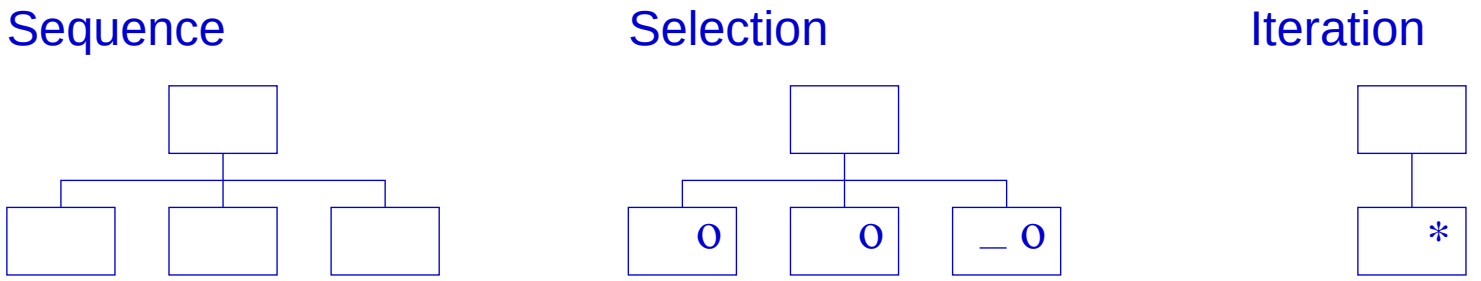
8 Jackson structured programming (JSP)

Jackson structured programming

- Developed in early 1970s, widely used in '80s and '90s
- JSP is a program design method, for systems which:
  - are realisable as a single sequential process
  - have well-defined input and output data streams
- Good for data processing applications
- Jackson structured programming only uses one kind of diagram – the Jackson structure diagram (JSD)
- So, the JSD describes both static data structure and dynamic program function

8.1 Jackson structure diagrams

- Rules for drawing JSDs:
  - *Sequence* is represented by unmarked boxes, runs from left to right
  - *Selection* is represented by boxes marked with circles, last one is the default (i.e., conditionless or ELSE)
  - *Iteration* is represented by an asterisked box
  - these three forms cannot be mixed at a branch



Components of a Jackson structure diagram.

