Compiler programming hw2

411121307 葉宸君

Problem 1 description

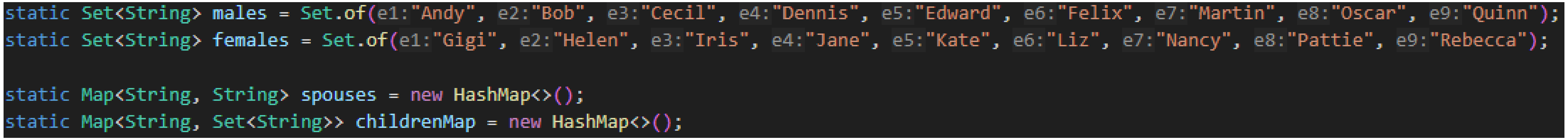
- Giving the following facts, find the relationship between any two person.
- Using language: Java, ML, Prolog.

Java

- Initialize:
  - Initialize two sets to separate male and female.
  - Using map to save the spouses. Should be initialize in pair.
  - Initialize the children that every parents have.
- Use ‘scanner’ to read the user input.
- Relation check:
  - If a and b have the same parents, they are siblings.
    - If a and b are siblings, and a and b are both male, they are brothers.
    - If a and b are siblings, and a and b are both female, they are sisters.
  - If a and b’s parents are siblings, they are cousins.

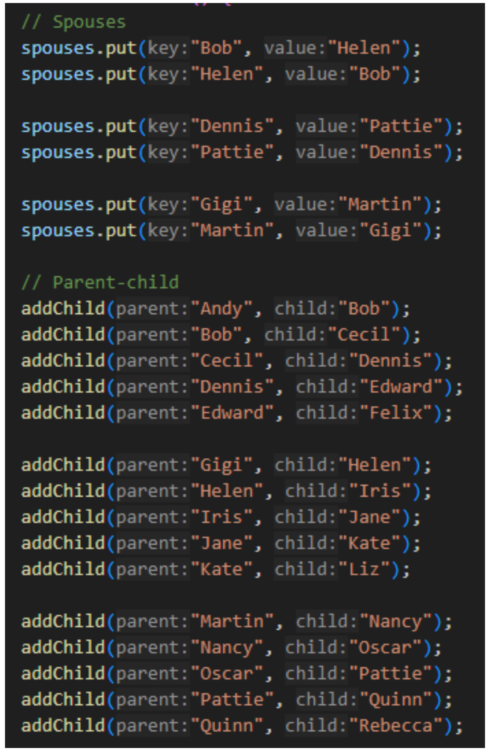
Java program listing

- Initialize



```
static Set<String> males = Set.of(e1:"Andy", e2:"Bob", e3:"Cecil", e4:"Dennis", e5:"Edward", e6:"Felix", e7:"Martin", e8:"Oscar", e9:"Quinn");
static Set<String> females = Set.of(e1:"Gigi", e2:"Helen", e3:"Iris", e4:"Jane", e5:"Kate", e6:"Liz", e7:"Nancy", e8:"Pattie", e9:"Rebecca");

static Map<String, String> spouses = new HashMap<>();
static Map<String, Set<String>> childrenMap = new HashMap<>();
```

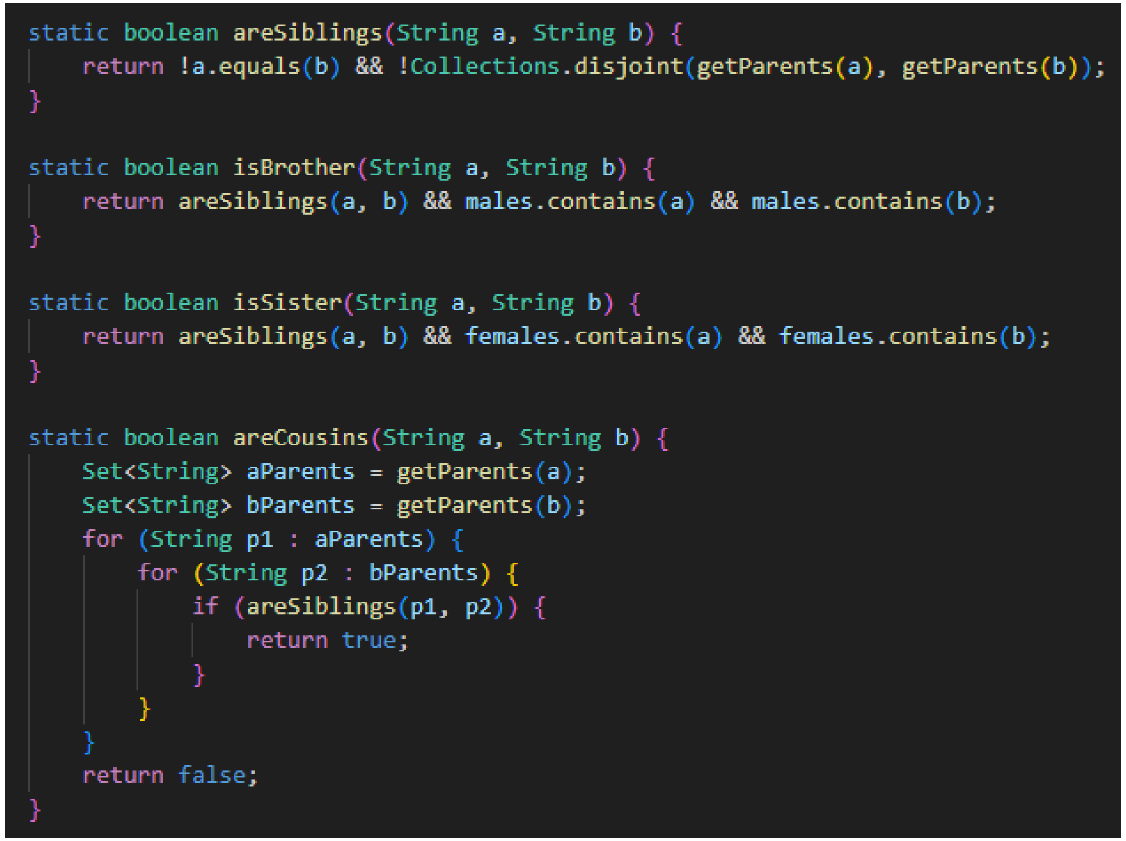
- Building relations in map



```
// Spouses
spouses.put(key:"Bob", value:"Helen");
spouses.put(key:"Helen", value:"Bob");

spouses.put(key:"Dennis", value:"Pattie");
spouses.put(key:"Pattie", value:"Dennis");

spouses.put(key:"Gigi", value:"Martin");
spouses.put(key:"Martin", value:"Gigi");

// Parent-child
addChild(parent:"Andy", child:"Bob");
addChild(parent:"Bob", child:"Cecil");
addChild(parent:"Cecil", child:"Dennis");
addChild(parent:"Dennis", child:"Edward");
addChild(parent:"Edward", child:"Felix");

addChild(parent:"Gigi", child:"Helen");
addChild(parent:"Helen", child:"Iris");
addChild(parent:"Iris", child:"Jane");
addChild(parent:"Jane", child:"Kate");
addChild(parent:"Kate", child:"Liz");

addChild(parent:"Martin", child:"Nancy");
addChild(parent:"Nancy", child:"Oscar");
addChild(parent:"Oscar", child:"Pattie");
addChild(parent:"Pattie", child:"Quinn");
addChild(parent:"Quinn", child:"Rebecca");
```

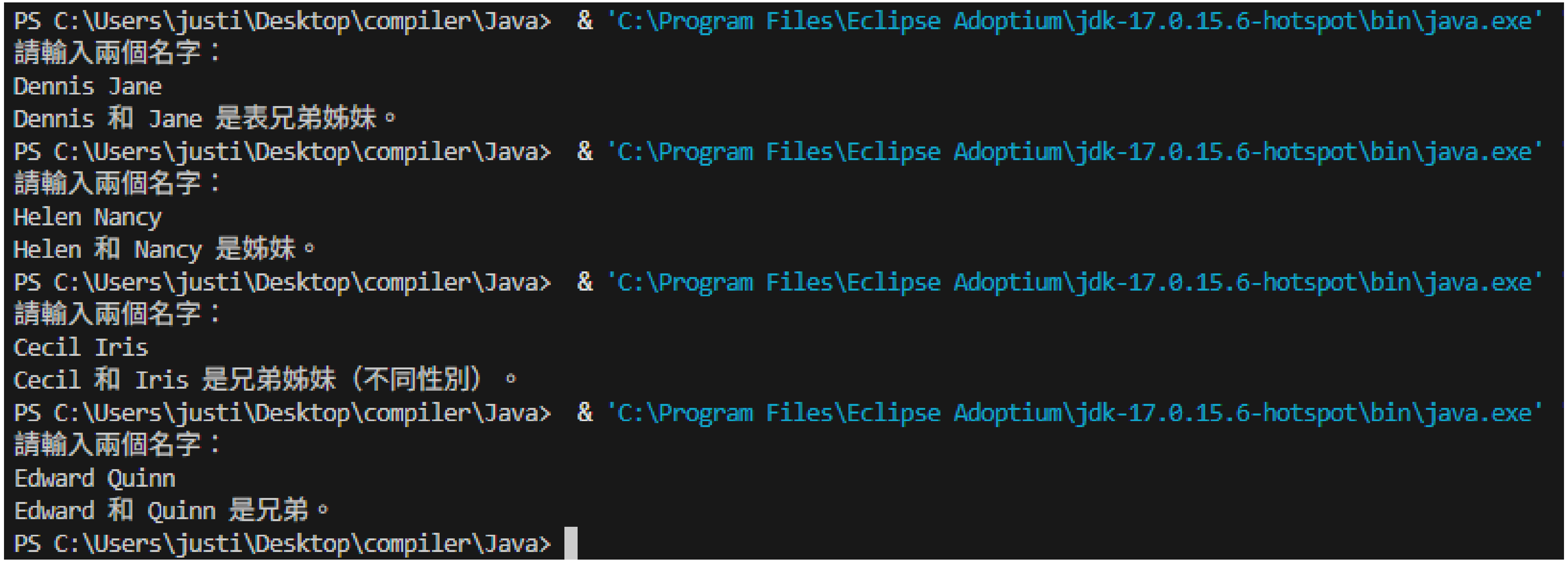
Java program listing cont.

