資訊工程學系

程式語言與編譯器程式設計報告 II

學號:411121273

姓名:鄭宇翔

目錄：

The problem description.

Practice on programming the exercises in 5 different programming languages:

1. *Java*
2. *ML*
3. *Prolog*
4. *COBOL*
5. *R*

Programming Exercise 1:

Given the following facts:

Fact #1: Andy, Bob, Cecil, Dennis, Edward, Felix, Martin, Oscar, Quinn are male, and Gigi, Helen, Iris, Jane, Kate, Liz, Nancy, Pattie, Rebecca are female.
Fact #2: Bob and Helen are married, Dennis and Pattie are married, and Gigi and Martin are married.
Fact #3: Andy is Bob's parent, Bob is Cecil's parent, Cecil is Dennis' parent, Dennis is Edward's parent, Edward is Felix's parent, Gigi is Helen's parent, Helen is Iris' parent, Iris is Jane's parent, Jane is Kate's parent, Kate is Liz's parent, Martin is Nancy's parent, Nancy is Oscar's parent, Oscar is Pattie's parent, Pattie is Quinn's parent, and Quinn is Rebecca's parent.

Define the following relations in your program:

Relation #1:
If X and Y are married, and X is Z's parent, then Y is also Z's parent.
Relation #2:
If X is Y's parent, and X is Z's parent, then Y and Z are siblings.
Relation #3:
If X and Y are siblings, X is male, and Y is male, then X and Y are brothers.
Relation #4:
If X and Y are siblings, X is female, and Y is female, then X and Y are sisters.
Relation #5:
If W and X are siblings, W is Y's parent, and X is Z's parent, then Y and Z are cousins.

Requirement: Your program needs to answer the relationship of any two persons correctly.

For example: are Liz and Rebecca cousins?

A) Write a Java program for this exercise.

B) Write an ML program for this exercise.

C) Write a Prolog program for this exercise.

Programming Exercise 2:

There are 3 tables for this exercise:

Table #1 Student-Main:

the main table with Student ID, Name, and Payment Type.

Table #2 Fees:

the Amount of fees required for each Payment Type.

Table #3 Student-Payment:

the Amount paid by students before due.

Requirement: Your program needs to do the following computations correctly:

Computation #1:

the total amount received from students before due.

Computation #2:

list all the students that did not pay the required fees with the amount short.

A) Write a COBOL program to do the computations.

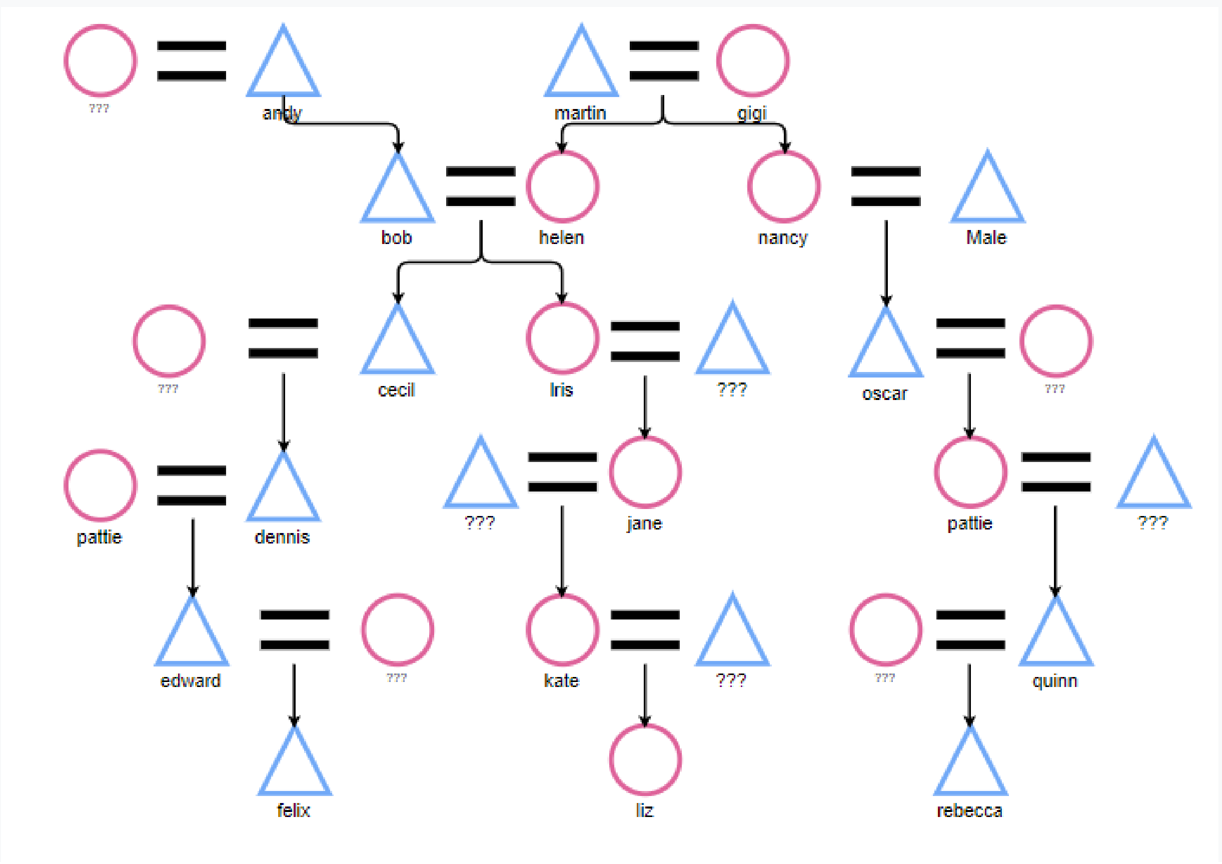
B) Write an R program to do the computations.

Highlight of the way you write the program.

Programming Exercise1

第一個練習題涉及到了有關人物關係的處理，我們的程式需要能夠判斷夫妻、親子、表親、兄弟、姊妹以及手足關係。

因題目所要求的人物關係複雜，因此在程式開發的過程中難以直觀的直接判斷程式運行的結果與邏輯是否正確，因此我製作了下圖的人物關係表，方便我們快速查看人物關係。



圖例:

藍色三角形符號代表男性

粉色圓形符號代表女性

等於符號代表夫妻關係

箭頭符號代表親子關係

圖形下若有文字則表示該人物的姓名

圖形下方若有三個問號表示題目並沒有給出人物資訊

Programming Exercise 1 with Java:

```java
import java.util.*;

class Person {
    String name;
    String gender;
    Person spouse;
    Person parent;
    List<Person> children;

    Person(String name, String gender) {
        this.name = name;
        this.gender = gender;
        this.children = new ArrayList<>();
    }

    void setSpouse(Person spouse) {
        this.spouse = spouse;
    }

    void addChild(Person child) {
        this.children.add(child);
        child.parent = this;
    }

    void addChildWithSpouse(Person child) {
        this.addChild(child);
        if (this.spouse != null) {
            this.spouse.addChild(child);
        }
    }

    static boolean areSiblings(Person p1, Person p2) {
        return p1.parent != null && p2.parent != null && p1.parent ==
p2.parent;
    }
    static boolean areBrothers(Person p1, Person p2) {
        return areSiblings(p1, p2) && p1.gender.equals("male") &&
p2.gender.equals("male");
    }

    static boolean areSisters(Person p1, Person p2) {
        return areSiblings(p1, p2) && p1.gender.equals("female") &&
p2.gender.equals("female");
    }

    static boolean areCousins(Person p1, Person p2) {
        if (p1.parent == null || p2.parent == null) return false;
        return areSiblings(p1.parent, p2.parent);
    }

    @Override
    public String toString() {
        return name;
    }
}

public class FamilyRelations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Define persons
        Map<String, Person> people = new HashMap<>();
        String[] males = {"Andy", "Bob", "Cecil", "Dennis", "Edward", "Felix",
"Martin", "Oscar", "Quinn"};
        String[] females = {"Gigi", "Helen", "Iris", "Jane", "Kate", "Liz",
"Nancy", "Pattie", "Rebecca"};

        for (String name : males) {
            people.put(name, new Person(name, "male"));
        }
        for (String name : females) {
            people.put(name, new Person(name, "female"));
}

// Define marriages
people.get("Bob").setSpouse(people.get("Helen"));
people.get("Dennis").setSpouse(people.get("Pattie"));
people.get("Gigi").setSpouse(people.get("Martin"));

// Define parent-child relationships
people.get("Andy").addChildWithSpouse(people.get("Bob"));
people.get("Bob").addChildWithSpouse(people.get("Cecil"));
people.get("Cecil").addChildWithSpouse(people.get("Dennis"));
people.get("Dennis").addChildWithSpouse(people.get("Edward"));
people.get("Edward").addChildWithSpouse(people.get("Felix"));
people.get("Gigi").addChildWithSpouse(people.get("Helen"));
people.get("Helen").addChildWithSpouse(people.get("Iris"));
people.get("Iris").addChildWithSpouse(people.get("Jane"));
people.get("Jane").addChildWithSpouse(people.get("Kate"));
people.get("Kate").addChildWithSpouse(people.get("Liz"));
people.get("Martin").addChildWithSpouse(people.get("Nancy"));
people.get("Nancy").addChildWithSpouse(people.get("Oscar"));
people.get("Oscar").addChildWithSpouse(people.get("Pattie"));
people.get("Pattie").addChildWithSpouse(people.get("Quinn"));
people.get("Quinn").addChildWithSpouse(people.get("Rebecca"));

// User input for checking relations
System.out.println("請輸入第一個人的名字：");
String name1 = scanner.nextLine();
System.out.println("請輸入第二個人的名字：");
String name2 = scanner.nextLine();

Person p1 = people.get(name1);
Person p2 = people.get(name2);

if (p1 == null || p2 == null) {
    System.out.println("輸入的名字不在列表中。");
    scanner.close();
    return;
}
```

```
        if (p1.spouse == p2) {
            System.out.println(p1 + " 和 " + p2 + " 是夫妻。");
        } else if (Person.areSiblings(p1, p2)) {
            System.out.println(p1 + " 和 " + p2 + " 是兄弟姐妹。");
            if (Person.areBrothers(p1, p2)) {
                System.out.println(p1 + " 和 " + p2 + " 是兄弟。");
            } else if (Person.areSisters(p1, p2)) {
                System.out.println(p1 + " 和 " + p2 + " 是姐妹。");
            }
        } else if (Person.areCousins(p1, p2)) {
            System.out.println(p1 + " 和 " + p2 + " 是表親。");
        } else {
            System.out.println(p1 + " 和 " + p2 + " 沒有直接關係。");
        }

        scanner.close();
    }
}
```

Describe:

這個程式定義了一個表示人的 Person 類別，並且實現了家庭關係的處理邏輯，包括夫妻、父母、兄弟姐妹、表親。

程式的主類 FamilyRelations 使用預先定義的一組人員來建立家庭樹，並通過使用者輸入來檢查兩個人之間的家庭關係。

類別

Person 類別

屬性

name: 人員的名字。

gender: 人員的性別。

spouse: 人員的配偶。

parent: 人員的父母。

children: 人員的子女列表。

建構子

Person(String name, String gender):

初始化人員的名字和性別、始化子女列表。

方法

setSpouse(Person spouse):

設定配偶。

addChild(Person child):

添加子女，並設定子女的父母為當前人員。

addChildWithSpouse(Person child):

將子女添加到當前人員及其配偶的子女列表中。

static boolean areSiblings(Person p1, Person p2):

判斷兩人是否為兄弟姐妹。

static boolean areBrothers(Person p1, Person p2):

判斷兩人是否為兄弟。

static boolean areSisters(Person p1, Person p2):

判斷兩人是否為姐妹。

static boolean areCousins(Person p1, Person p2):

判斷兩人是否為表親。

public String toString():

返回人員的名字。

FamilyRelations 類別

main

建立一個 Scanner 物件來讀取使用者輸入。

使用 Map 來儲存人員物件。

定義一組男性和女性人員，並將其添加到 Map 中。

設定婚姻關係。

定義父母與子女之間的關係。

根據使用者輸入，檢查兩人之間的家庭關係並輸出結果。

運行

程式首先建立人員物件並設置他們之間的家庭關係。

使用者輸入兩個人名。

程式根據輸入的名字在 Map 中查找對應的人員物件。

根據查找到的兩個人員物件，程式使用各種判斷方法來確定他們之間的關係，並輸出結果。

Programming Exercise 1 with ML:

```
datatype person = Andy | Bob | Cecil | Dennis | Edward | Felix | Martin | Oscar | Quinn
  | Gigi | Helen | Iris | Jane | Kate | Liz | Nancy | Pattie | Rebecca;

datatype gender = Male | Female;

datatype family =
    Parent of person * person
  | Married of person * person;

val males = [Andy, Bob, Cecil, Dennis, Edward, Felix, Martin, Oscar, Quinn];
val females = [Gigi, Helen, Iris, Jane, Kate, Liz, Nancy, Pattie, Rebecca];

val marriages = [Married(Bob, Helen), Married(Dennis, Pattie), Married(Gigi, Martin)];
val parentChild = [Parent(Andy, Bob), Parent(Bob, Cecil), Parent(Bob, Iris), Parent(Cecil,
Dennis), Parent(Dennis, Edward),
                   Parent(Edward, Felix), Parent(Gigi, Helen), Parent(Helen, Iris), Parent(Iris,
Jane),
                   Parent(Jane, Kate), Parent(Kate, Liz), Parent(Martin, Nancy), Parent(Nancy,
Oscar),
                   Parent(Oscar, Pattie), Parent(Pattie, Quinn), Parent(Quinn, Rebecca)];

fun isMale p = List.exists (fn x => x = p) males;
fun isFemale p = List.exists (fn x => x = p) females;

fun marriedTo p =
    case List.find (fn Married(x, y) => x = p orelse y = p | _ => false) marriages of
         SOME(Married(x, y)) => if x = p then y else x
       | NONE => p;

fun parentOf p = List.filter (fn Parent(x, _) => x = p | _ => false) parentChild;
fun childrenOf p = List.map (fn Parent(_, c) => c) (parentOf p);

fun findParents p = List.map (fn Parent(p, _) => p) (List.filter (fn Parent(_, y) => y = p)
parentChild);
```

```
fun siblings p =
    let
        val parentList = findParents p
        val siblingSets = List.concat (List.map childrenOf parentList)
    in
        List.filter (fn x => x <> p) siblingSets
    end;

fun areSiblings (x, y) =
    let
        val xParents = findParents x
        val yParents = findParents y
    in
        List.exists (fn px => List.exists (fn py => px = py) yParents) xParents
    end;

fun areBrothers (x, y) =
    areSiblings (x, y) andalso isMale x andalso isMale y;

fun areSisters (x, y) =
    areSiblings (x, y) andalso isFemale x andalso isFemale y;

fun grandparents p = List.concat (List.map findParents (findParents p));

fun areCousins (x, y) =
    let
        val xGrandparents = grandparents x
        val yGrandparents = grandparents y
    in
        List.exists (fn xp => List.exists (fn yp => xp = yp) yGrandparents) xGrandparents
        andalso not (areSiblings (x, y))
    end;

fun isParent (x, y) =
    List.exists (fn Parent(p, c) => p = x andalso c = y | _ => false) parentChild;

fun relationship (x, y) =
    if isParent (x, y) then "parent"
    else if isParent (y, x) then "child"
    else if List.exists (fn Married(a, b) => (a = x andalso b = y) orelse (a = y andalso b = x)) marriages then "spouse"
    else if areBrothers (x, y) then "brothers"
    else if areSisters (x, y) then "sisters"
    else if areSiblings (x, y) then "siblings"
    else if areCousins (x, y) then "cousins"
    else "no relation";

(* test *)
val _ = print("Relationship between Cecil and Iris: " ^ relationship (Cecil, Iris) ^ "\n");
val _ = print("Relationship between Liz and Kate: " ^ relationship (Liz, Kate) ^ "\n");
val _ = print("Relationship between Dennis and Jane: " ^ relationship (Dennis, Jane) ^ "\n");
val _ = print("Relationship between Helen and Iris: " ^ relationship (Helen, Iris) ^ "\n");
val _ = print("Relationship between Bob and Helen: " ^ relationship (Bob, Helen) ^ "\n");
val _ = print("Relationship between Gigi and Martin: " ^ relationship (Gigi, Martin) ^ "\n");
val _ = print("Relationship between Dennis and Pattie: " ^ relationship (Dennis, Pattie) ^ "\n");
```