8.2 JSP procedure

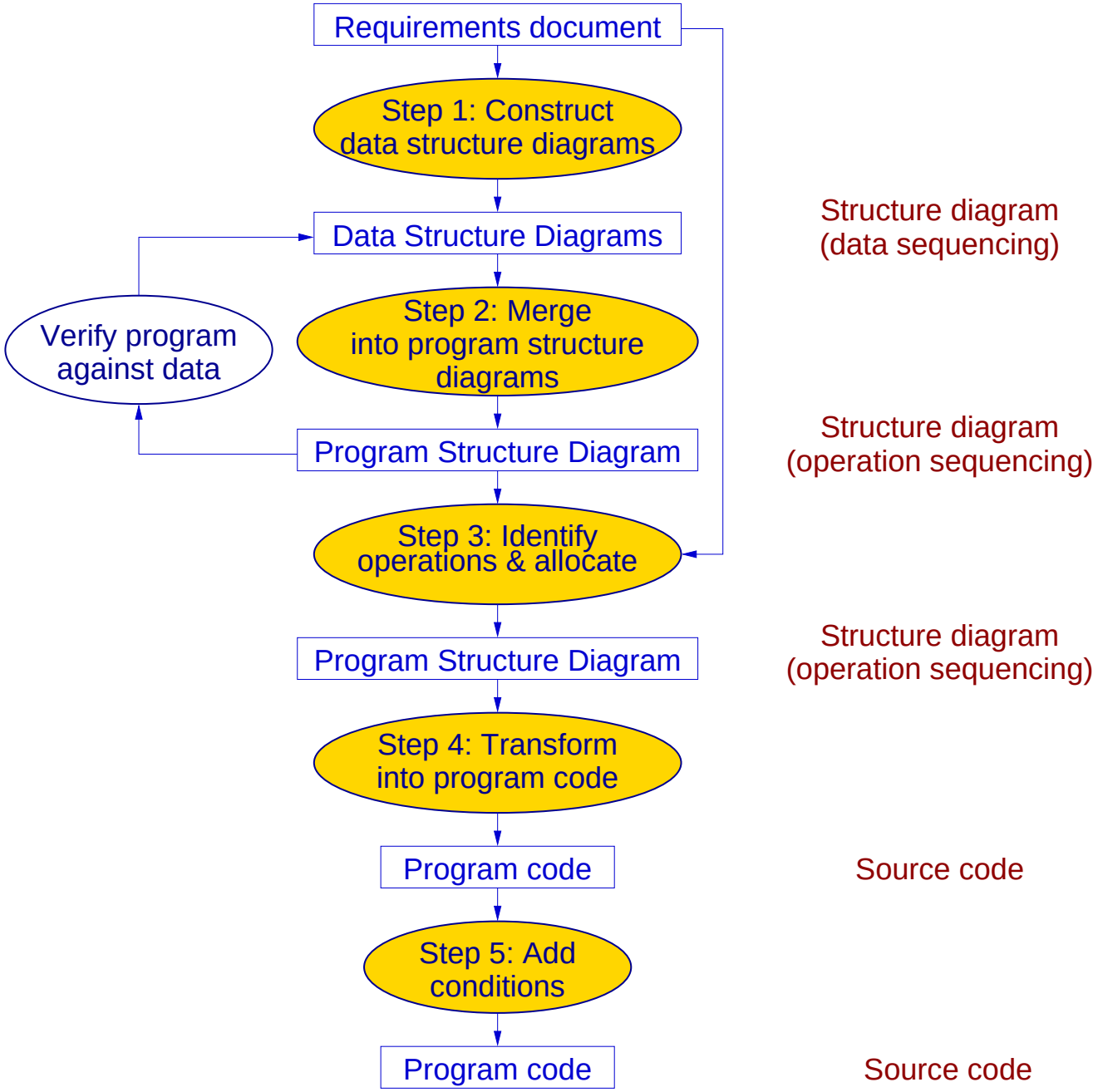
1. *Draw* a JSD for each input and output data stream

2. *Merge* all of these to form the program JSD

3. *List* operations that the program needs to perform and allocate each one to a block in the program JSD

4. *Convert* the program blocks into text in order, without specific conditions for any of the decision points

5. *Add* the conditions for each selection and iteration



Flow diagram of the JSP process.

JSP characteristics

- JSP emphasises algorithm development, so it uses JSDs in conjunction with pseudo-code process specifications
- Result is program structure that reflects task structure
  - JSP decomposes procedures
  - It does not address modularity
- Step 2 transforms from static data sequences into a dynamic (time-ordered) operation sequence
  - First two steps take over half the total design effort
- Only one kind of diagram used, but its *interpretation* changes

Example: A petrol filling station has a number of self-service pumps, each of which can be used to dispense both diesel and unleaded petrol. There is a small local computer in each pump that maintains the display of price and volume on the pump; when the customer returns the pump nozzle to its socket, this computer sends a record to the cashier's console computer, containing the details of the current transaction as the sequence:

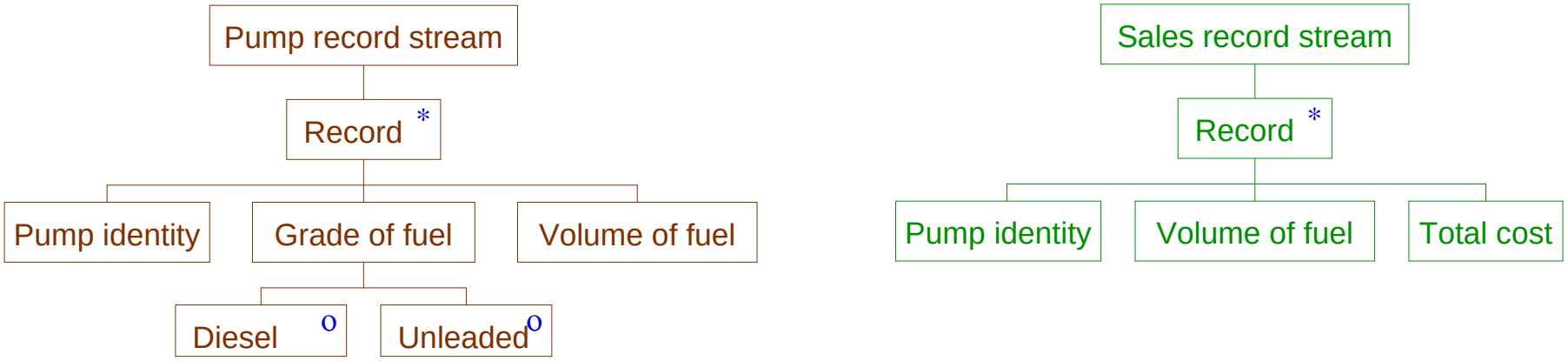
pump identity; grade of fuel; volume of fuel