- Check relation whether exist



```java
static boolean areSiblings(String a, String b) {
    return !a.equals(b) && !Collections.disjoint(getParents(a), getParents(b));
}

static boolean isBrother(String a, String b) {
    return areSiblings(a, b) && males.contains(a) && males.contains(b);
}

static boolean isSister(String a, String b) {
    return areSiblings(a, b) && females.contains(a) && females.contains(b);
}

static boolean areCousins(String a, String b) {
    Set<String> aParents = getParents(a);
    Set<String> bParents = getParents(b);
    for (String p1 : aParents) {
        for (String p2 : bParents) {
            if (areSiblings(p1, p2)) {
                return true;
            }
        }
    }
    return false;
}
```

Java test run results



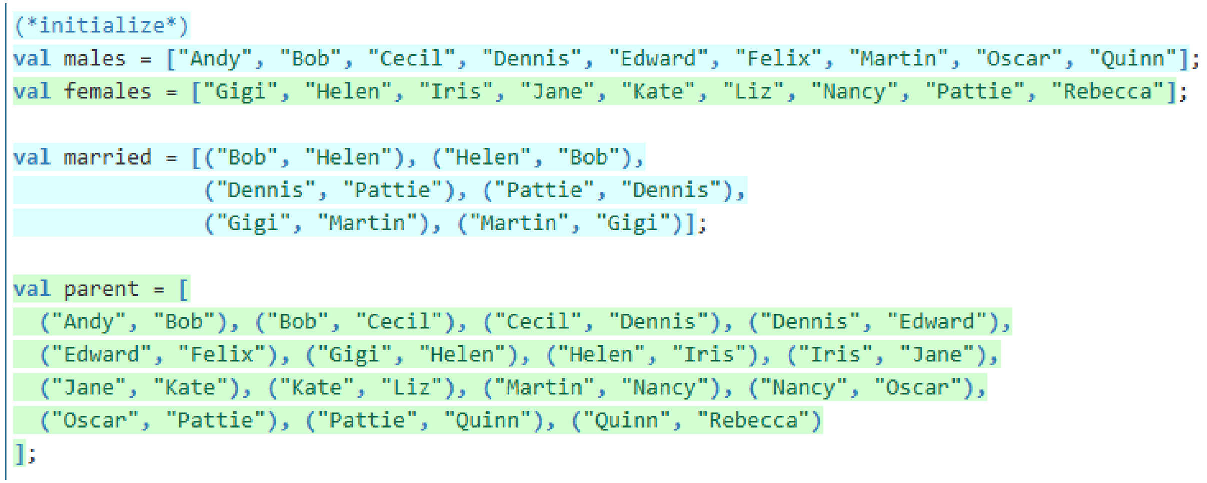
```
PS C:\Users\justi\Desktop\compiler\Java>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.15.6-hotspot\bin\java.exe'
請輸入兩個名字：
Dennis Jane
Dennis 和 Jane 是表兄弟姊妹。
PS C:\Users\justi\Desktop\compiler\Java>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.15.6-hotspot\bin\java.exe'
請輸入兩個名字：
Helen Nancy
Helen 和 Nancy 是姊妹。
PS C:\Users\justi\Desktop\compiler\Java>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.15.6-hotspot\bin\java.exe'
請輸入兩個名字：
Cecil Iris
Cecil 和 Iris 是兄弟姊妹（不同性別）。
PS C:\Users\justi\Desktop\compiler\Java>  & 'C:\Program Files\Eclipse Adoptium\jdk-17.0.15.6-hotspot\bin\java.exe'
請輸入兩個名字：
Edward Quinn
Edward 和 Quinn 是兄弟。
PS C:\Users\justi\Desktop\compiler\Java>
```

ML

- Initialize relationship.
- Define a function that returns the children's parents.
- Siblings check:
    - Find the children's parents first.
    - List all the child the parents have.
    - Check every children whether have the same name as the input.
    - If yes, return true for siblings.
- Cousin check:
    - Find input child's parents first.
    - Find the parents' siblings.
    - Find the siblings' children.
    - These are the input child's cousins.

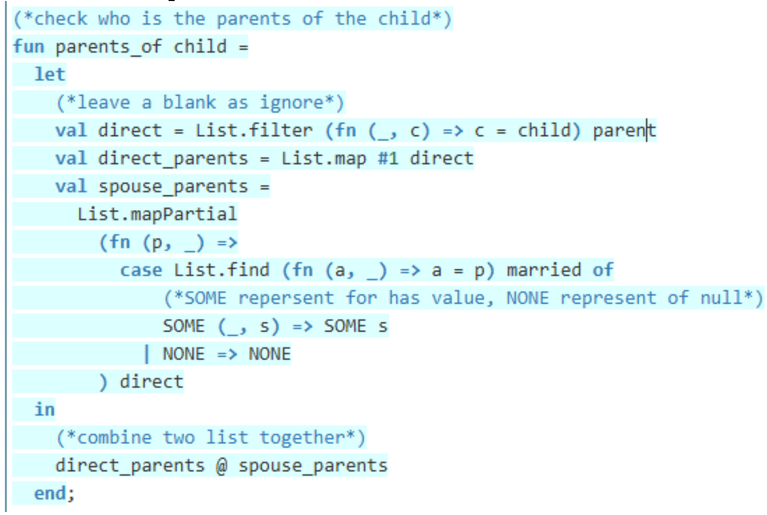
ML program listing

- Initialize relationship



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

val married = [("Bob", "Helen"), ("Helen", "Bob"),
               ("Dennis", "Pattie"), ("Pattie", "Dennis"),
               ("Gigi", "Martin"), ("Martin", "Gigi")];

val parent = [
  ("Andy", "Bob"), ("Bob", "Cecil"), ("Cecil", "Dennis"), ("Dennis", "Edward"),
  ("Edward", "Felix"), ("Gigi", "Helen"), ("Helen", "Iris"), ("Iris", "Jane"),
  ("Jane", "Kate"), ("Kate", "Liz"), ("Martin", "Nancy"), ("Nancy", "Oscar"),
  ("Oscar", "Pattie"), ("Pattie", "Quinn"), ("Quinn", "Rebecca")
];
```

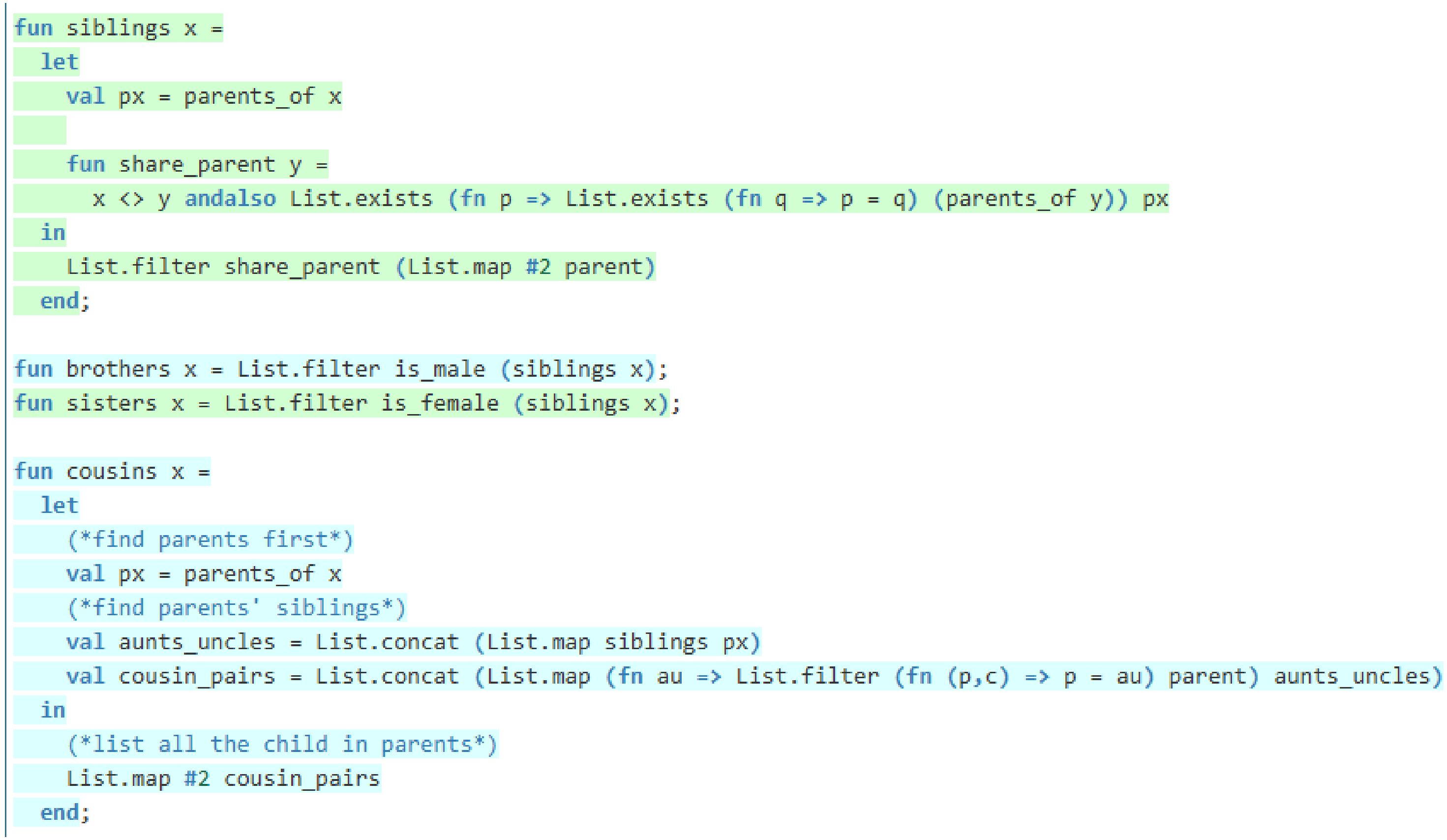
- Find parents



