How to THINK like a Programmer

Problem Solving for the Bewildered

Paul Vickers

chapter 5

calculating and keeping track of things

Purpose

- This chapter is concerned with working storage and the need to write down and keep track of values
  - Variables
  - Arithmetic
  - High-level data abstractions

Working storage

- Original solution for adding sugar (chapter 4):

```
WHILE (sugar required)
    Add spoonful of sugar ;
ENDWHILE
```

- Needed more information to know when enough sugar had been added. Keeping count allows this:

```
WHILE (sugars added not equal to number required)
    Add spoonful of sugar ;
    Add 1 to number of sugars added ;
ENDWHILE
```

- May sound long-winded but the computer is stupid and needs **everything** spelling out

Variables

- We may keep track of different items in our head or even on a piece of paper
  - Ask 12 guests what they want to drink: keep track of number of teas, number of coffees, number of sodas, etc.
- In a program we use **variables** to keep track of values
- Variable is an item of information whose value can change over time (the value can **vary**, hence **variable**)
- A variable has two properties:
  - A name or **identifier**
  - A value
- Similar to algebra where x is a variable

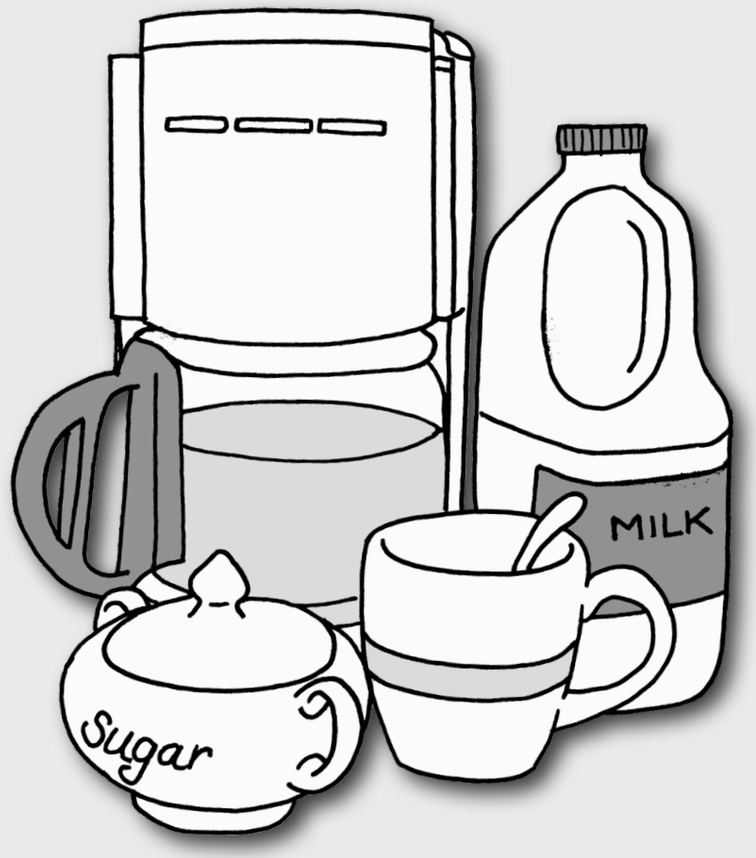
ACTIVITY

What are the variables in the coffee making problem? That is, what values do we need to keep track of?

Using an electric filter machine (also called a percolator) make coffee for up to six guests. Add milk and sugar as required

Solving the problem

- Principal parts of problem
  - 1) Make a pot of coffee. 2) Pour several cups. 3) Each cup may or may not need milk or sugar.
  - Or, looking at it another way: 1) The filter machine, 2) the cups, 3) the coffee, 4) some water, 5) the cups of coffee–just coffee, or milk and sugar as well?



First draft solution


```
1.  Put water in coffee machine ;
2.  Open coffee holder
3.  Put filter paper in machine ;
4.  Measure coffee ;
5.  Put coffee into filter paper ;
6.  Shut the coffee holder ;
7.  Turn on machine ;
8.  Wait for coffee to filter through ;
9.  Find out how many sugars required ;
10. Find out whether milk required ;
11. WHILE (sugars added not equal to number required)
      11.1  Add spoonful of sugar ;
      11.2  Add 1 to number of sugars added ;
    ENDWHILE
12. IF (white coffee required)
      12.1  Add milk/cream ;
    ENDIF
13. Pour coffee into mug ;
14. Stir coffee ;
15. Turn off machine ;
```

Blue statements deal with making coffee
Red statements deal with processing a single cup

To make up to six cups tasks 9–14 need to be repeated

More than 1 cup


```
1.  WHILE (cups of coffee required)
      1.1  Find out how many sugars required ;
      1.2  Find out whether milk required ;
      1.3  WHILE (sugars added not equal to number required)
             1.3.1  Add spoonful of sugar ;
             1.3.2  Add 1 to number of sugars added ;
           ENDWHILE
      1.4  IF (white coffee required)
             1.4.1  Add milk/cream ;
           ENDIF
      1.5  Pour coffee into mug ;
      1.6  Stir coffee ;
    ENDWHILE
```

- Notice the construct nesting (1.3,1.4): constructs within constructs

ACTIVITY

Trace through the solution from the previous algorithm going through the first `WHILE` six times, using different milk & sugar requirements each time

```
1.  WHILE (cups of coffee required)
      1.1  Find out how many sugars required ;
      1.2  Find out whether milk required ;
      1.3  WHILE (sugars added not equal to number required)
             1.3.1  Add spoonful of sugar ;
             1.3.2  Add 1 to number of sugars added ;
           ENDWHILE
      1.4  IF (white coffee required)
             1.4.1  Add milk/cream ;
           ENDIF
      1.5  Pour coffee into mug ;
      1.6  Stir coffee ;
ENDWHILE
```

How can we keep track of how many coffees have been made?

