))))
      (loop))))
```

Here function uses recursion to continuously request input, and each call to the loop represents a new command line input.

2.3 Appropriate Use of Higher Order Functions

We have helper functions that also contain higher order functions:

1) **filter-and-display**.

Higher-Order Function:

1. filter – uses to filter and keep only the necessary elements that satisfy the given predicate.

Code:

```
(define (filter-and-display lst filter-members message)
  ; defining a variable 'filtered'; stores a result
  ; the 'filter' function applies 'filter-members' to each element of list and keeps only the necessary elements.
  (let ((filtered (filter filter-members lst)))
    (if (null? filtered)
        (display (string-append "No " message " found.\n")) ; if empty - no 'message' found.
        (begin
          (display (string-append message ":\n"))
          (display-names filtered)))))
```

2) **remove-duplicates.**

Higher-Order Function:

1. filter – uses to filter out duplicates and keeps only the first occurrence of each element.

Code:

```
(define (remove-duplicates lst)
  (if (null? lst)  ; base case: returns empty list if it's empty
      '()
      (cons (car lst)  ; includes the first emelent of the list in the result of the next line
            (remove-duplicates (filter (lambda (x) (not (equal? x (car lst)))) (cdr lst))))))  ; recursive case: process the rest
```

B1: children.

The “children” function uses a helper function “filter-and-display" to filter entries where the person has at least one parent.

Code:

```
(define (children lst)
  (filter-and-display
   lst
   (lambda (entry)
```

```
    ; Check if the person has at least one parent (mother or father)
    (and (not (null? (cadr entry)))
        ; checks whether mother or father exists
        (or (not (null? (caadr entry)))
            (not (null? (cadadr entry))))))
  "Children"))
```

B2: oldest-living-member.

Higher-Order Functions:

1. filter – filters out members who does not have date of death;
2. sort – sorts the living members by date of birth, oldest goes first.

Code:

```
(define (oldest-living-member lst)
  ; Filter the list to include only living members (those with no date of death)
  (let ((alive-members (filter (lambda (entry) (null? (cadr (caddr entry)))) lst)))
    (if (null? alive-members)
        (display "No living members found.\n")
        ; If living members exist, find the oldest and sort by date of birth (oldest first)
        (let ((oldest (car (sort alive-members (lambda (a b) (date<? (car (caddr a)) (car (caddr b))))))))
          (display "Oldest living member: ")
          (display-name (car oldest))))))
```

B3: average-age-on-death

Higher-Order Functions:

1. map - applies function to all deceased people and returns a list of their ages;
2. apply + - sums the ages at death.

Code:

```
(define (average-age-on-death lst)
  ; Filter the list to include only deceased members (those with a date of death)
  (let ((deceased (filter (lambda (entry) (not (null? (cadr (caddr entry))))) lst)))
    (if (null? deceased)
        (display "No deceased members found.\n")
        ; If deceased members exist, calculate the average age at death. Map over deceased members to calculate their age at death.
        (let ((total-age (apply + (map (lambda (entry) (- (caddr (cadr (caddr entry))) (caddr (car (caddr entry))))) deceased))))
          (display "Average age at death: ")
```

(display (/ (exact->inexact total-age) (length deceased))) ;; Dividing the total age by the number of deceased members

(newline)))))

B4: birthday-month-same.

The “birhday-month-same" function uses a helper function “filter-and-display" to check if the birth month matches the specified month.

Code:

```
;; B4: Function to find members with birthdays in January
(define (birthday-month-same lst month)
  ; Use filter-and-display to filter members with birthdays in the month of January.
  (filter-and-display lst
                      ; Lambda function to check if the member's birth month matches the month of January.
                      (lambda (entry) (= (cadr (car (caddr entry))) month)) "Members with birthdays in January"))
```

B5: sort-by-first.

Higher-Order Function:

1. sort – sorts the list by first name in ascending order.

Code:

```
(define (sort-by-first lst)
  ; Sort the list by first name in ascending order using included comparison function
  (let ((sorted (sort lst (lambda (a b) (string<? (symbol->string (caar a)) (symbol->string (caar b)))))))
    (display "Sorted members by first name:\n")
    (display-names sorted)))
```

B6: change-name-to-Maria.

Higher-Order Function:

1. map – updates if the name matches “Mary” and replaces this name with “Maria”.

Code:

```
(define (change-name-to-Maria lst old-name new-name)
  ; Helper function to update the name if it matches 'old-name'
  (define (update-name name)
    ; Check if the first name matches 'old-name'. And replace an old name with a new name.
    (if (eq? (car name) old-name) (cons new-name (cdr name)) name))
  ; Helper function to update an entry
```

```
(define (update-entry entry)
  ; Update the name              | Keep the rest unchanged
  (cons (update-name (car entry)) (cdr entry)))
(let ((updated (map update-entry lst))) ; map
  (display "Updated list with 'Mary' changed to 'Maria':\n")
  (display-names updated)))
```

3. Annotated Screenshots Demonstrating Implementation

Provide screenshots that demonstrate the results generated by running your code. That is show the output obtained in the REPL when calling your functions. Alternatively, you may simply cut and paste from the REPL. Please annotate all screenshots to say what they show. Briefly describe what each shows.

3.1 TASK 1

B1 output: all children from paternal branch.

```
Enter command number: 10
Children:
John Smith
Ana Smith
Jane Doe
Fred Smith
Alan Doe
Mary Doe
```

3.2 TASK 2

B2 output: oldest living member from paternal branch.

```
Enter command number: 11
Oldest living member: Ana Smith
```

3.3 TASK 3

B3 displaying average age at death in the paternal branch.

```
Enter command number: 12
Average age at death: 74.75
```

3.4 TASK 4

B4 displaying only members from the whole tree with birthdays in January.

```
Enter command number: 13
Members with birthdays in January:
Tom Blake
```

3.5 TASK 5

B5 displaying sorted members by their first name from paternal branch.

```
Enter command number: 14
Sorted members by first name:
Alan Doe
Ana Smith
Eve Talis
Fred Smith
Jane Doe
John Smith
John Doe
Lisa Brown
Mary Doe
Tom Smith
```

3.6 TASK 6

B6 displaying modified family tree with the name “Mary” changed to “Maria”.