Describe:

程式定義了若干資料型態來表示人員、性別以及家庭關係，並且包含了一些判斷家庭成員關係的函數。本文將逐步解釋程式的設計與實現，包括資料型態定義、資料初始化、以及關係判斷函數的邏輯。

資料型態定義

程式首先定義了以下三種資料型態：

person：

表示人員的資料型態，包括 Andy, Bob, Cecil 等。

gender：

表示性別的資料型態，包括 Male 和 Female。

family：

表示家庭關係的資料型態，包括 Parent（父母）和 Married（婚姻）

運行

我們可以使用 relationship 函數，並且輸入兩個的姓名，即可判斷兩人的關係。

Programming Exercise 1 with prolog:

```
male(andy).
male(bob).
male(cecil).
male(dennis).
male(edward).
male(felix).
male(martin).
male(oscar).
male(quinn).

female(gigi).
female(helen).
female(iris).
female(jane).
female(kate).
female(liz).
female(nancy).
female(pattie).
female(rebecca).

married(bob, helen).
married(dennis, pattie).
married(gigi, martin).

parent(andy, bob).
parent(bob, cecil).
parent(cecil, dennis).
parent(dennis, edward).
parent(edward, felix).
parent(gigi, helen).
parent(helen, iris).
parent(iris, jane).
parent(jane, kate).
parent(kate, liz).
parent(martin, nancy).
parent(nancy, oscar).
parent(oscar, pattie).
parent(pattie, quinn).
parent(quinn, rebecca).

parent(X, Z) :- married(X, Y), parent(Y, Z).

sibling(X, Y) :- parent(P, X), parent(P, Y), X \= Y.

brother(X, Y) :- sibling(X, Y), male(X).

sister(X, Y) :- sibling(X, Y), female(X).

cousin(X, Y) :- parent(P1, X), parent(P2, Y), sibling(P1, P2).

relationship(X, Y, siblings) :- sibling(X, Y).
relationship(X, Y, brothers) :- brother(X, Y).
relationship(X, Y, sisters) :- sister(X, Y).
relationship(X, Y, cousins) :- cousin(X, Y).
relationship(X, Y, parent) :- parent(X, Y).
relationship(X, Y, child) :- parent(Y, X).
relationship(X, Y, married) :- married(X, Y).
relationship(X, Y, married) :- married(Y, X).

find_relationship(X, Y, Relationship) :-
    relationship(X, Y, Relationship), !.
find_relationship(_, _, 'no direct relationship').
% Query example
% ?- find_relationship(liz, rebecca, Relationship).
```

Describe:

程式碼的開頭定義了所有的家族成員，包括男性和女性、列出了已婚的配偶關係，
並且使用 parent 規則來描述父母與子女的關係，
之後定義了我們需要判斷的人物關係邏輯。

運行

透過 find_relationship(Cecil, Iris, Relationship).

就能夠輸出元素1以及元素2的關係

Programming Exercise 2

Programming Exercise 2 需要使用三個CSV檔案，

Fees.csv:

```
Payment Type,Amount
A,0.00
B,"21,345.00 "
C,"42,690.00 "
```

Student-Main.csv

```
Student ID,Name,Payment Type
920121001,Andy,A
920121002,Bob,B
920121003,Cecil,C
920121004,Dennis,A
920121005,Edward,B
920121006,Felix,C
920121007,Gigi,B
920121008,Helen,B
920121009,Iris,B
920121010,Jane,A
920121011,Kate,B
920121012,Liz,C
920121013,Martin,A
920121014,Nancy,B
920121015,Oscar,C
920121016,Pattie,B
920121017,Quinn,B
920121018,Rebecca,B
```

Student-Payment.csv

```
Student ID,Amount
920121005,"21,345.00 "
920121009,"21,345.00 "
920121003,"42,690.00 "
920121017,"21,345.00 "
920121012,"21,345.00 "
920121002,"21,345.00 "
```

```
920121014,"15,000.00 "
920121018,"21,345.00 "
920121011,"20,000.00 "
920121006,"42,690.00 "
920121015,"21,345.00 "
920121008,"10,000.00 "
```

Programming Exercise 2 with R

```
# 設定檔案路徑
file_path <- "C:/Users/shane/OneDrive/文件/PL_HW2"

# 讀取 CSV
student_main <- read.csv(file.path(file_path, "HW2-Student-Main.csv"))
fees <- read.csv(file.path(file_path, "HW2-Fees.csv"))
student_payment <- read.csv(file.path(file_path, "HW2-Student-Payment.csv"))

# 檢查每個數據框的結構
cat("Structure of student_main:\n")
str(student_main)
cat("Structure of fees:\n")
str(fees)
cat("Structure of student_payment:\n")
str(student_payment)

# 確保列名正確且唯一
colnames(student_main) <- c("StudentID", "Name", "PaymentType")
colnames(fees) <- c("PaymentType", "AmountRequired")
colnames(student_payment) <- c("StudentID", "AmountPaid")

# 去除數字中的逗號並轉換為數值型
student_payment$AmountPaid <- as.numeric(gsub(",", "", student_payment$AmountPaid))
fees$AmountRequired <- as.numeric(gsub(",", "", fees$AmountRequired))

# 確認轉換後的資料
print(student_payment)
print(fees)
```

```
# 計算 1: 收到的總金額
total_received <- sum(student_payment$AmountPaid, na.rm = TRUE)
cat("Total amount received from students before due:", total_received, "\n")

# 計算 2: 列出未支付足夠費用的學生以及缺少的金額
# 合併 Student-Main 和 Fees 表格來得到每個學生需要支付的金額
merged_data <- merge(student_main, fees, by = "PaymentType")

# 再合併 Student-Payment 表格來得到每個學生已支付的金額
merged_data <- merge(merged_data, student_payment, by = "StudentID", all.x = TRUE)

# 將 NA 值替換為 0，表示這些學生沒有支付任何金額
merged_data$AmountPaid[is.na(merged_data$AmountPaid)] <- 0

# 計算每個學生欠缺的金額
merged_data$AmountShort <- merged_data$AmountRequired - merged_data$AmountPaid

# 列出未支付足夠費用的學生
students_not_paid_enough <- merged_data %>%
  filter(AmountShort > 0) %>%
  select(StudentID, Name, AmountShort)

cat("Students that did not pay the required fees with the amount short:\n")
print(students_not_paid_enough)
```