The problem is to design the software that receives these messages. For each message it receives, the program is required to generate a line on the printer in front of the cashier, giving the details of the transaction in terms of

pump identity; volume of fuel; total cost

Step 1: Draw the input/output JSDs

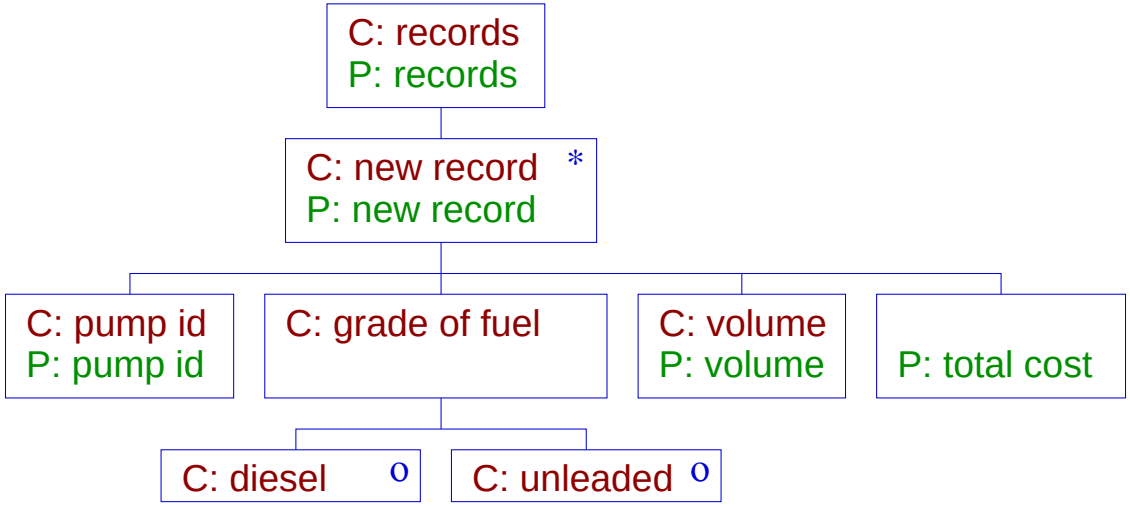
- In this case, single input stream and single output stream:
  - input stream previously set by pump designers
  - output data stream also well-defined



JSDs of the input (left) and output (right), for the sales record.

Step 2: Create the JSD for the program

- Merged directly since input & output JSDs are similar
- Compositional approach brings input & output together



Program structure based on the inputs and outputs.

- verification – check that program tree is consistent with consuming input (C), and producing output (P)