```
Enter command number: 15
Updated list with 'Mary' changed to 'Maria':
Maria Blake
Ana Ali
Theo Blake
Greta Blake
Maria Jones
Tom Blake
Ada West
Md Ali
Ned Bloom
John Bloom
John Smith
Ana Smith
Jane Doe
Fred Smith
Eve Talis
John Doe
Lisa Brown
Tom Smith
Alan Doe
Maria Doe
```

4. TESTING

Provide a test plan covering all of your functions and the results of applying the tests. Use the data analysed by your partner to test your code.

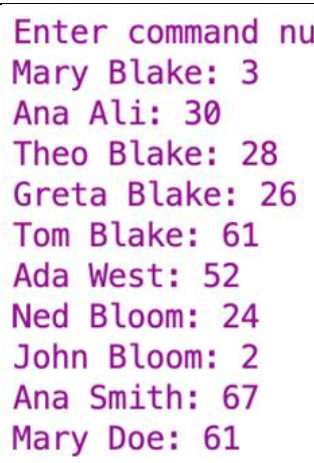
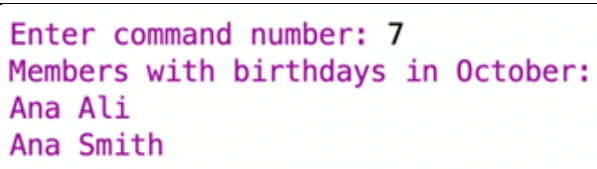
Test No.	What is being tested?	Expected output	Screenshot of actual results	Pass/Fail
1	B1 / command "10"	All the children from paternal branch: John Smith Ana Smith Jane Doe Fred Smith Alan Doe Mary Doe	Children: John Smith Ana Smith Jane Doe Fred Smith Alan Doe Mary Doe	
2	B2 / command "11"	Oldest living member: Ana Smith	Oldest living member: Ana Smith	
3	B3 / command "12"	Average age at death: 75	Average age at death: 75	
4	B4 / command "13"	Members with birthdays in January: Tom Blake	Members with birthdays in January: Tom Blake	

5	B5 / command “14”	Sorted members by first name in alphabetical order: Alan Doe Ana Smith Eve Talis Fred Smith Jane Doe John Smith John Doe Lisa Brown Mary Doe Tom Smith	Sorted members by first name: Alan Doe Ana Smith Eve Talis Fred Smith Jane Doe John Smith John Doe Lisa Brown Mary Doe Tom Smith	

6	B6 / command “15”	Updated family tree with 'Mary' changed to 'Maria': Maria Blake Ana Ali Theo Blake Greta Blake Maria Jones Tom Blake Ada West Md Ali Ned Bloom John Bloom John Smith Ana Smith Jane Doe Fred Smith Eve Talis John Doe Lisa Brown Tom Smith Alan Doe Maria Doe	Updated list with 'Mary' changed to 'Maria': Maria Blake Ana Ali Theo Blake Greta Blake Maria Jones Tom Blake Ada West Md Ali Ned Bloom John Bloom John Smith Ana Smith Jane Doe Fred Smith Eve Talis John Doe Lisa Brown Tom Smith Alan Doe Maria Doe	

7	Testing B1 function with partner's branch (Maternal) / command "10"	Mary Blake Ana Ali Theo Blake Greta Blake John Bloom	Children: Mary Blake Ana Ali Theo Blake Greta Blake John Bloom	
8	B2 function with partner's branch (Maternal) / command "11"	Should return the oldest living member: Tom Blake	Oldest living member: Tom Blake	
9	B3 function with partner's branch (Maternal) / command "12"	Return's average age at death in Maternal branch: 54	Average age at death: 54.0	
10	B5 function with partner's branch (Maternal) / command "14"	Should return sorted members by first name from Maternal branch: Ada West Ana Ali Greta Blake John Bloom Mary Blake Mary Jones Md Ali Ned Bloom	Ada West Ana Ali Greta Blake John Bloom Mary Blake Mary Jones Md Ali Ned Bloom Theo Blake Tom Blake	

		Theo Blake Tom Blake		
11	Testing B4 function with another selected month – November / command “13”	No members found with birthdays in November	No Members with birthdays in November found.	
12	Testing function A1 using Maternal branch / command “4”	Names of all parents in maternal branch: Ana Ali Theo Blake Greta Blake Mary Jones Tom Blake Ada West Md Ali Ned Bloom	Enter command number: 4 Parents in maternal branch: Ana Ali Theo Blake Greta Blake Mary Jones Tom Blake Ada West Md Ali Ned Bloom	
13	A2 using maternal branch / command “5” - display all living members	Names of all living members of maternal branch: Mary Blake Ana Ali Theo Blake Greta Blake Tom Blake Ada West	Enter command number: 5 Living members: Mary Blake Ana Ali Theo Blake Greta Blake Tom Blake Ada West Ned Bloom John Bloom	

		Ned Bloom John Bloom		
14	A3 using maternal branch / command ”6”	Ages of all living members Mary Blake: 3 Ana Ali: 30 Theo Blake: 28 Greta Blake: 26 Tom Blake: 61 Ada West: 52 Ned Bloom: 24 John Bloom: 2 Ana Smith: 67 Mary Doe: 61		
15	A4 using maternal branch / command “7”	Members with birthdays in October Ana Ali Ana Smith		

16	A5 using maternal branch / command “8”	Living members: Mary Blake Ana Ali Theo Blake Greta Blake Tom Blake Ada West Ned Bloom John Bloom	Living members: Mary Blake Ana Ali Theo Blake Greta Blake Tom Blake Ada West Ned Bloom John Bloom	
17	A6 using maternal branch / command “9”	Mary Blake Ana Ali Theo Blake Greta Blake Mary Jones Tom Blake Ada West Md Ali Ned Bloom Juan Bloom Juan Smith Ana Smith Jane Doe Fred Smith Eve Talis Juan Doe Lisa Brown Tom Smith Alan Doe Mary Doe	Updated list with 'John' changed to 'Juan': Mary Blake Ana Ali Theo Blake Greta Blake Mary Jones Tom Blake Ada West Md Ali Ned Bloom Juan Bloom Juan Smith Ana Smith Jane Doe Fred Smith Eve Talis Juan Doe Lisa Brown Tom Smith Alan Doe Mary Doe	

18	A1 using paternal branch / command “4”	Jane Doe Fred Smith Eve Talis John Doe Lisa Brown Tom Smith Alan Doe	Enter command number: 4 Parents in paternal branch: Jane Doe Fred Smith Eve Talis John Doe Lisa Brown Tom Smith Alan Doe	
19	A2 using paternal branch / command “5”	Ana Smith Mary Doe	Enter command number: 5 Living members in Paternal branch: Ana Smith Mary Doe	
20	A4 using paternal branch / command “7”	No members found with birthdays in November.	Enter command number: 7 No Members with birthdays in November found.	
21	A5 using paternal branch / command “8”	Lisa Brown Jane Doe John Doe Alan Doe Mary Doe John Smith Ana Smith Fred Smith Tom Smith Eve Talis	Enter command number: 8 Sorted members by last name: Lisa Brown Jane Doe John Doe Alan Doe Mary Doe John Smith Ana Smith Fred Smith Tom Smith Eve Talis	

22	Wrong input / command “16”	Invalid command	Enter command number: 16 Invalid command! Try again.	
23	Wrong input / command “twelve”	Invalid command	Enter command number: twelve Invalid command! Try again.	

5. Self-Assessment

*Being able to assess your performance **objectively** is a skill needed in the professional world. The criteria are not just about have you completed a task, but more about how well you have completed it.*

Some points for reflection might be:

- *Would an employer be happy with this?*
- *Could a user who was unfamiliar with the problem understand your work?*
- *Is the work easy to read?*
- *Are the methods used justified?*
- *Are the methods used sensible?*
- *Etc.*

Please note that you are only responsible for estimating the score for your features.

Partner A	Partner B	Out of	My estimated score
Parents	Children	3	3
Living members	Oldest member	4	4
Current age	Average age on death	5	5
Same birthday	Same birthday	8	8
Sort by last name	Sort by first name	10	10
Change a name	Change a name	10	10
Appropriate use of lambda functions		5	5
Appropriate use of recursion		7	6
Appropriate use of higher order functions		8	8
Quality of code		10	9
Screencast		10	8
Testing		10	8
Documentation		10	8
TOTAL		**100**	**92**

6. APPENDIX A – LISTING OF THE CODE FILE (NOT SCREENSHOTS!)