```
(*check who is the parents of the child*)
fun parents_of child =
  let
    (*leave a blank as ignore*)
    val direct = List.filter (fn (_, c) => c = child) parent
    val direct_parents = List.map #1 direct
    val spouse_parents =
      List.mapPartial
        (fn (p, _) =>
          case List.find (fn (a, _) => a = p) married of
              (*SOME repersent for has value, NONE represent of null*)
              SOME (_, s) => SOME s
            | NONE => NONE
        ) direct
  in
    (*combine two list together*)
    direct_parents @ spouse_parents
  end;
```

ML program listing cont.

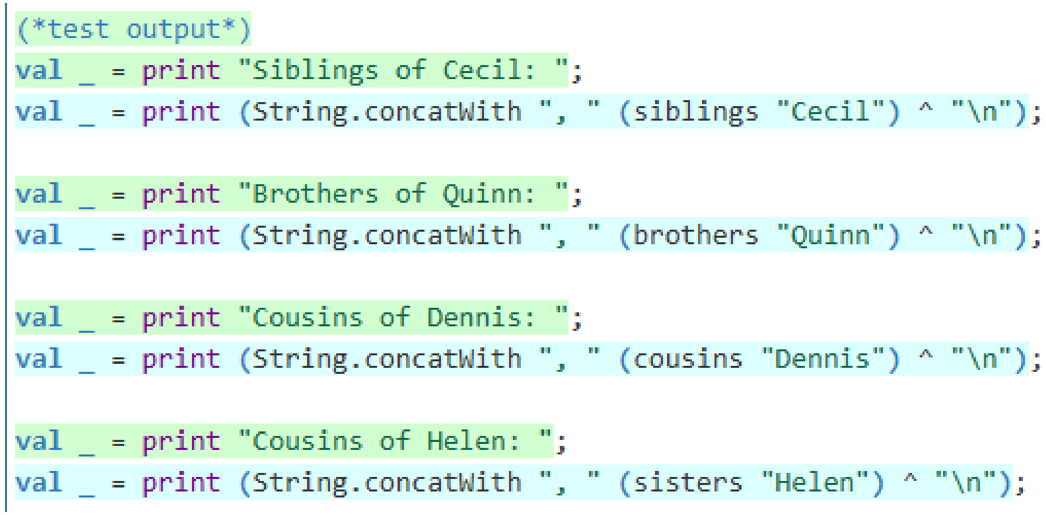
- Check relationship



```
fun siblings x =
  let
    val px = parents_of x

    fun share_parent y =
      x <> y andalso List.exists (fn p => List.exists (fn q => p = q) (parents_of y)) px
  in
    List.filter share_parent (List.map #2 parent)
  end;

fun brothers x = List.filter is_male (siblings x);
fun sisters x = List.filter is_female (siblings x);

fun cousins x =
  let
    (*find parents first*)
    val px = parents_of x
    (*find parents' siblings*)
    val aunts_uncles = List.concat (List.map siblings px)
    val cousin_pairs = List.concat (List.map (fn au => List.filter (fn (p,c) => p = au) parent) aunts_uncles)
  in
    (*list all the child in parents*)
    List.map #2 cousin_pairs
  end;
```

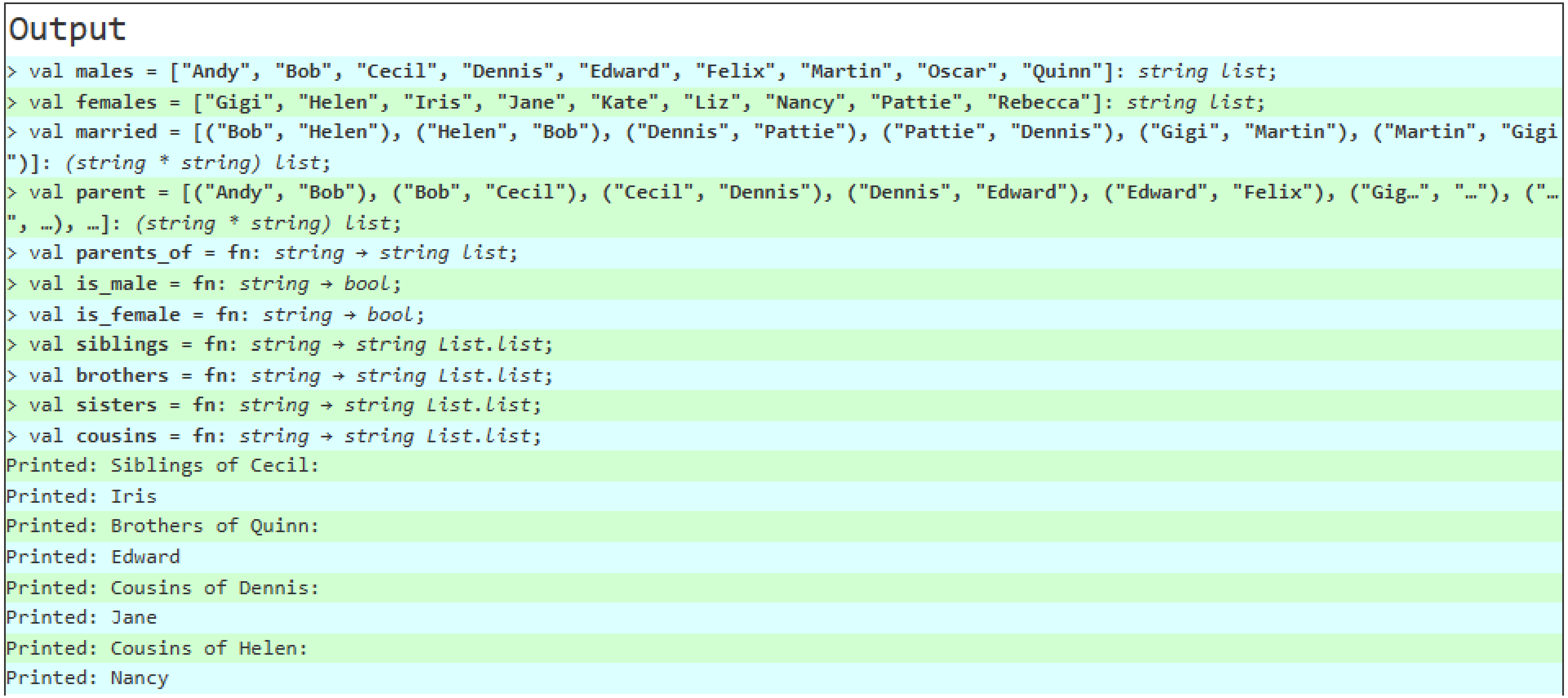
ML program listing cont.

- Output test cases



```
(*test output*)
val _ = print "Siblings of Cecil: ";
val _ = print (String.concatWith ", " (siblings "Cecil") ^ "\n");

val _ = print "Brothers of Quinn: ";
val _ = print (String.concatWith ", " (brothers "Quinn") ^ "\n");

val _ = print "Cousins of Dennis: ";
val _ = print (String.concatWith ", " (cousins "Dennis") ^ "\n");