Programming Exercise 2 with COBOL

```
IDENTIFICATION DIVISION.
PROGRAM-ID. SumAmounts.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT StudentFile ASSIGN TO 'Student-Payment.csv'
        ORGANIZATION IS LINE SEQUENTIAL.

DATA DIVISION.
FILE SECTION.
FD StudentFile.
01 StudentRecord.
    05 StudentID PIC X(10).
    05 Amount PIC X(15).

WORKING-STORAGE SECTION.
01 TotalAmount        PIC 9(10)V99 VALUE 0.
01 EOF-FLAG           PIC X VALUE 'N'.
01 FirstRecord        PIC X VALUE 'Y'.
01 WS-INDEX           PIC 9(2) VALUE 1.
01 WS-OUTPUT-INDEX PIC 9(2) VALUE 1.
01 WS-CHAR            PIC X.
01 CleanAmount        PIC X(15) VALUE SPACES.
01 NumAmount          PIC 9(10)V99.

PROCEDURE DIVISION.
0000-MAIN-PROCEDURE.
    OPEN INPUT StudentFile
    PERFORM UNTIL EOF-FLAG = 'Y'
        READ StudentFile
            AT END
                MOVE 'Y' TO EOF-FLAG
            NOT AT END
                IF FirstRecord = 'Y'
```

```
             THEN
                 MOVE 'N' TO FirstRecord
             ELSE

 MOVE SPACES TO CleanAmount
 MOVE 1 TO WS-OUTPUT-INDEX
 PERFORM VARYING WS-INDEX
 FROM 1 BY 1 UNTIL WS-INDEX > LENGTH OF Amount
     MOVE Amount(WS-INDEX:1) TO WS-CHAR
     IF WS-CHAR NOT = '"' THEN
         MOVE WS-CHAR TO CleanAmount(WS-OUTPUT-INDEX:1)
         ADD 1 TO WS-OUTPUT-INDEX
     END-IF
 END-PERFORM

 MOVE FUNCTION NUMVAL(CleanAmount) TO NumAmount
 ADD NumAmount TO TotalAmount

 DISPLAY "Amount: " CleanAmount
 DISPLAY "NumAmount: " NumAmount
 END-IF
     END-READ
 END-PERFORM
 CLOSE StudentFile
 DISPLAY "Total Amount: " TotalAmount
 STOP RUN.
IDENTIFICATION DIVISION.
PROGRAM-ID. SHORTAMO.

ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT StudentMainFile ASSIGN TO 'Student-Main.csv'
           ORGANIZATION IS LINE SEQUENTIAL.
     SELECT FeesFile ASSIGN TO 'Fees.csv'
           ORGANIZATION IS LINE SEQUENTIAL.
     SELECT StudentPaymentFile ASSIGN TO 'Student-Payment.csv'
           ORGANIZATION IS LINE SEQUENTIAL.
```

```
DATA DIVISION.
FILE SECTION.
FD StudentMainFile.
01 StudentMainRecord.
    05 SM-Student-ID PIC X(10).
    05 SM-Name PIC X(20).
    05 SM-Payment-Type PIC X(1).

FD FeesFile.
01 FeesRecord.
    05 F-Payment-Type PIC X(1).
    05 F-Amount PIC 9(6)V99.

FD StudentPaymentFile.
01 StudentPaymentRecord.
    05 SP-Student-ID PIC X(10).
    05 SP-Amount PIC 9(6)V99.

WORKING-STORAGE SECTION.
01 WS-Student-ID PIC X(10).
01 WS-Name PIC X(20).
01 WS-Payment-Type PIC X(1).
01 WS-Fee-Amount PIC 9(6)V99.
01 WS-Paid-Amount PIC 9(6)V99.
01 WS-Short-Amount PIC S9(6)V99.
01 EOF-SM-FLAG PIC X VALUE 'N'.
01 EOF-FE-FLAG PIC X VALUE 'N'.
01 EOF-SP-FLAG PIC X VALUE 'N'.
01 WS-DISPLAY-AMOUNT PIC 9(6).99.

PROCEDURE DIVISION.
BEGIN.
    OPEN INPUT StudentMainFile FeesFile StudentPaymentFile
    PERFORM READ-FeesFile
    PERFORM UNTIL EOF-SM-FLAG = 'Y'
        PERFORM READ-StudentMainFile
        PERFORM PROCESS-Student
        END-PERFORM
        CLOSE StudentMainFile FeesFile StudentPaymentFile
        STOP RUN.

READ-FeesFile.
        READ FeesFile
                AT END MOVE 'Y' TO EOF-FE-FLAG
        END-READ.

READ-StudentMainFile.
        READ StudentMainFile
                AT END MOVE 'Y' TO EOF-SM-FLAG
        END-READ.

PROCESS-Student.
  MOVE SM-Payment-Type TO WS-Payment-Type
  PERFORM FIND-FEE-Amount
  IF WS-Fee-Amount NOT = ZERO
          MOVE ZERO TO WS-Paid-Amount
          PERFORM READ-StudentPaymentFile
          IF WS-Paid-Amount < WS-Fee-Amount
  COMPUTE WS-Short-Amount = WS-Fee-Amount - WS-Paid-Amount
  MOVE WS-Short-Amount TO WS-DISPLAY-AMOUNT
  DISPLAY SM-Student-ID SPACE SM-Name SPACE WS-DISPLAY-AMOUNT
          END-IF
  END-IF.

FIND-FEE-Amount.
        MOVE ZERO TO WS-Fee-Amount
        PERFORM UNTIL EOF-FE-FLAG = 'Y'
                IF F-Payment-Type = WS-Payment-Type
                        MOVE F-Amount TO WS-Fee-Amount
                        MOVE 'Y' TO EOF-FE-FLAG
                ELSE
                        PERFORM READ-FeesFile
                END-IF
        END-PERFORM.
```

```
READ-StudentPaymentFile.
    MOVE 'N' TO EOF-SP-FLAG
    PERFORM UNTIL EOF-SP-FLAG = 'Y'
        READ StudentPaymentFile
            AT END MOVE 'Y' TO EOF-SP-FLAG
        END-READ
        IF SP-Student-ID = SM-Student-ID
            MOVE SP-Amount TO WS-Paid-Amount
            MOVE 'Y' TO EOF-SP-FLAG
        END-IF
    END-PERFORM.
```

該程式與 R 語言做相同的事情但語法不同

The program listing.