```
;;Name: Brendon Shymanskyi

;;Student ID: 001419181

;;Month of Birth: January


;; Data format: Name, Mother, Father, Date of birth, Date of death.

;; An empty list means Unknown.


;; Maternal branch

(define Mb

  '(((Mary Blake) ((Ana Ali) (Theo Blake)) ((17 9 2022) ()))

    ((Ana Ali) ((Ada West) (Md Ali)) ((4 10 1995) ()))

    ((Theo Blake) ((Mary Jones) (Tom Blake)) ((9 5 1997) ()))

    ((Greta Blake) ((Mary Jones) (Tom Blake)) ((16 3 1999) ()))

    ((Mary Jones) (() ()) ((12 5 1967) (19 5 2024)))

    ((Tom Blake) (() ()) ((17 1 1964) ()))

    ((Ada West) (() ()) ((22 8 1973) ()))

    ((Md Ali) (() ()) ((14 2 1972) (2 5 2023)))

    ((Ned Bloom) (() ()) ((23 4 2001) ()))

    ((John Bloom) ((Greta Blake) (Ned Bloom)) ((5 12 2023) ()))))


;; Paternal branch

(define Pb

  '(((John Smith) ((Jane Doe) (Fred Smith)) ((1 12 1956) (3 3 2021)))

    ((Ana Smith) ((Jane Doe) (Fred Smith)) ((6 10 1958) ()))

    ((Jane Doe) ((Eve Talis) (John Doe)) ((2 6 1930) (4 12 1992)))

    ((Fred Smith) ((Lisa Brown) (Tom Smith)) ((17 2 1928) (13 9 2016)))

    ((Eve Talis) (() ()) ((15 5 1900) (19 7 1978)))

    ((John Doe) (() ()) ((18 2 1899) (7 7 1970)))

    ((Lisa Brown) (() ()) ((31 6 1904) (6 3 1980)))

    ((Tom Smith) (() ()) ((2 8 1897) (26 11 1987)))

    ((Alan Doe) ((Eve Talis) (John Doe)) ((8 9 1932) (23 12 2000)))

    ((Mary Doe) (() (Alan Doe)) ((14 4 1964) ()))))
```

```
;;------|Helper Functions|------;;


;; function to display names

(define (display-name name)

  (display (string-append (symbol->string (car name)) " " (symbol->string (cadr name))))  ; converts the first two elements of the list from symbols to strings and making a space between them

  (newline))


;; filter-and-display

(define (filter-and-display lst filter-members message)

  ; defining a variable 'filtered'; stores a result

  ; the 'filter' function applies 'filter-members' to each element of list and keeps only the necessary elements.

  (let ((filtered (filter filter-members lst)))

    (if (null? filtered)

        (display (string-append "No " message " found.\n")) ; if empty - no 'message' found.