Step 3: List operations & allocate to program blocks

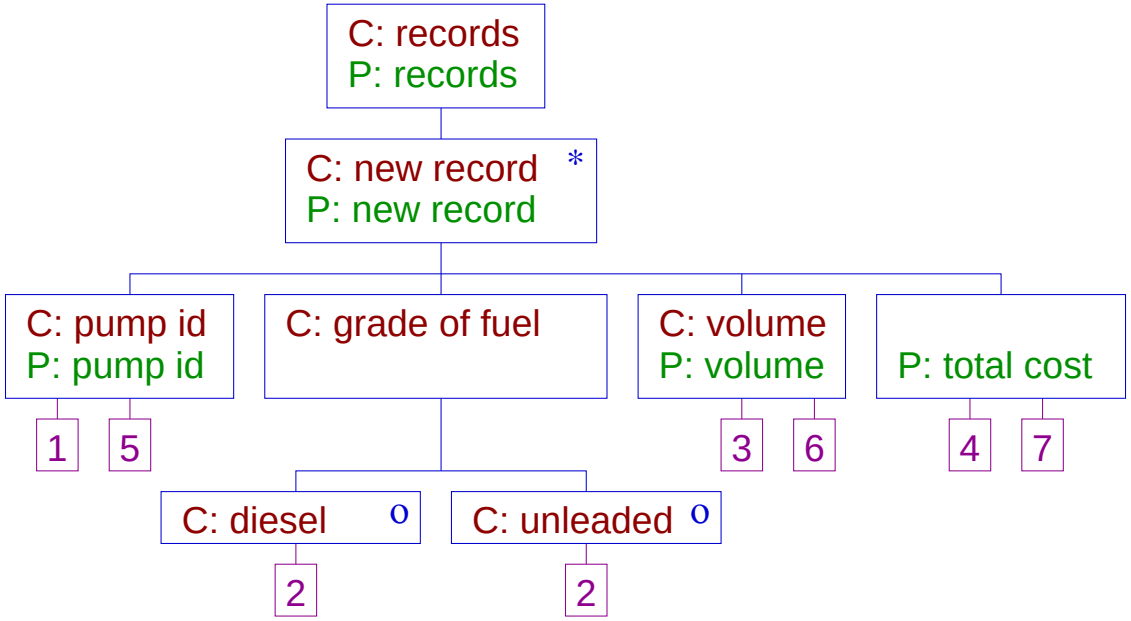
Inputs:

1. obtain pump identity
2. obtain grade of fuel
3. obtain volume of fuel
4. multiply grade price per unit by volume dispensed

Outputs:

5. write pump identity
6. write volume of fuel
7. write cost to customer

- Allocate to leaf elements in Structure Diagram:



Allocation of operations to elements in the structure diagram.

Step 4: Convert to text

- Outline program:

```
open records;
get pump identity;
print pump identity;
get grade of fuel;
get volume of fuel;
print volume of fuel;
calculate total cost;
print total cost;
close records;
```

Extra actions added

Step 4: Convert to text

- Outline program:

```
open records;
get pump identity;
print pump identity;
get grade of fuel;
get volume of fuel;
print volume of fuel;
calculate total cost;
print total cost;
close records;
```

Step 5: Add conditions

- With conditional information:

```
open records;
FOR each record,
    get pump identity;
    print pump identity;
    get grade of fuel;
    get volume of fuel;
    print volume of fuel;
    IF grade == diesel,
        THEN cost = dieselPrice * volume;
        ELSE cost = unleadedPrice * volume;
    ENDIF
    print cost;
ENDFOR
close records;
```

Finally, translated from pseudocode into programming language

8.3 Program inversion and structure clashes

- Problems occur in JSP as assumptions are made about mappings of data structures.

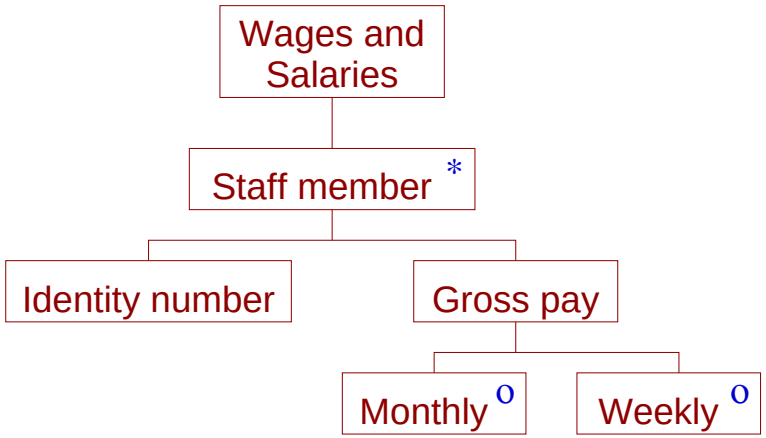
  Example: A company gives its work force a long break at Christmas and New Year, but does not want to provide pay in advance. So to pay its workers during the break it must post cheques to their home addresses. The Company employs people both on a permanent and a part-time basis. Permanent employees are paid monthly and part-time workers are paid weekly. All employees have a unique staff identity number, allocated to them when they join the company.