Exercise 1

JAVA

```
import java.util.*;

class Person {
    String name;
    String gender;
    Person spouse;
    Person parent;
    List<Person> children;

    Person(String name, String gender) {
        this.name = name;
        this.gender = gender;
        this.children = new ArrayList<>();
    }

    void setSpouse(Person spouse) {
        this.spouse = spouse;
    }

    void addChild(Person child) {
        this.children.add(child);
        child.parent = this;
    }

    void addChildWithSpouse(Person child) {
        this.addChild(child);
        if (this.spouse != null) {
            this.spouse.addChild(child);
        }
    }

    static boolean areSiblings(Person p1, Person p2) {
        return p1.parent != null && p2.parent != null && p1.parent == p2.parent;
    }

    static boolean areBrothers(Person p1, Person p2) {
        return areSiblings(p1, p2) && p1.gender.equals("male") && p2.gender.equals("male");
    }

    static boolean areSisters(Person p1, Person p2) {
        return areSiblings(p1, p2) && p1.gender.equals("female") &&
p2.gender.equals("female");
    }

    static boolean areCousins(Person p1, Person p2) {
        if (p1.parent == null || p2.parent == null) return false;
        return areSiblings(p1.parent, p2.parent);
    }

    @Override
    public String toString() {
        return name;
    }
}

public class FamilyRelations {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Define persons
        Map<String, Person> people = new HashMap<>();
        String[] males = {"Andy", "Bob", "Cecil", "Dennis", "Edward", "Felix", "Martin", "Oscar",
"Quinn"};
        String[] females = {"Gigi", "Helen", "Iris", "Jane", "Kate", "Liz", "Nancy", "Pattie",
"Rebecca"};

        for (String name : males) {
```

```
    people.put(name, new Person(name, "male"));
}
for (String name : females) {
    people.put(name, new Person(name, "female"));
}

// Define marriages
people.get("Bob").setSpouse(people.get("Helen"));
people.get("Dennis").setSpouse(people.get("Pattie"));
people.get("Gigi").setSpouse(people.get("Martin"));

// Define parent-child relationships
people.get("Andy").addChildWithSpouse(people.get("Bob"));
people.get("Bob").addChildWithSpouse(people.get("Cecil"));
people.get("Cecil").addChildWithSpouse(people.get("Dennis"));
people.get("Dennis").addChildWithSpouse(people.get("Edward"));
people.get("Edward").addChildWithSpouse(people.get("Felix"));
people.get("Gigi").addChildWithSpouse(people.get("Helen"));
people.get("Helen").addChildWithSpouse(people.get("Iris"));
people.get("Iris").addChildWithSpouse(people.get("Jane"));
people.get("Jane").addChildWithSpouse(people.get("Kate"));
people.get("Kate").addChildWithSpouse(people.get("Liz"));
people.get("Martin").addChildWithSpouse(people.get("Nancy"));
people.get("Nancy").addChildWithSpouse(people.get("Oscar"));
people.get("Oscar").addChildWithSpouse(people.get("Pattie"));
people.get("Pattie").addChildWithSpouse(people.get("Quinn"));
people.get("Quinn").addChildWithSpouse(people.get("Rebecca"));

// User input for checking relations
System.out.println("請輸入第一個人的名字：");
String name1 = scanner.nextLine();
System.out.println("請輸入第二個人的名字：");
String name2 = scanner.nextLine();

Person p1 = people.get(name1);
Person p2 = people.get(name2);

if (p1 == null || p2 == null) {
```

```
            System.out.println("輸入的名字不在列表中。");
            scanner.close();
            return;
        }

        if (p1.spouse == p2) {
            System.out.println(p1 + " 和 " + p2 + " 是夫妻。");
        } else if (Person.areSiblings(p1, p2)) {
            System.out.println(p1 + " 和 " + p2 + " 是兄弟姐妹。");
            if (Person.areBrothers(p1, p2)) {
                System.out.println(p1 + " 和 " + p2 + " 是兄弟。");
            } else if (Person.areSisters(p1, p2)) {
                System.out.println(p1 + " 和 " + p2 + " 是姐妹。");
            }
        } else if (Person.areCousins(p1, p2)) {
            System.out.println(p1 + " 和 " + p2 + " 是表親。");
        } else {
            System.out.println(p1 + " 和 " + p2 + " 沒有直接關係。");
        }

        scanner.close();
    }
}
```

ML

```
datatype person = Andy | Bob | Cecil | Dennis | Edward | Felix | Martin | Oscar | Quinn
  | Gigi | Helen | Iris | Jane | Kate | Liz | Nancy | Pattie | Rebecca;

datatype gender = Male | Female;

datatype family =
    Parent of person * person
  | Married of person * person;

val males = [Andy, Bob, Cecil, Dennis, Edward, Felix, Martin, Oscar, Quinn];
val females = [Gigi, Helen, Iris, Jane, Kate, Liz, Nancy, Pattie, Rebecca];

val marriages = [Married(Bob, Helen), Married(Dennis, Pattie), Married(Gigi, Martin)];
val parentChild = [Parent(Andy, Bob), Parent(Bob, Cecil), Parent(Bob, Iris), Parent(Cecil,
Dennis), Parent(Dennis, Edward),
                  Parent(Edward, Felix), Parent(Gigi, Helen), Parent(Helen, Iris), Parent(Iris,
Jane),
                  Parent(Jane, Kate), Parent(Kate, Liz), Parent(Martin, Nancy), Parent(Nancy,
Oscar),
                  Parent(Oscar, Pattie), Parent(Pattie, Quinn), Parent(Quinn, Rebecca)];

fun isMale p = List.exists (fn x => x = p) males;
fun isFemale p = List.exists (fn x => x = p) females;

fun marriedTo p =
    case List.find (fn Married(x, y) => x = p orelse y = p | _ => false) marriages of
         SOME(Married(x, y)) => if x = p then y else x
       | NONE => p;

fun parentOf p = List.filter (fn Parent(x, _) => x = p | _ => false) parentChild;
fun childrenOf p = List.map (fn Parent(_, c) => c) (parentOf p);

fun findParents p = List.map (fn Parent(p, _) => p) (List.filter (fn Parent(_, y) => y = p)
parentChild);

fun siblings p =
    let
        val parentList = findParents p
        val siblingSets = List.concat (List.map childrenOf parentList)
    in
        List.filter (fn x => x <> p) siblingSets
    end;

fun areSiblings (x, y) =
    let
        val xParents = findParents x
        val yParents = findParents y
    in
```

```
        List.exists (fn px => List.exists (fn py => px = py) yParents) xParents
    end;

fun areBrothers (x, y) =
    areSiblings (x, y) andalso isMale x andalso isMale y;

fun areSisters (x, y) =
    areSiblings (x, y) andalso isFemale x andalso isFemale y;

fun grandparents p = List.concat (List.map findParents (findParents p));

fun areCousins (x, y) =
    let
        val xGrandparents = grandparents x
        val yGrandparents = grandparents y
    in
        List.exists (fn xp => List.exists (fn yp => xp = yp) yGrandparents) xGrandparents
        andalso not (areSiblings (x, y))
    end;

fun isParent (x, y) =
    List.exists (fn Parent(p, c) => p = x andalso c = y | _ => false) parentChild;

fun relationship (x, y) =
    if isParent (x, y) then "parent"
    else if isParent (y, x) then "child"
    else if List.exists (fn Married(a, b) => (a = x andalso b = y) orelse (a = y andalso b = x))
marriages then "spouse"
    else if areBrothers (x, y) then "brothers"
    else if areSisters (x, y) then "sisters"
    else if areSiblings (x, y) then "siblings"
    else if areCousins (x, y) then "cousins"
    else "no relation";

(* test *)
val _ = print("Relationship between Cecil and Iris: " ^ relationship (Cecil, Iris) ^ "\n");
val _ = print("Relationship between Liz and Kate: " ^ relationship (Liz, Kate) ^ "\n");
val _ = print("Relationship between Dennis and Jane: " ^ relationship (Dennis, Jane) ^ "\n");
val _ = print("Relationship between Helen and Iris: " ^ relationship (Helen, Iris) ^ "\n");
val _ = print("Relationship between Bob and Helen: " ^ relationship (Bob, Helen) ^ "\n");
val _ = print("Relationship between Gigi and Martin: " ^ relationship (Gigi, Martin) ^ "\n");
val _ = print("Relationship between Dennis and Pattie: " ^ relationship (Dennis, Pattie) ^ "\n");
```