        (begin

          (display (string-append message ":\n"))

          (display-names filtered)))))


;; compare dates (year, month, day)

(define (date<? date1 date2)

  (or (< (caddr date1) (caddr date2))  ; compare years

      (and (= (caddr date1) (caddr date2))  ; if years are equal - compare months

           (or (< (cadr date1) (cadr date2))

               (and (= (cadr date1) (cadr date2))  ; if months are equal - compare days

                    (< (car date1) (car date2)))))))


;; removes duplicates | recursive function

(define (remove-duplicates lst)

  (if (null? lst)  ; base case: returns empty list if it's empty

      '()

      (cons (car lst)  ; includes the first emelent of the list in the result of the next line
```

```
        (remove-duplicates (filter (lambda (x) (not (equal? x (car lst)))) (cdr lst))))))  ; recursive case: process the rest


;;------|Main Functions|------;;10


;; C1, C2, C3:
(define (display-names lst)
  ; displaying all people in the given list or combined lists
  (for-each (lambda (entry) (display-name (car entry))) lst))


;; ------------------------


;; A1: Function to display parents from a given list
(define (parents lst)
  (let ((parent-names
         (remove-duplicates
          (append (map caadr lst)  ; extract mothers
                  (map cadadr lst)))))  ; extract fathers
    (filter-and-display
     lst
     (lambda (entry)
       (member (car entry) parent-names))  ; checks if is a parent
     "Parents in maternal branch")))


;; A2: Function to filter and display living members
(define (living-members lst)
  (filter-and-display
   lst
   (lambda (entry) (null? (cadr (caddr entry))))  ; checks if member is alive
   "Living members"))  ; message to display


;; A3: Current age (living members only)
(define (current-age lst)
  (let ((current-year 2025))
```

```
    (for-each
     (lambda (entry)
       (let* ((dob-info (car (caddr entry)))   ; extract date of birth
              (dod-info (cadr (caddr entry)))  ; extract date of death
              (birth-year (caddr dob-info)))  ; birth year
         (when (null? dod-info)  ; only show for living members
           (display
            (string-append
             (symbol->string (caar entry))  ; First name
             " "
             (symbol->string (cadar entry))  ; Last name
             ": "
             (number->string (- current-year birth-year))
             "\n")))))
     lst)))

;; A4: Function to find members with birthdays in October
(define (same-birthday-month lst month)
  (filter-and-display
   lst
   (lambda (entry) (= (cadr (car (caddr entry))) month))  ; checks birthday month
   "Members with birthdays in October"))  ; message

;; A5: Function to sort members by last name
(define (sort-by-last lst)
  (let ((sorted (sort lst
                      (lambda (a b)
                        (string<? (symbol->string (cadr (car a)))  ; comparing last names
                                  (symbol->string (cadr (car b))))))))
    (display "Sorted members by last name:\n")
    (display-names sorted)))

;; A6: Function to change "John" to "Juan" recursively in any list structure
```

```
(define (change-name-to-Juan lst old-name new-name)
  ; helper function to update the name if it matches 'old-name'
  (define (update-name name)
    (if (eq? (car name) old-name) (cons new-name (cdr name)) name))  ; combining with new name if the old name not matches
  ; helper to update an entry
  (define (update-entry entry)
    (cons (update-name (car entry)) (cdr entry)))  ; update the new part of the entry
  (let ((updated (map update-entry lst))) ; map: applies update-entry to each entry in lst.
    (display "Updated list with 'John' changed to 'Juan':\n")
    (display-names updated)))


;; ------------------------


;; B1: Display all children from list
(define (children lst)
  (filter-and-display
   lst
   (lambda (entry)
     ; Check if the person has at least one parent (mother or father)
     (and (not (null? (cadr entry)))
          ; checks whether mother or father exists
          (or (not (null? (caadr entry)))
              (not (null? (cadadr entry))))))
   "Children"))


;; B2: Function to find the oldest living member
(define (oldest-living-member lst)
  ; Filter the list to include only living members (those with no date of death)
  (let ((alive-members (filter (lambda (entry) (null? (cadr (caddr entry)))) lst)))
    (if (null? alive-members)
        (display "No living members found.\n")
        ; If living members exist, find the oldest and sort by date of birth (oldest first)
        (let ((oldest (car (sort alive-members (lambda (a b) (date<? (car (caddr a)) (car (caddr b))))))))
```

```
        (display "Oldest living member: ")
        (display-name (car oldest))))))