val _ = print "Cousins of Helen: ";
val _ = print (String.concatWith ", " (sisters "Helen") ^ "\n");
```

ML test run results



Output

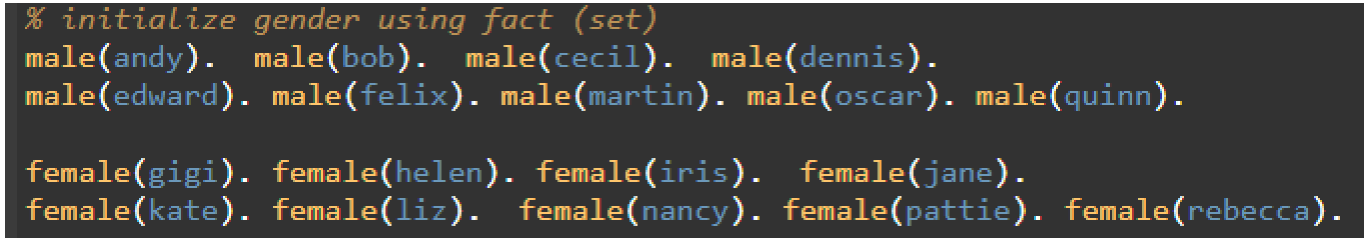
```
> val males = ["Andy", "Bob", "Cecil", "Dennis", "Edward", "Felix", "Martin", "Oscar", "Quinn"]: string list;
> val females = ["Gigi", "Helen", "Iris", "Jane", "Kate", "Liz", "Nancy", "Pattie", "Rebecca"]: string list;
> val married = [("Bob", "Helen"), ("Helen", "Bob"), ("Dennis", "Pattie"), ("Pattie", "Dennis"), ("Gigi", "Martin"), ("Martin", "Gigi
")]: (string * string) list;
> val parent = [("Andy", "Bob"), ("Bob", "Cecil"), ("Cecil", "Dennis"), ("Dennis", "Edward"), ("Edward", "Felix"), ("Gig…", "…"), ("…
", …), …]: (string * string) list;
> val parents_of = fn: string → string list;
> val is_male = fn: string → bool;
> val is_female = fn: string → bool;
> val siblings = fn: string → string List.list;
> val brothers = fn: string → string List.list;
> val sisters = fn: string → string List.list;
> val cousins = fn: string → string List.list;
Printed: Siblings of Cecil:
Printed: Iris
Printed: Brothers of Quinn:
Printed: Edward
Printed: Cousins of Dennis:
Printed: Jane
Printed: Cousins of Helen:
Printed: Nancy
```

Prolog

- Initialize relationship:
  - Gender using fact (Set).
  - Marriage and direct_parent using relation (HashMap).
  - Parent using rule.
- Relation check:
  - if x and y have same parents, and x != y, they are siblings.
  - if x and y are siblings, and both x and y are male, they are brothers.
  - if x and y are siblings, and both x and y are female, they are sisters.
  - if y's parent w and z' parent x are siblings, then z and y are cousins.

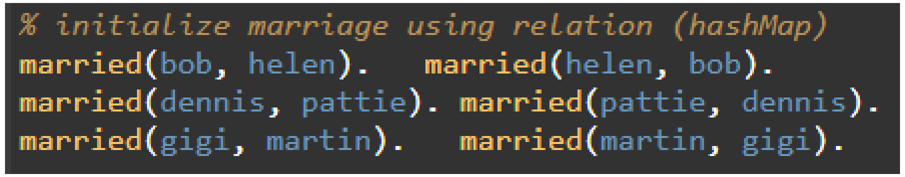
Prolog program listing

- Initialize gender



```
% initialize gender using fact (set)
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).
```

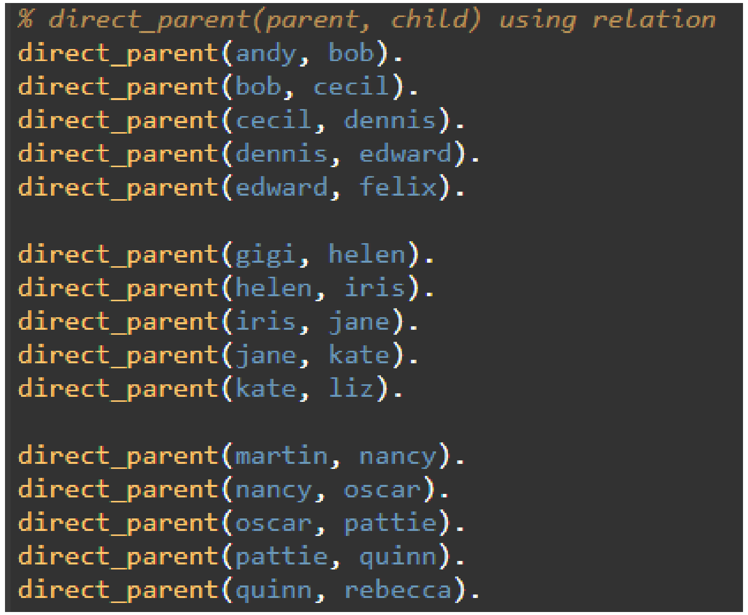
- Initialize marriage



```
% initialize marriage using relation (hashMap)
married(bob, helen).   married(helen, bob).
married(dennis, pattie). married(pattie, dennis).
married(gigi, martin).   married(martin, gigi).
```

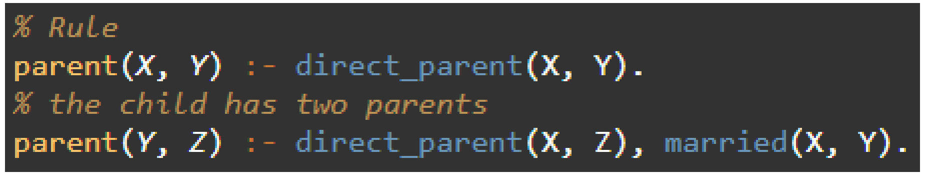
Prolog program listing cont.

- Initialize direct parents



```
% direct_parent(parent, child) using relation
direct_parent(andy, bob).
direct_parent(bob, cecil).
direct_parent(cecil, dennis).
direct_parent(dennis, edward).
direct_parent(edward, felix).

direct_parent(gigi, helen).
direct_parent(helen, iris).
direct_parent(iris, jane).
direct_parent(jane, kate).
direct_parent(kate, liz).

direct_parent(martin, nancy).
direct_parent(nancy, oscar).
direct_parent(oscar, pattie).
direct_parent(pattie, quinn).
direct_parent(quinn, rebecca).
```

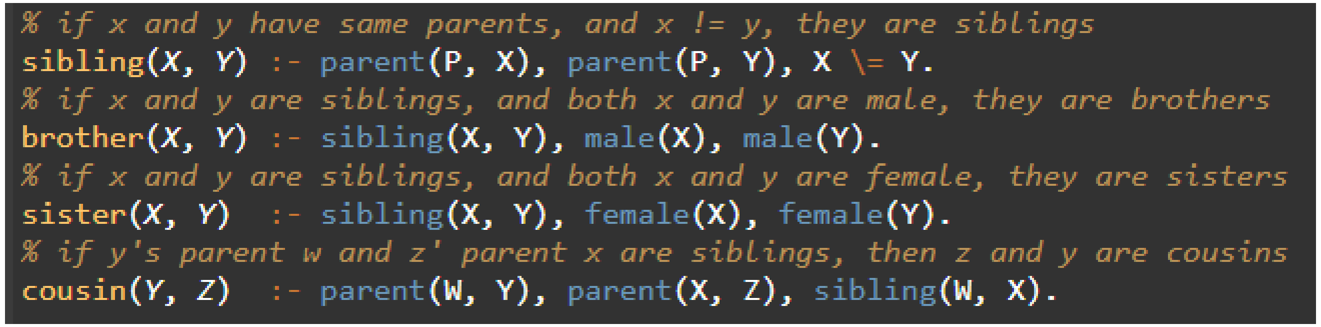
- Building complete parents relation