prolog

```
male(andy).
male(bob).
male(cecil).
male(dennis).
male(edward).
male(felix).
male(martin).
male(oscar).
male(quinn).

female(gigi).
female(helen).
female(iris).
female(jane).
female(kate).
female(liz).
female(nancy).
female(pattie).
female(rebecca).

married(bob, helen).
married(dennis, pattie).
married(gigi, martin).

parent(andy, bob).
parent(bob, cecil).
parent(cecil, dennis).
parent(dennis, edward).
parent(edward, felix).
parent(gigi, helen).
parent(helen, iris).
parent(iris, jane).
parent(jane, kate).
parent(kate, liz).
parent(martin, nancy).
parent(nancy, oscar).
parent(oscar, pattie).
parent(pattie, quinn).
parent(quinn, rebecca).

parent(X, Z) :- married(X, Y), parent(Y, Z).

sibling(X, Y) :- parent(P, X), parent(P, Y), X \= Y.

brother(X, Y) :- sibling(X, Y), male(X).

sister(X, Y) :- sibling(X, Y), female(X).

cousin(X, Y) :- parent(P1, X), parent(P2, Y), sibling(P1, P2).

relationship(X, Y, siblings) :- sibling(X, Y).
relationship(X, Y, brothers) :- brother(X, Y).
relationship(X, Y, sisters) :- sister(X, Y).
relationship(X, Y, cousins) :- cousin(X, Y).
relationship(X, Y, parent) :- parent(X, Y).
relationship(X, Y, child) :- parent(Y, X).
relationship(X, Y, married) :- married(X, Y).
relationship(X, Y, married) :- married(Y, X).

find_relationship(X, Y, Relationship) :-
    relationship(X, Y, Relationship), !.
find_relationship(_, _, 'no direct relationship').

% Query example
% To find the relationship between Liz and Rebecca
% ?- find_relationship(liz, rebecca, Relationship).
% ?- find_relationship(Cecil, Iris, Relationship).
```

Programming Exercise 2

Fees.csv:

```
Payment Type,Amount
A,0.00
B,"21,345.00 "
C,"42,690.00 "
```

Student-Main.csv

```
Student ID,Name,Payment Type
920121001,Andy,A
920121002,Bob,B
920121003,Cecil,C
920121004,Dennis,A
920121005,Edward,B
920121006,Felix,C
920121007,Gigi,B
920121008,Helen,B
920121009,Iris,B
920121010,Jane,A
920121011,Kate,B
920121012,Liz,C
920121013,Martin,A
920121014,Nancy,B
920121015,Oscar,C
920121016,Pattie,B
920121017,Quinn,B
920121018,Rebecca,B
```

Student-Payment.csv

```
Student ID,Amount
920121005,"21,345.00 "
920121009,"21,345.00 "
```

```
920121003,"42,690.00 "
920121017,"21,345.00 "
920121012,"21,345.00 "
920121002,"21,345.00 "
920121014,"15,000.00 "
920121018,"21,345.00 "
920121011,"20,000.00 "
920121006,"42,690.00 "
920121015,"21,345.00 "
920121008,"10,000.00 "
```

R

```r
# 設定檔案路徑
file_path <- "C:/Users/shane/OneDrive/文件/PL_HW2"

# 讀取 CSV
student_main <- read.csv(file.path(file_path, "HW2-Student-Main.csv"))
fees <- read.csv(file.path(file_path, "HW2-Fees.csv"))
student_payment <- read.csv(file.path(file_path, "HW2-Student-Payment.csv"))

# 檢查每個數據框的結構
cat("Structure of student_main:\n")
str(student_main)
cat("Structure of fees:\n")
str(fees)
cat("Structure of student_payment:\n")
str(student_payment)

# 確保列名正確且唯一
colnames(student_main) <- c("StudentID", "Name", "PaymentType")
colnames(fees) <- c("PaymentType", "AmountRequired")
colnames(student_payment) <- c("StudentID", "AmountPaid")

# 去除數字中的逗號並轉換為數值型
student_payment$AmountPaid <- as.numeric(gsub(",", "", student_payment$AmountPaid))
fees$AmountRequired <- as.numeric(gsub(",", "", fees$AmountRequired))
```

```
# 計算 1: 收到的總金額
total_received <- sum(student_payment$AmountPaid, na.rm = TRUE)
cat("Total amount received from students before due:", total_received, "\n")

# 計算 2: 列出未支付足夠費用的學生以及缺少的金額
# 合併 Student-Main 和 Fees 表格來得到每個學生需要支付的金額
merged_data <- merge(student_main, fees, by = "PaymentType")

# 再合併 Student-Payment 表格來得到每個學生已支付的金額
merged_data <- merge(merged_data, student_payment, by = "StudentID", all.x = TRUE)

# 將 NA 值替換為 0，表示這些學生沒有支付任何金額
merged_data$AmountPaid[is.na(merged_data$AmountPaid)] <- 0

# 計算每個學生欠缺的金額
merged_data$AmountShort <- merged_data$AmountRequired - merged_data$AmountPaid

# 列出未支付足夠費用的學生
students_not_paid_enough <- merged_data %>%
  filter(AmountShort > 0) %>%
  select(StudentID, Name, AmountShort)

cat("Students that did not pay the required fees with the amount short:\n")
print(students_not_paid_enough)
```

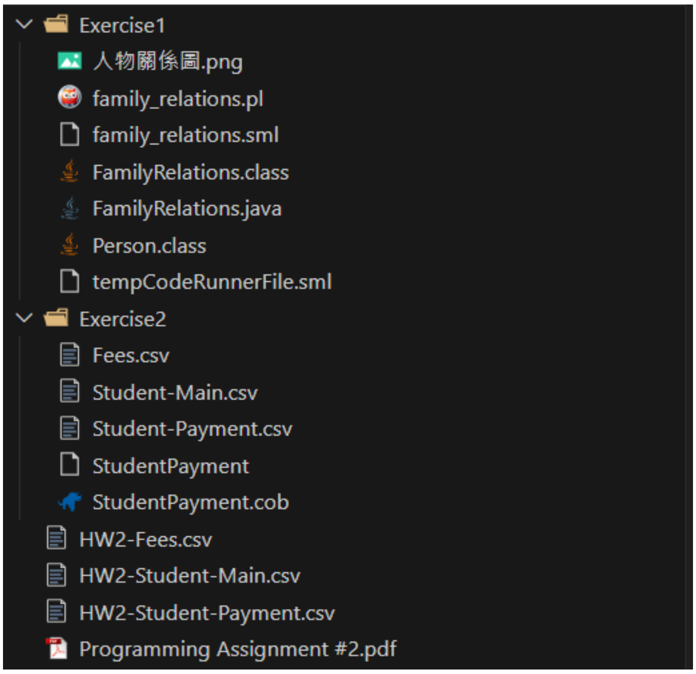
COBOL

```
IDENTIFICATION DIVISION.
        PROGRAM-ID. SumAmounts.

        ENVIRONMENT DIVISION.
        INPUT-OUTPUT SECTION.
        FILE-CONTROL.
            SELECT StudentFile ASSIGN TO 'Student-Payment.csv'
                ORGANIZATION IS LINE SEQUENTIAL.

        DATA DIVISION.
```