;; B3: Function to calculate average age at death
(define (average-age-on-death lst)
  ; Filter the list to include only deceased members (those with a date of death)
  (let ((deceased (filter (lambda (entry) (not (null? (cadr (caddr entry))))) lst)))
    (if (null? deceased)
        (display "No deceased members found.\n")
        ; If deceased members exist, calculate the average age at death. Map over deceased members to calculate their age at death.
        (let ((total-age (apply + (map (lambda (entry) (- (caddr (cadr (caddr entry))) (caddr (car (caddr entry))))) deceased))))
          (display "Average age at death: ")
          (display (/ (exact->inexact total-age) (length deceased))) ;; Dividing the total age by the number of deceased members
          (newline)))))

;; B4: Function to find members with birthdays in January
(define (birthday-month-same lst month)
  ; Use filter-and-display to filter members with birthdays in the month of January.
  (filter-and-display lst
                      ; Lambda function to check if the member's birth month matches the month of January.
                      (lambda (entry) (= (cadr (car (caddr entry))) month)) "Members with birthdays in January"))

;; B5: Sort members by first name
(define (sort-by-first lst)
  ; Sort the list by first name in ascending order using included comparison function
  (let ((sorted (sort lst (lambda (a b) (string<? (symbol->string (caar a)) (symbol->string (caar b)))))))
    (display "Sorted members by first name:\n")
    (display-names sorted)))

;; B6: Change "Mary" to "Maria"
(define (change-name-to-Maria lst old-name new-name)
  ; Helper function to update the name if it matches 'old-name'
  (define (update-name name)
```

```
    ; Check if the first name matches 'old-name'. And replace an old name with a new name.
    (if (eq? (car name) old-name) (cons new-name (cdr name)) name))
  ; Helper function to update an entry
  (define (update-entry entry)
    ; Update the name            | Keep the rest unchanged
    (cons (update-name (car entry)) (cdr entry)))
  (let ((updated (map update-entry lst))) ; map
    (display "Updated list with 'Mary' changed to 'Maria':\n")
    (display-names updated)))

;;------|Command Functions|------;;

(define (display-menu)
  (display "\nAvailable commands:\n")
  (display "1. Display members from Maternal Branch.\n")
  (display "2. Display members from Paternal Branch.\n")
  (display "3. Display members from both branches.\n")
  (display ">Tasks A:\n4. Display all parents in the Maternall branch.\n")
  (display "5. Display all Living members from Maternal branch.\n")
  (display "6. Display current age of all people alive.\n")
  (display "7. Display members with birthdays in October.\n")
  (display "8. Display sorted people by last name in Maternal branch.\n")
  (display "9. Change all John names to Juan.\n")
  (display ">Tasks B:\n10. Display all children from Paternal branch.\n")
  (display "11. Display the oldest living member.\n")
  (display "12. Calculate average age at death.\n")
  (display "13. Display members with birthdays in January.\n")
  (display "14. Sort members by first name.\n")
  (display "15. Change all Mary names to Maria.\n")
  (newline)
  (display "Enter command number: "))

(define (command-loop)
```

```
(let loop ()
  (display-menu)
  (let ((input-command (read)))
    (if (not (number? input-command))
        (begin
          (display "Invalid command! Try again.\n")
          (loop))
        (cond
          ((= input-command 1) (display-names Mb)) ; C1
          ((= input-command 2) (display-names Pb)) ; C2
          ((= input-command 3) (display-names (append Mb Pb))) ; C3
          ((= input-command 4) (parents Mb)) ; A1
          ((= input-command 5) (living-members Mb)) ; A2
          ((= input-command 6) (current-age (append Mb Pb))) ; A3
          ((= input-command 7) (same-birthday-month (append Mb Pb) 10)) ; A4
          ((= input-command 8) (sort-by-last Mb)) ; A5
          ((= input-command 9) (change-name-to-Juan (append Mb Pb) 'John 'Juan)) ; A6
          ((= input-command 10) (children Pb)) ; B1
          ((= input-command 11) (oldest-living-member Pb)) ; B2
          ((= input-command 12) (average-age-on-death Pb)) ; B3
          ((= input-command 13) (birthday-month-same (append Mb Pb) 1)) ; B4
          ((= input-command 14) (sort-by-first Pb)) ; B5
          ((= input-command 15) (change-name-to-Maria (append Mb Pb) 'Mary 'Maria)) ; B6
          (else (display "Invalid command! Try again.\n"))))
    (loop))))



(command-loop)
```