```
% Rule
parent(X, Y) :- direct_parent(X, Y).
% the child has two parents
parent(Y, Z) :- direct_parent(X, Z), married(X, Y).
```

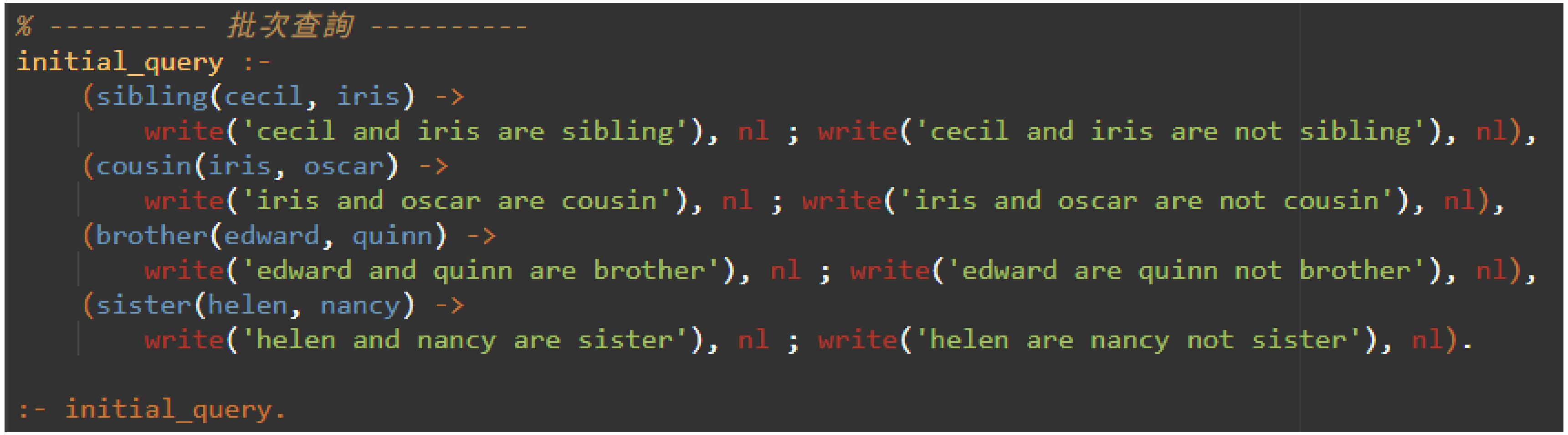
Prolog program listing cont.

- Check relationship



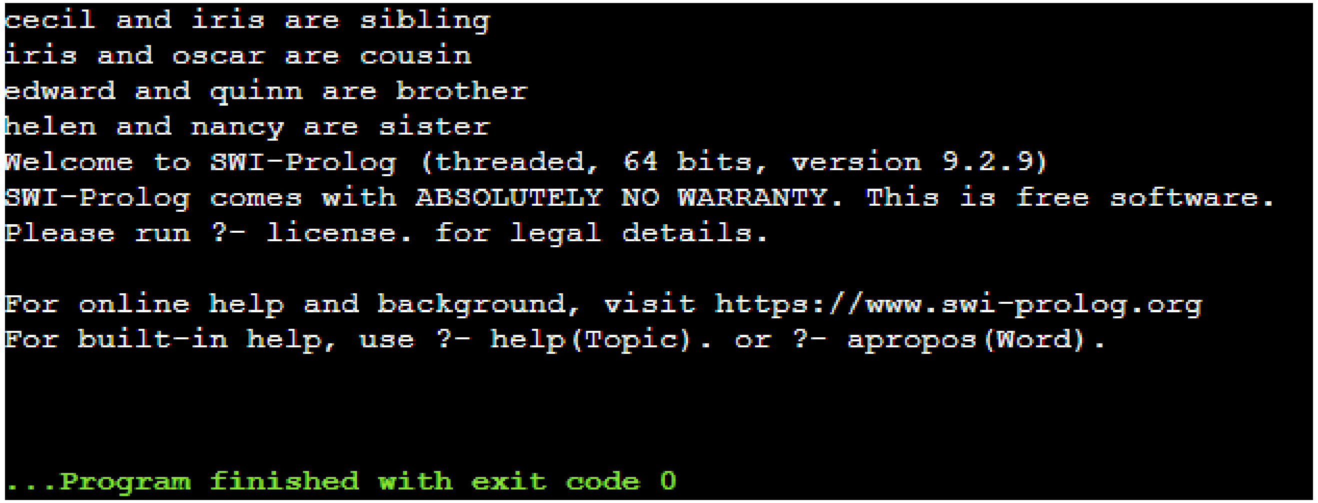
```prolog
% if x and y have same parents, and x != y, they are siblings
sibling(X, Y) :- parent(P, X), parent(P, Y), X \= Y.
% if x and y are siblings, and both x and y are male, they are brothers
brother(X, Y) :- sibling(X, Y), male(X), male(Y).
% if x and y are siblings, and both x and y are female, they are sisters
sister(X, Y)  :- sibling(X, Y), female(X), female(Y).
% if y's parent w and z' parent x are siblings, then z and y are cousins
cousin(Y, Z)  :- parent(W, Y), parent(X, Z), sibling(W, X).
```

- Input test cases



```prolog
% ---------- 批次查詢 ----------
initial_query :-
    (sibling(cecil, iris) ->
        write('cecil and iris are sibling'), nl ; write('cecil and iris are not sibling'), nl),
    (cousin(iris, oscar) ->
        write('iris and oscar are cousin'), nl ; write('iris and oscar are not cousin'), nl),
    (brother(edward, quinn) ->
        write('edward and quinn are brother'), nl ; write('edward are quinn not brother'), nl),
    (sister(helen, nancy) ->
        write('helen and nancy are sister'), nl ; write('helen are nancy not sister'), nl).

:- initial_query.
```

Prolog test run results



```
cecil and iris are sibling
iris and oscar are cousin
edward and quinn are brother
helen and nancy are sister
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).



...Program finished with exit code 0
```

Problem 2 description

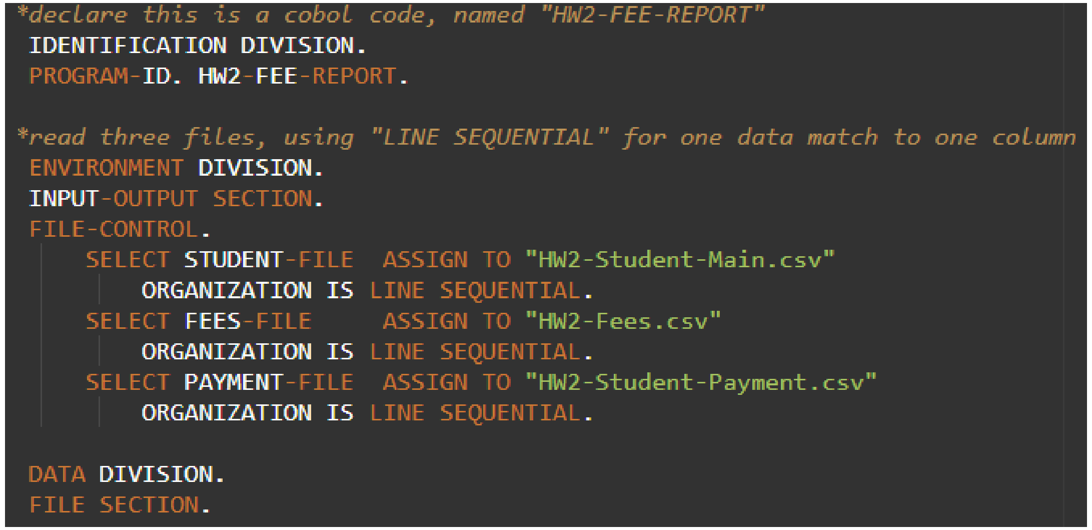
- Given three files, giving the information about student id, how much they paid already, how much should they pay.
- Output the total amount received from students before due and list all the students that did not pay the required fees with the amount short.
- Using language: Cobol, R.

Cobol

- Open three files, and process the data with “LINE SEQUENTIAL“.
- Save student id, name, payment type and other data.
- Check students’ payment:
  - Process all the column, match the payment type, search for how much that student had already paid.
  - Add total students payment.
  - If paid isn’t enough, display.
- Display:
  - Students that didn’t pay enough.
  - Total students paid amount.

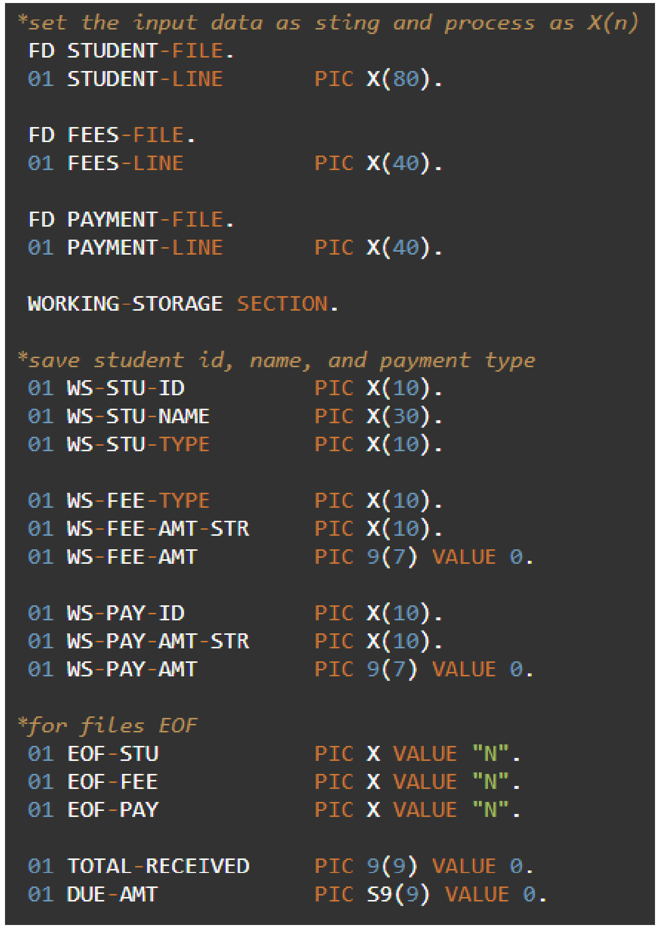
Cobol program listing

- Read files



```
*declare this is a cobol code, named "HW2-FEE-REPORT"
 IDENTIFICATION DIVISION.
 PROGRAM-ID. HW2-FEE-REPORT.

*read three files, using "LINE SEQUENTIAL" for one data match to one column
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT STUDENT-FILE  ASSIGN TO "HW2-Student-Main.csv"
         ORGANIZATION IS LINE SEQUENTIAL.
     SELECT FEES-FILE     ASSIGN TO "HW2-Fees.csv"
         ORGANIZATION IS LINE SEQUENTIAL.
     SELECT PAYMENT-FILE  ASSIGN TO "HW2-Student-Payment.csv"
         ORGANIZATION IS LINE SEQUENTIAL.