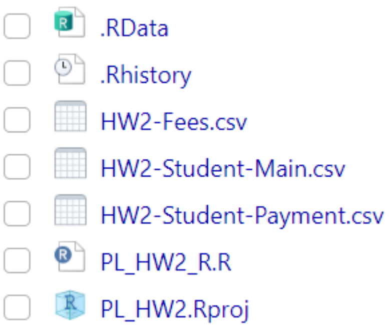
```
FILE SECTION.
FD StudentFile.
01 StudentRecord.
    05 StudentID PIC X(10).
    05 Amount PIC X(15).

WORKING-STORAGE SECTION.
01 TotalAmount      PIC 9(10)V99 VALUE 0.
01 EOF-FLAG         PIC X VALUE 'N'.
01 FirstRecord     PIC X VALUE 'Y'.
01 WS-INDEX         PIC 9(2) VALUE 1.
01 WS-OUTPUT-INDEX PIC 9(2) VALUE 1.
01 WS-CHAR          PIC X.
01 CleanAmount      PIC X(15) VALUE SPACES.
01 NumAmount         PIC 9(10)V99.

PROCEDURE DIVISION.
0000-MAIN-PROCEDURE.
    OPEN INPUT StudentFile
    PERFORM UNTIL EOF-FLAG = 'Y'
        READ StudentFile
            AT END
                MOVE 'Y' TO EOF-FLAG
            NOT AT END
                IF FirstRecord = 'Y'
                THEN
                    MOVE 'N' TO FirstRecord
                ELSE

    MOVE SPACES TO CleanAmount
    MOVE 1 TO WS-OUTPUT-INDEX
    PERFORM VARYING WS-INDEX
    FROM 1 BY 1 UNTIL WS-INDEX > LENGTH OF Amount
        MOVE Amount(WS-INDEX:1) TO WS-CHAR
        IF WS-CHAR NOT = '"' THEN
            MOVE WS-CHAR TO CleanAmount(WS-OUTPUT-INDEX:1)
            ADD 1 TO WS-OUTPUT-INDEX
        END-IF
```

```
END-PERFORM

MOVE FUNCTION NUMVAL(CleanAmount) TO NumAmount
ADD NumAmount TO TotalAmount

DISPLAY "Amount: " CleanAmount
DISPLAY "NumAmount: " NumAmount
END-IF
    END-READ
END-PERFORM
CLOSE StudentFile
DISPLAY "Total Amount: " TotalAmount
STOP RUN.
```

檔案總表:



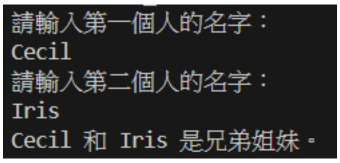
```
Exercise1
    人物關係圖.png
    family_relations.pl
    family_relations.sml
    FamilyRelations.class
    FamilyRelations.java
    Person.class
    tempCodeRunnerFile.sml
Exercise2
    Fees.csv
    Student-Main.csv
    Student-Payment.csv
    StudentPayment
    StudentPayment.cob
HW2-Fees.csv
HW2-Student-Main.csv
HW2-Student-Payment.csv
Programming Assignment #2.pdf
```



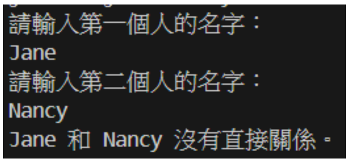
- □ .RData
- □ .Rhistory
- □ HW2-Fees.csv
- □ HW2-Student-Main.csv
- □ HW2-Student-Payment.csv
- □ PL_HW2_R.R
- □ PL_HW2.Rproj

Test run results.

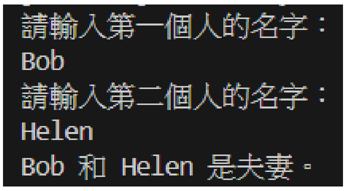
Java



```
請輸入第一個人的名字：
Cecil
請輸入第二個人的名字：
Iris
Cecil 和 Iris 是兄弟姐妹。
```

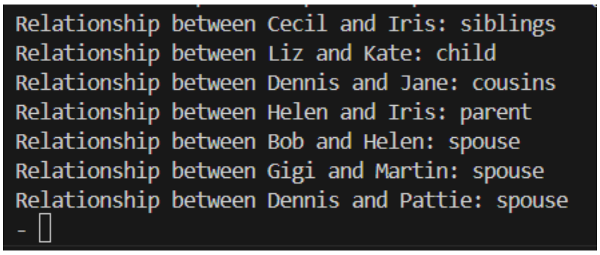


```
請輸入第一個人的名字：
Jane
請輸入第二個人的名字：
Nancy
Jane 和 Nancy 沒有直接關係。
```



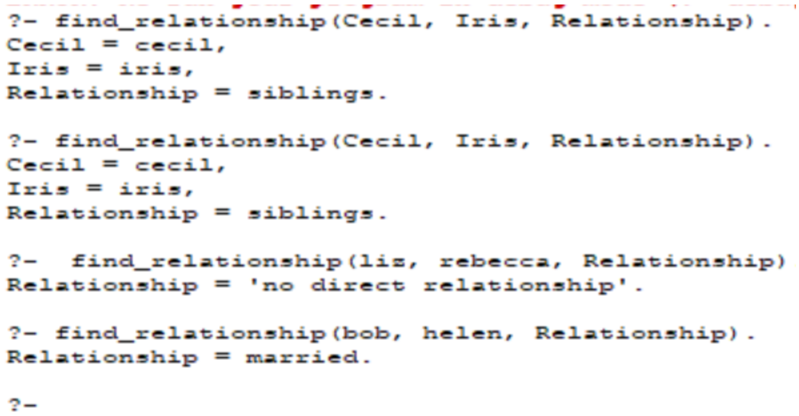
```
請輸入第一個人的名字：
Bob
請輸入第二個人的名字：
Helen
Bob 和 Helen 是夫妻。
```

ML



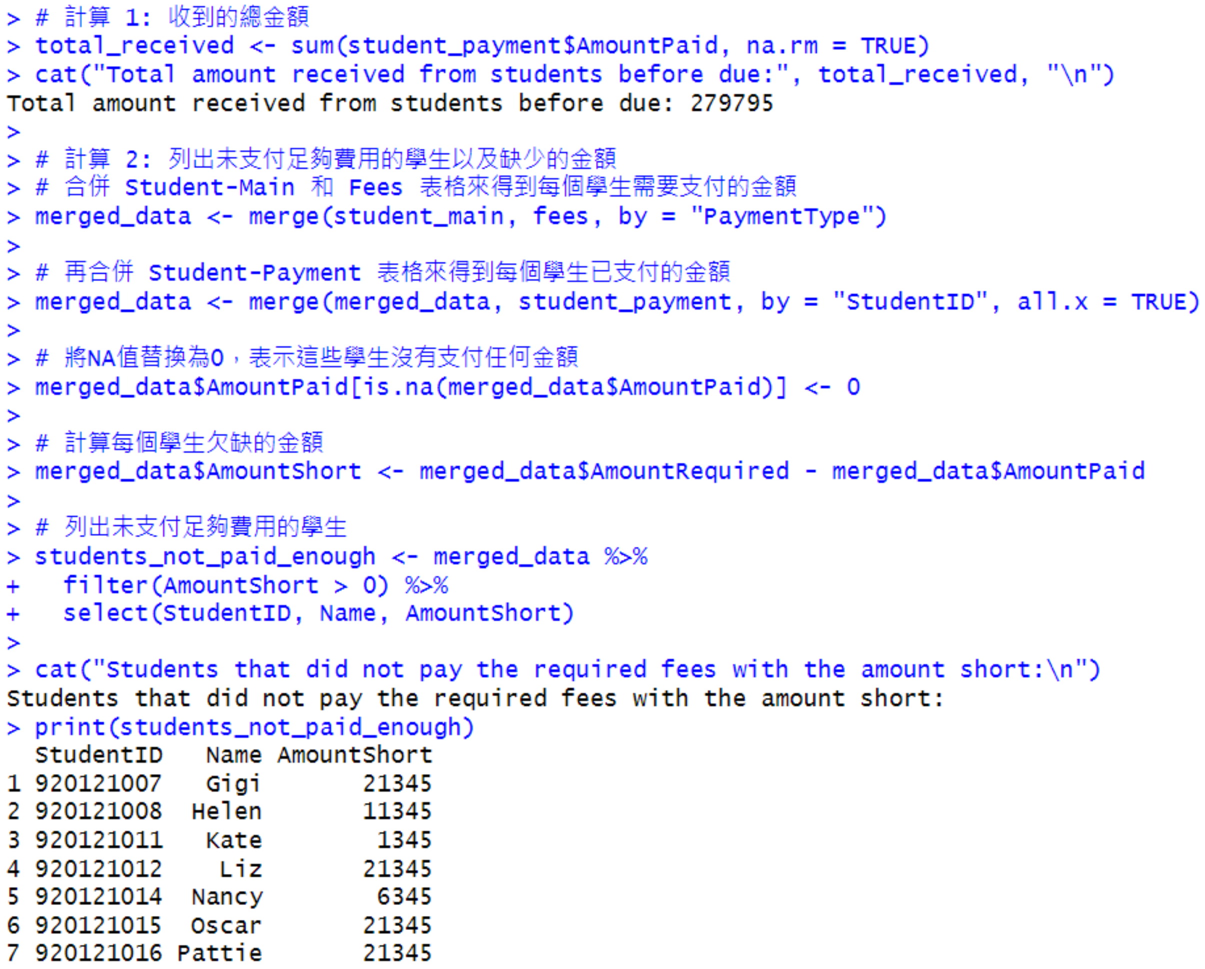
```
Relationship between Cecil and Iris: siblings
Relationship between Liz and Kate: child
Relationship between Dennis and Jane: cousins
Relationship between Helen and Iris: parent
Relationship between Bob and Helen: spouse
Relationship between Gigi and Martin: spouse
Relationship between Dennis and Pattie: spouse
- 
```