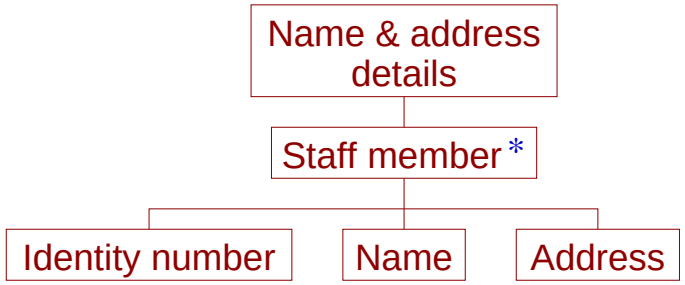
Step 1: Input and output structures

Input 1: Finance department provides a payments file

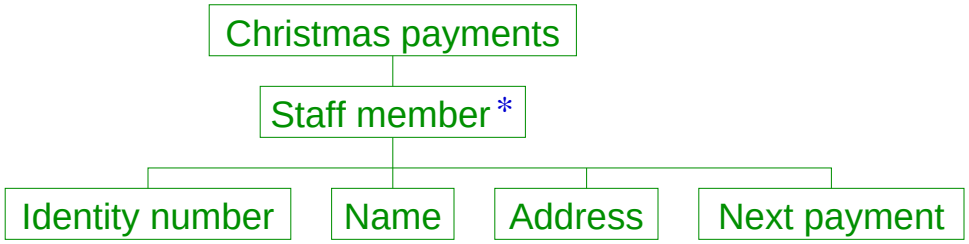
Input 2: Personnel department provides personal details

Output: Information needed for each posted cheque



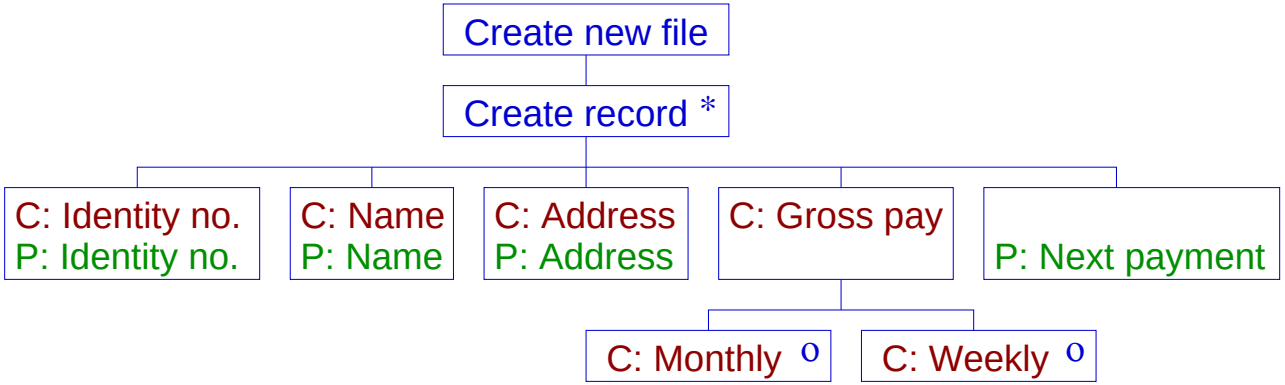


Structure of input files: payments file (left) and employee details (right).



Structure of the output file for the Christmas payments.

Step 2: Merged program structure



Derived JSD of the program.

More complicated design, but is it finished?

Structure clash!