 DATA DIVISION.
 FILE SECTION.
```

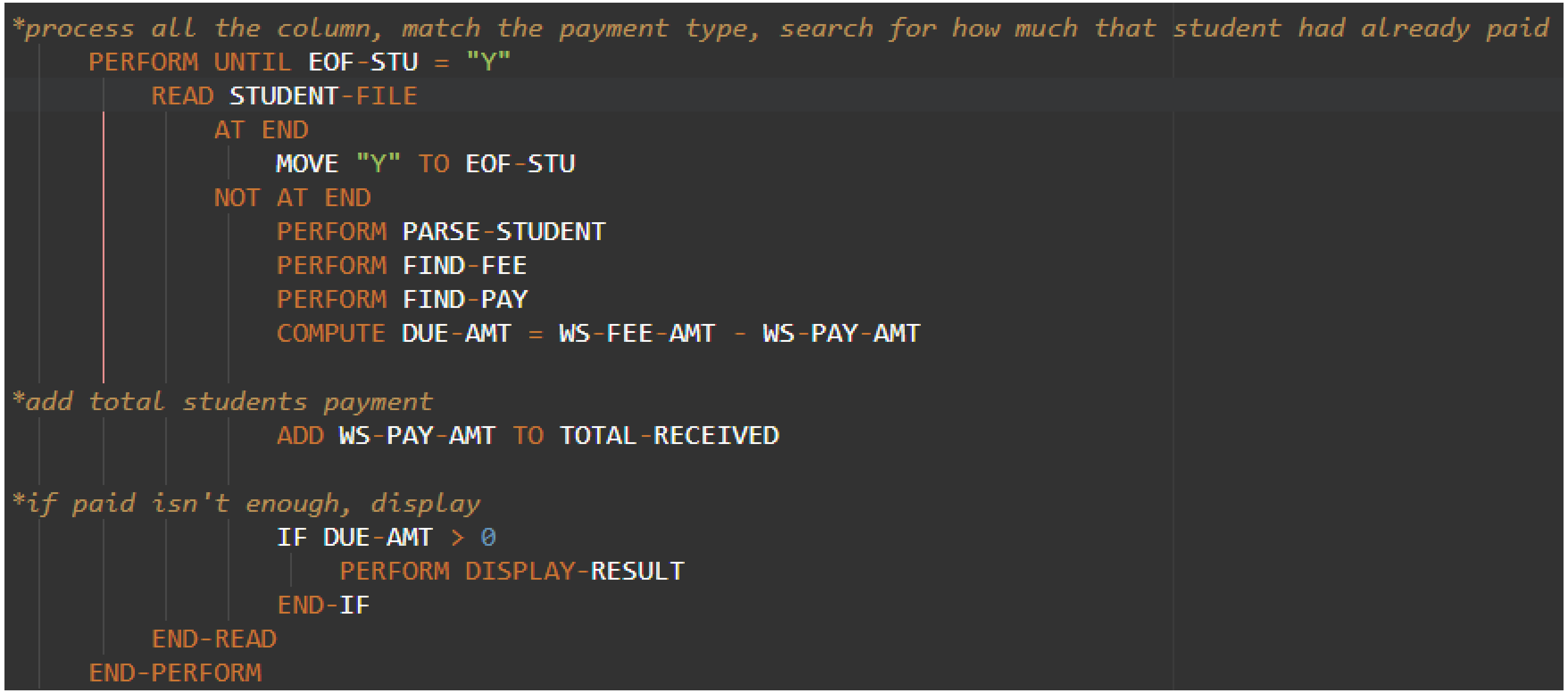
- Process files’ data



```
*set the input data as sting and process as X(n)
 FD STUDENT-FILE.
 01 STUDENT-LINE       PIC X(80).

 FD FEES-FILE.
 01 FEES-LINE          PIC X(40).

 FD PAYMENT-FILE.
 01 PAYMENT-LINE       PIC X(40).

 WORKING-STORAGE SECTION.

*save student id, name, and payment type
 01 WS-STU-ID          PIC X(10).
 01 WS-STU-NAME        PIC X(30).
 01 WS-STU-TYPE        PIC X(10).

 01 WS-FEE-TYPE        PIC X(10).
 01 WS-FEE-AMT-STR     PIC X(10).
 01 WS-FEE-AMT         PIC 9(7) VALUE 0.

 01 WS-PAY-ID          PIC X(10).
 01 WS-PAY-AMT-STR     PIC X(10).
 01 WS-PAY-AMT         PIC 9(7) VALUE 0.

*for files EOF
 01 EOF-STU            PIC X VALUE "N".
 01 EOF-FEE            PIC X VALUE "N".
 01 EOF-PAY            PIC X VALUE "N".

 01 TOTAL-RECEIVED     PIC 9(9) VALUE 0.
 01 DUE-AMT            PIC S9(9) VALUE 0.
```

Cobol program listing cont.

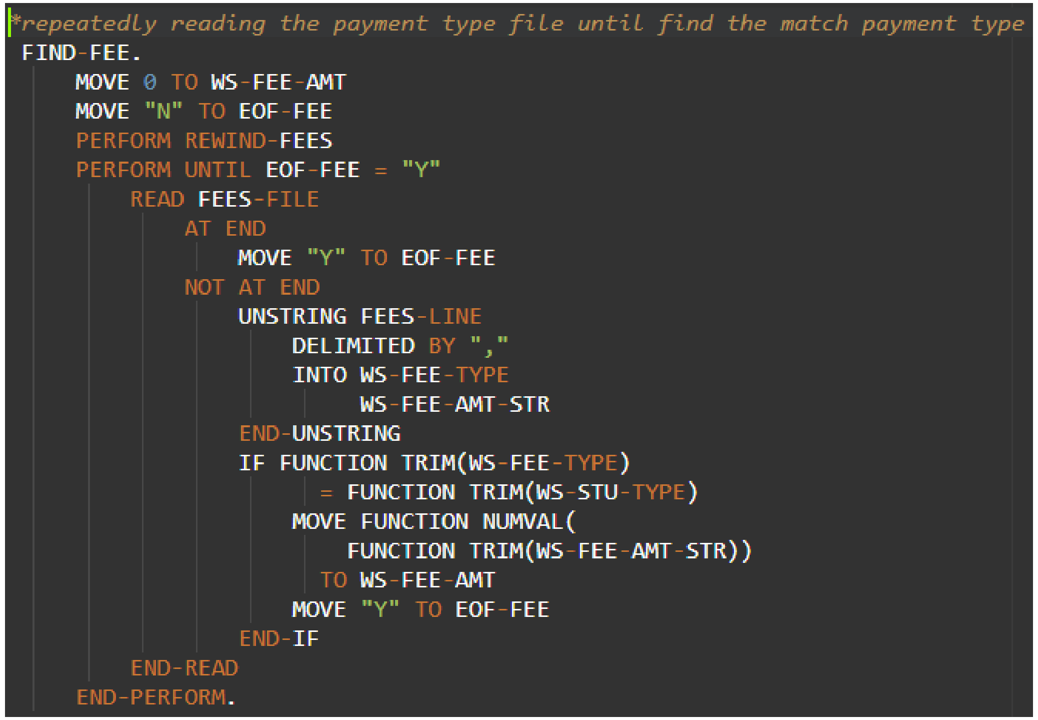
- Search payment type, add total payment, display students that didn’t pay enough.



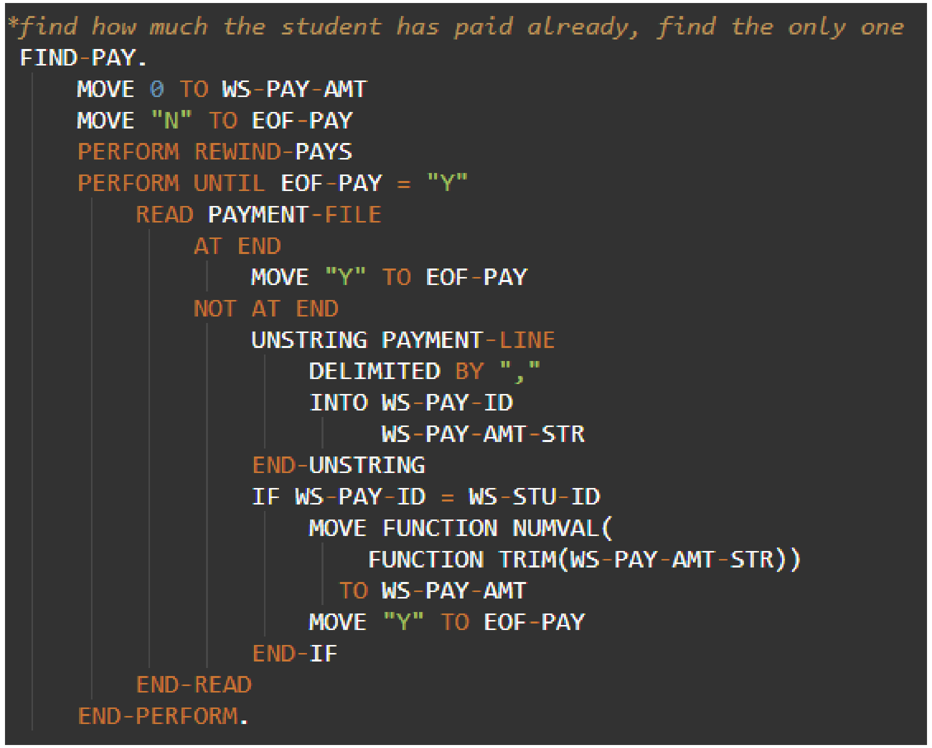
```
*process all the column, match the payment type, search for how much that student had already paid
     PERFORM UNTIL EOF-STU = "Y"
         READ STUDENT-FILE
             AT END
                 MOVE "Y" TO EOF-STU
             NOT AT END
                 PERFORM PARSE-STUDENT
                 PERFORM FIND-FEE
                 PERFORM FIND-PAY
                 COMPUTE DUE-AMT = WS-FEE-AMT - WS-PAY-AMT