Prolog



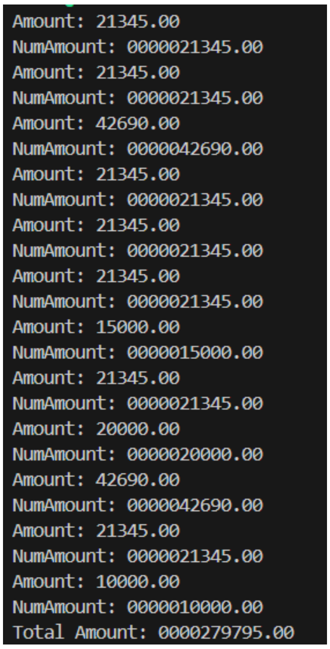
```
?- find_relationship(Cecil, Iris, Relationship).
Cecil = cecil,
Iris = iris,
Relationship = siblings.

?- find_relationship(Cecil, Iris, Relationship).
Cecil = cecil,
Iris = iris,
Relationship = siblings.

?-  find_relationship(liz, rebecca, Relationship)
Relationship = 'no direct relationship'.

?- find_relationship(bob, helen, Relationship).
Relationship = married.

?-
```

R

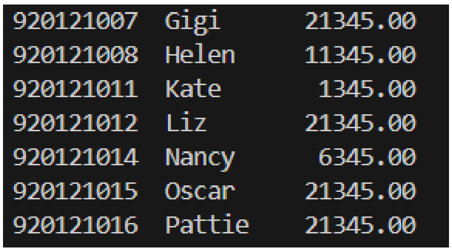


```
> # 計算 1: 收到的總金額
> total_received <- sum(student_payment$AmountPaid, na.rm = TRUE)
> cat("Total amount received from students before due:", total_received, "\n")
Total amount received from students before due: 279795
>
> # 計算 2: 列出未支付足夠費用的學生以及缺少的金額
> # 合併 Student-Main 和 Fees 表格來得到每個學生需要支付的金額
> merged_data <- merge(student_main, fees, by = "PaymentType")
>
> # 再合併 Student-Payment 表格來得到每個學生已支付的金額
> merged_data <- merge(merged_data, student_payment, by = "StudentID", all.x = TRUE)
>
> # 將NA值替換為0，表示這些學生沒有支付任何金額
> merged_data$AmountPaid[is.na(merged_data$AmountPaid)] <- 0
>
> # 計算每個學生欠缺的金額
> merged_data$AmountShort <- merged_data$AmountRequired - merged_data$AmountPaid
>
> # 列出未支付足夠費用的學生
> students_not_paid_enough <- merged_data %>%
+   filter(AmountShort > 0) %>%
+   select(StudentID, Name, AmountShort)
>
> cat("Students that did not pay the required fees with the amount short:\n")
Students that did not pay the required fees with the amount short:
> print(students_not_paid_enough)
  StudentID   Name AmountShort
1 920121007   Gigi       21345
2 920121008  Helen       11345
3 920121011   Kate        1345
4 920121012    Liz       21345
5 920121014  Nancy        6345
6 920121015  Oscar       21345
7 920121016 Pattie       21345
```

COBOL



```
Amount: 21345.00
NumAmount: 0000021345.00
Amount: 21345.00
NumAmount: 0000021345.00
Amount: 42690.00
NumAmount: 0000042690.00
Amount: 21345.00
NumAmount: 0000021345.00
Amount: 21345.00
NumAmount: 0000021345.00
Amount: 21345.00
NumAmount: 0000021345.00
Amount: 15000.00
NumAmount: 0000015000.00
Amount: 21345.00
NumAmount: 0000021345.00
Amount: 20000.00
NumAmount: 0000020000.00
Amount: 42690.00
NumAmount: 0000042690.00
Amount: 21345.00
NumAmount: 0000021345.00
Amount: 10000.00
NumAmount: 0000010000.00
Total Amount: 0000279795.00
```



```
920121007  Gigi      21345.00
920121008  Helen     11345.00
920121011  Kate       1345.00
920121012  Liz       21345.00
920121014  Nancy      6345.00
920121015  Oscar     21345.00
920121016  Pattie    21345.00
```

Discussion.

寫這份作業對我來說充滿了挑戰，因為之前所有的程式都是使用 C/C++完成的，幾乎沒有接觸過其他程式語言，因此這項作業要求我們利用五種不同的程式語言撰寫救我而言是很大的挑戰，在撰寫的過程中我也感受到了不同語言之間的特性差異，相同的任務利用不同的程式語言做撰寫，行數卻差很多，有的程式語言可以利用很簡潔的語法以及邏輯完成任務，有些語言卻需要做很多的操作才能達成。

令我印象最深刻的就是 prolog 語言，他是一種邏輯程式設計語言，因此我們只需要將 list 以及相應的邏輯簡單的建立好，就能輕鬆的獲得想要的結果，反觀在使用 java 跟 ML 時，需要複雜的操作步驟。

針對第二題我也能感受到 R 語言與 COBOL 語言之間的差異，就這次的經驗，我更喜歡 R，因為他有許多好用的函數，我們只需要呼叫，他就能為我們快速進行資料的操作，反觀 COBOL 需要程式設計師做很多的預處理才能，完成 R 一個指令就能完成的任務，此外 R 語言針對表單的針對表單的構建也更加直覺快速。此外針對 COBOL 語言，我覺得很特別的是它限制了工程師每一行的字元上限，這是我從未想過的，覺得是很特殊的規定，讓習慣取長變數名稱的我，在撰寫 COBOL 時相當的痛苦，因為一不小心就會超出界線。

這個程式作業幫助我對程式設計有更多的認識，也接觸到了從未接觸過的語言，也訓練了自己快速學習一個新程式的能力，雖然完成這份作業後，我沒辦法成為這些語言的專家，但這讓我在未來接觸新的平台，新的語言以及新的開發環境時，有更多的勇氣與經驗。