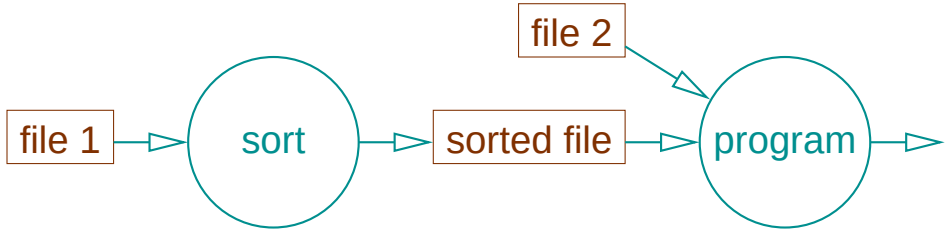
```
009003  M  £1500.00
008607  W  £ 200.00
008331  M  £1240.00
......  .  ........
......  .  ........
......  .  ........
......
```

```
008607  Charlie Chips  9 Cabins Way
008721  Les Logjam     15 Sawpit Lane
009003  Sam Sawdust    14 Timber Road
......  ............   ..............
......  ............   ..............
......  ............   ..............
......  ........
```

Data structures for finance (left) and personnel (right) are inconsistently ordered, producing a *ordering clash*.

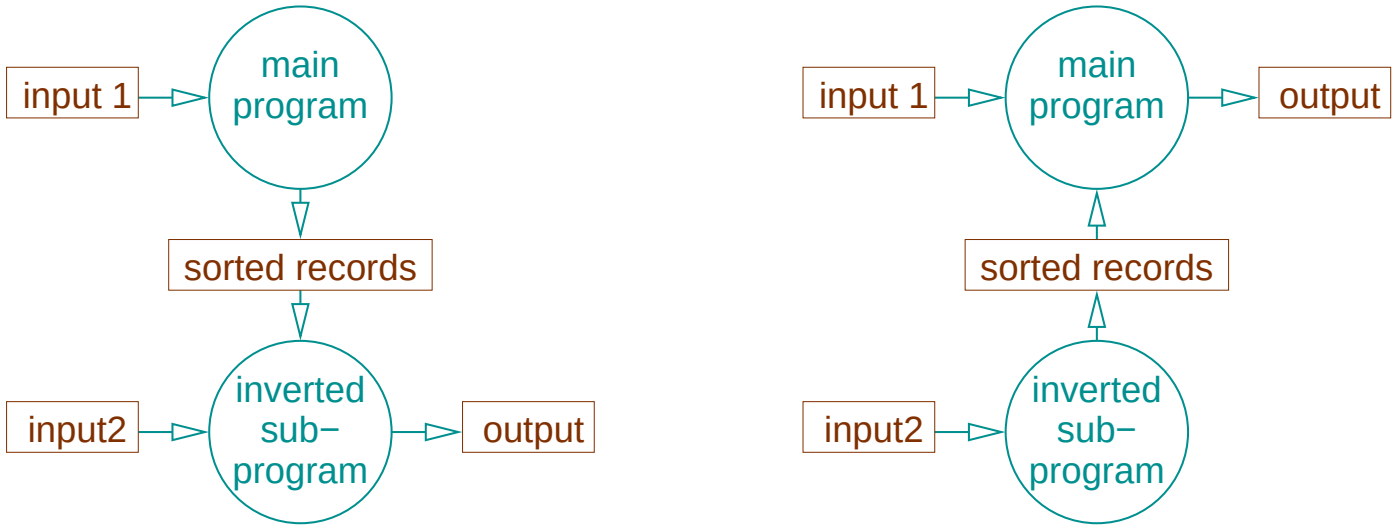
- Types of structure clash:
  - *Ordering clash*: data sorted in different ways
  - *Boundary clash*: data grouped by different criteria e.g., text files are variable-length character strings (high-level), stored in fixed-size disk blocks (low level)
  - *Multi-threading clash*: data units overlap in input file e.g., converting stereo wave file into mono files
- Clashes occur due to assumptions implicit in JSP

- *Program inversion* sorts one input to match the other:



Removal of ordering clash by creating an intermediate file.

- Sorted data can be passed directly to main program, i.e., one program is inverted (as the other's subroutine):



Alternative program-inversion structures:
(left) main program sorts, (right) subprogram sorts.

8 Summary of JSP

- Jackson Structured Programming steps using JSDs:

  1. *Draw* JSD for inputs/outputs
  2. *Merge* into program JSD
  3. *List* operations & allocate
  4. *Convert* sequence of blocks to text
  5. *Add* conditions for selection & iteration

- Structure clashes

  – Ordering, Boundary, Multi-threading

  – Program inversion