*add total students payment
                 ADD WS-PAY-AMT TO TOTAL-RECEIVED

*if paid isn't enough, display
                 IF DUE-AMT > 0
                     PERFORM DISPLAY-RESULT
                 END-IF
         END-READ
     END-PERFORM
```

Cobol program listing cont.

- Subprocess: find how much students should pay and how much they already paid.

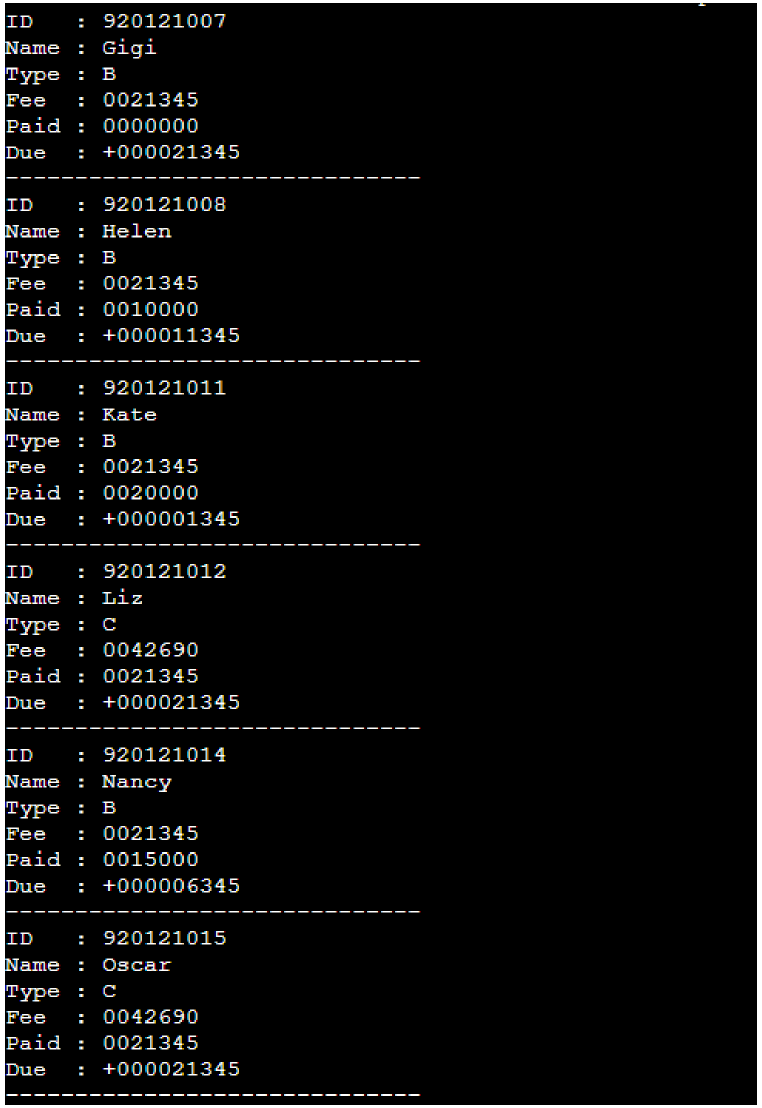


```
*repeatedly reading the payment type file until find the match payment type
 FIND-FEE.
     MOVE 0 TO WS-FEE-AMT
     MOVE "N" TO EOF-FEE
     PERFORM REWIND-FEES
     PERFORM UNTIL EOF-FEE = "Y"
         READ FEES-FILE
             AT END
                 MOVE "Y" TO EOF-FEE
             NOT AT END
                 UNSTRING FEES-LINE
                     DELIMITED BY ","
                     INTO WS-FEE-TYPE
                          WS-FEE-AMT-STR
                 END-UNSTRING
                 IF FUNCTION TRIM(WS-FEE-TYPE)
                       = FUNCTION TRIM(WS-STU-TYPE)
                     MOVE FUNCTION NUMVAL(
                         FUNCTION TRIM(WS-FEE-AMT-STR))
                       TO WS-FEE-AMT
                     MOVE "Y" TO EOF-FEE
                 END-IF
         END-READ
     END-PERFORM.
```

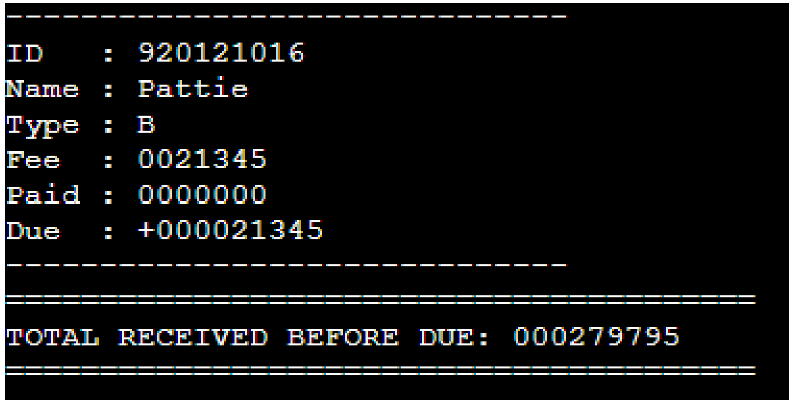


```
*find how much the student has paid already, find the only one
 FIND-PAY.
     MOVE 0 TO WS-PAY-AMT
     MOVE "N" TO EOF-PAY
     PERFORM REWIND-PAYS
     PERFORM UNTIL EOF-PAY = "Y"
         READ PAYMENT-FILE
             AT END
                 MOVE "Y" TO EOF-PAY
             NOT AT END
                 UNSTRING PAYMENT-LINE
                     DELIMITED BY ","
                     INTO WS-PAY-ID
                          WS-PAY-AMT-STR
                 END-UNSTRING
                 IF WS-PAY-ID = WS-STU-ID
                     MOVE FUNCTION NUMVAL(
                         FUNCTION TRIM(WS-PAY-AMT-STR))
                       TO WS-PAY-AMT
                     MOVE "Y" TO EOF-PAY
                 END-IF
         END-READ
     END-PERFORM.
```

Cobol test run results



```
ID   : 920121007
Name : Gigi
Type : B
Fee  : 0021345
Paid : 0000000
Due  : +000021345
------------------------------
ID   : 920121008
Name : Helen
Type : B
Fee  : 0021345
Paid : 0010000
Due  : +000011345
------------------------------
ID   : 920121011
Name : Kate
Type : B
Fee  : 0021345
Paid : 0020000
Due  : +000001345
------------------------------
ID   : 920121012
Name : Liz
Type : C
Fee  : 0042690
Paid : 0021345
Due  : +000021345
------------------------------
ID   : 920121014
Name : Nancy
Type : B
Fee  : 0021345
Paid : 0015000
Due  : +000006345
------------------------------
ID   : 920121015
Name : Oscar
Type : C
Fee  : 0042690
Paid : 0021345
Due  : +000021345
------------------------------
```



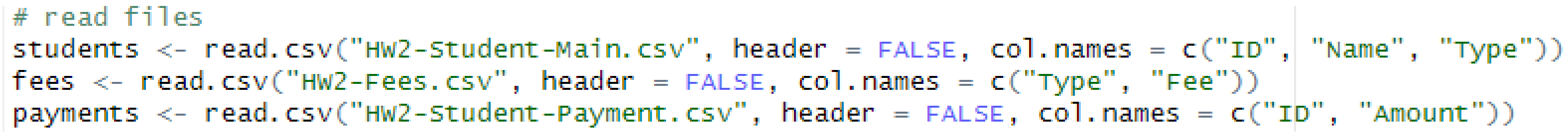
```
------------------------------
ID   : 920121016
Name : Pattie
Type : B
Fee  : 0021345
Paid : 0000000
Due  : +000021345
------------------------------
========================================
TOTAL RECEIVED BEFORE DUE: 000279795
========================================
```

R

- Read files.
- Make sure data types are right.
- Process data:
    - if a student have more than one payment data, add them.
    - based on the payment type, combine the student data and how much should they pay.
    - combine student data and how much they already paid.
- Calculate arrears.
- Display list and total pay amount.

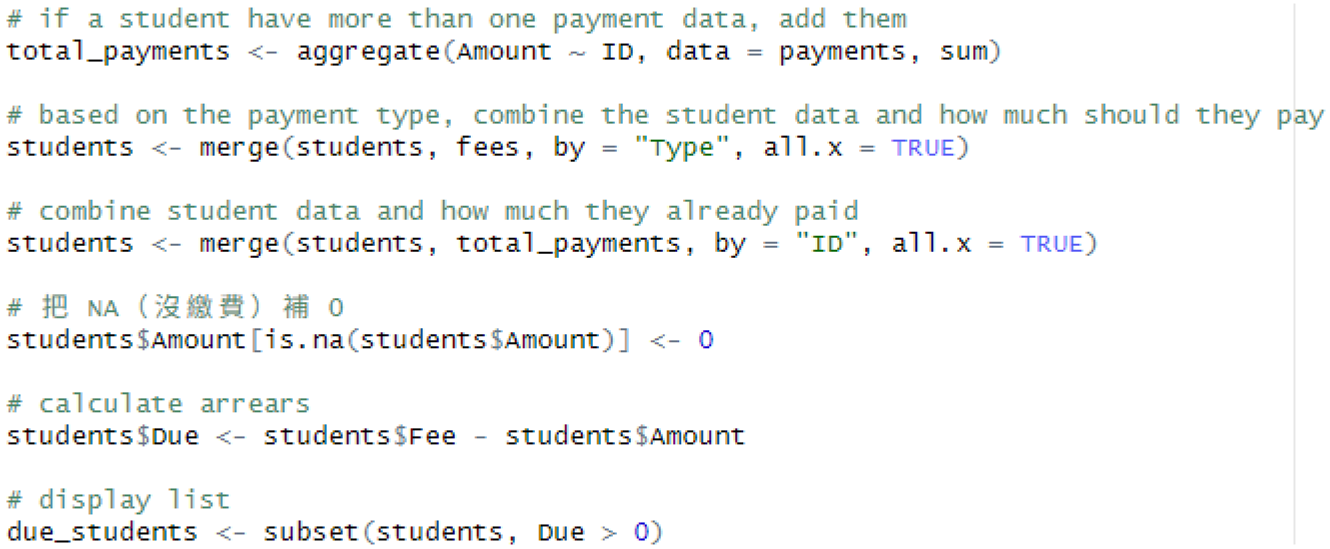
R program listing

- Read files



```
# read files
students <- read.csv("HW2-Student-Main.csv", header = FALSE, col.names = c("ID", "Name", "Type"))
fees <- read.csv("HW2-Fees.csv", header = FALSE, col.names = c("Type", "Fee"))
payments <- read.csv("HW2-Student-Payment.csv", header = FALSE, col.names = c("ID", "Amount"))
```

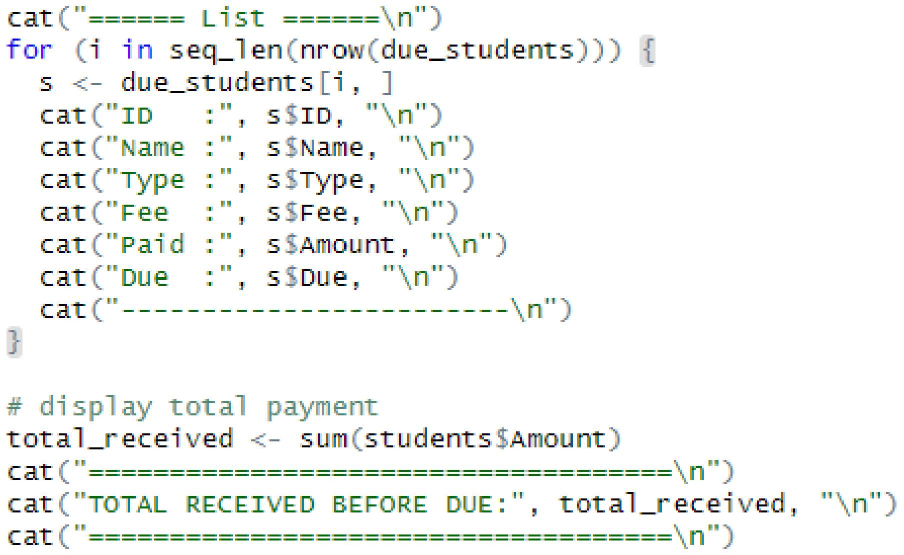
- Process data



```
# if a student have more than one payment data, add them
total_payments <- aggregate(Amount ~ ID, data = payments, sum)

# based on the payment type, combine the student data and how much should they pay
students <- merge(students, fees, by = "Type", all.x = TRUE)

# combine student data and how much they already paid
students <- merge(students, total_payments, by = "ID", all.x = TRUE)

# 把 NA（沒繳費）補 0
students$Amount[is.na(students$Amount)] <- 0

# calculate arrears
students$Due <- students$Fee - students$Amount

# display list
due_students <- subset(students, Due > 0)
```

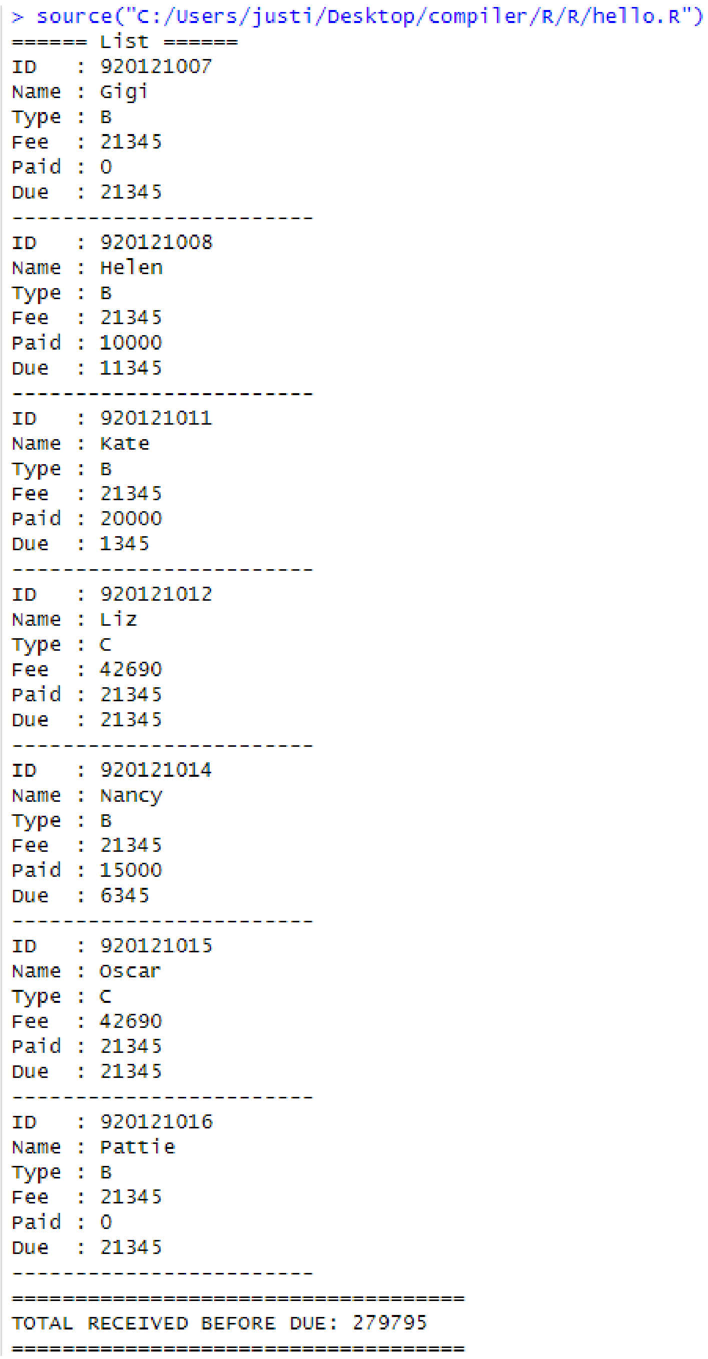
R program listing cont.

- Display



```r
cat("====== List ======\n")
for (i in seq_len(nrow(due_students))) {
  s <- due_students[i, ]
  cat("ID   :", s$ID, "\n")
  cat("Name :", s$Name, "\n")
  cat("Type :", s$Type, "\n")
  cat("Fee  :", s$Fee, "\n")
  cat("Paid :", s$Amount, "\n")
  cat("Due  :", s$Due, "\n")
  cat("------------------------\n")
}

# display total payment
total_received <- sum(students$Amount)
cat("====================================\n")
cat("TOTAL RECEIVED BEFORE DUE:", total_received, "\n")
cat("====================================\n")
```

R test run results



```
> source("C:/Users/justi/Desktop/compiler/R/R/hello.R")
====== List ======
ID   : 920121007
Name : Gigi
Type : B
Fee  : 21345
Paid : 0
Due  : 21345
------------------------
ID   : 920121008
Name : Helen
Type : B
Fee  : 21345
Paid : 10000
Due  : 11345
------------------------
ID   : 920121011
Name : Kate
Type : B
Fee  : 21345
Paid : 20000
Due  : 1345
------------------------
ID   : 920121012
Name : Liz
Type : C
Fee  : 42690
Paid : 21345
Due  : 21345
------------------------
ID   : 920121014
Name : Nancy
Type : B
Fee  : 21345
Paid : 15000
Due  : 6345
------------------------
ID   : 920121015
Name : Oscar
Type : C
Fee  : 42690
Paid : 21345
Due  : 21345
------------------------
ID   : 920121016
Name : Pattie
Type : B
Fee  : 21345
Paid : 0
Due  : 21345
------------------------
====================================
TOTAL RECEIVED BEFORE DUE: 279795
====================================
```

Discussion

- In question 1, we use Java, ML and Prolog to find the relationship between two people.
- Java is imperative language, most of the code are based on OOT.
  - Advantage: Clear structure and high maintainability.
  - Disadvantage: The reasoning logic is lengthy and is not suitable for knowledge expression.
- ML is functional programming language, suitable in logic processing.
  - Advantage: Suitable for writing clear logical rules.
  - Disadvantage: Compared to Prolog, lacks built-in automatic reasoning capabilities

Discussion cont.

- Prolog is logical language, reasoning through rules and facts.
  - Advantage: Can directly express relationships and rules, the writing method is concise and intuitive.
  - Disadvantage: Hard to learn for the beginner.
- In question 2, we use Cobol and R to read multiple files and calculate the students’ payment.
- Cobol is imperative language, often used in business.
- Cobol in question 2:
  - Suitable for processing structured text files (such as CSV).
  - Use FILE SECTION to read files and WORKING-STORAGE to temporarily store data.
  - All logic is clearly listed, such as "read line by line → analyze by segment → accumulate → display results“.
  - Suitable for processing the fixed process of traditional accounting systems.

Discussion cont.

- R is a functional + declarative language, often used in statistical computing and data analysis.
- R in question 2:
  - Use “read.csv()” to load data in one line, which is fast.
  - Provides “merge()”, “aggregate()” and other functions to associate and aggregate data.
  - Suitable for exploratory data analysis and report writing.