Counting the cups

```
1.  Find out how many coffees required ;
2.  WHILE (cups poured not equal to cups required)
      2.1  Find out how many sugars required ;
      2.2  Find out whether milk required ;
      2.3  WHILE (sugars added not equal to number required)
             2.3.1  Add spoonful of sugar ;
             2.3.2  Add 1 to number of sugars added ;
           ENDWHILE
    2.4  IF (white coffee required)
             2.4.1  Add milk/cream ;
           ENDIF
    2.5  Pour coffee into mug ;
    2.6  Stir coffee ;
    2.7  Add 1 to number of cups poured
  ENDWHILE
```

Subproblem: How much coffee?

- Amount of coffee & water differs according to number of cups required
- Previous solution only measured enough coffee for one cup so here's an algorithm to make the pot:

```
1.  Find out how many cups are required ;
2.  Put water for number of cups required
    in coffee machine ;
3.  Open coffee holder ;
4.  Put filter paper in machine ;
5.  Measure coffee for number required ;
6.  Put coffee into filter paper ;
7.  Shut the coffee holder ;
8.  Turn on machine ;
9.  Wait for coffee to filter through
```

ACTIVITY

Is the solution below complete/sufficient? Try it out.

```
// Set up coffee machine
1.  Find out how many cups are required ;
2.  Put water for number of cups required in coffee machine ;
3.  Open coffee holder ;
4.  Put filter paper in machine ;
5.  Measure coffee for number required ;
6.  Put coffee into filter paper ;
7.  Shut the coffee holder ;
8.  Turn on machine ;
9.  Wait for coffee to filter through
// Process the cups of coffee
10. WHILE (cups poured not equal to cups required)
      // Add  sugar and milk as necessary
      10.1  Find out how many sugars required ;
      10.2  Find out whether milk required ;
      10.3  WHILE (sugars added not equal to number required)
              10.3.1  Add spoonful of sugar ;
              10.3.2  Add 1 to number of sugars added ;
            ENDWHILE
      10.4  IF (white coffee required)
              10.4.1  Add milk/cream ;
            ENDIF
      10.5  Pour coffee into mug ;
      10.6  Stir coffee ;
      10.7  Add 1 to number of cups poured
    ENDWHILE
11. Turn off machine ;
```

Residual problem

- The algorithm isn't **limited** to six cups
  - What happens if we try and make 10 cups? 30 cups? When will the pot overflow?
  - Is the algorithm based on any explicit assumptions? Hidden assumptions?

ACTIVITY

Think about the different ways you could prevent Solution 5.5 from being applied to more than six cups of coffee. Any problem always has more than one solution, so write down some possible ways of solving this problem. Here are some hints to get you started:

- Consider rephrasing the condition in the `WHILE` (Task #10)
- Can you restrict carrying out Tasks #2 through #11 in some way?
- How about putting some conditions on Task #1?

Candidate solutions

- How about this?

```
1.  Find out how many cups are required ;
2.  IF (six or fewer cups wanted)
      2.1  Put water for cups required in coffee machine ;
      2.2  etc. etc.
    ENDIF
```

- Or this?

```
1.  Find out how many cups are required ;
2.  IF (more than six cups wanted)
      2.1  limit cups required to six ;
    ENDIF
3.  Put water for cups required in coffee machine ;
4.  etc. etc.
```

- What if nobody wants coffee? (zero cups). Don't want to measure zero coffee, zero water, wait for zero coffee to filter through, etc.

Complete solution?

```
1.  Find out how many cups are required ;
2.  IF (more than zero cups wanted)
      2.1.  IF (more than six cups wanted)
              2.1.1  limit cups required to six ;
            ENDIF
      2.2.  Put water for cups required in coffee machine ;
      2.3.  Open coffee holder ;
      2.4.  Put filter paper in machine ;
      2.5.  Measure coffee for number required ;
      2.6.  Put coffee into filter paper ;
      2.7.  Shut the coffee holder ;
      2.8.  Turn on machine ;
      2.9.  Wait for coffee to filter through ;
      2.10. WHILE (cups poured not equal to cups required)
              2.10.1.  Find out how many sugars required ;
              2.10.2.  Find out whether milk required ;
              2.10.3.  WHILE(sugars added not equal to required)
                         2.10.3.1.  Add spoonful of sugar ;
                         2.10.3.2.  Add 1 to number of sugars added ;
                       ENDWHILE
              2.10.4.  IF (white coffee required)
                         2.10.4.1.  Add milk/cream ;
                       ENDIF
              2.10.5.  Pour coffee into mug ;
              2.10.6.  Stir coffee ;
              2.10.7.  Add 1 to number of cups poured
            ENDWHILE
      2.11. Turn off machine ;
ENDIF
```

ACTIVITY

Find the variables. Write down their details in the table below. Give them meaningful identifiers (names) as this makes the solution much easier to read and understand. For each variable you identify give a short description that clearly explains what the variable represents. Also, indicate the range of values that the variable might typically represent. There are five variables to find. The first one is given:

Identifier (variable's name)	Description	Range
1. `coffeesRequired`	Holds the number of cups of coffee to be made	0–20
2.		
3.		
4.		
5.		

The variables?

- Use camel casing for the names

Identifier (variable's name)	Description	Range
1. `coffeesRequired`	Holds the number of cups of coffee to be made	0–20
2. `milkRequired`	Holds the milk preference for one drinker	{Yes, No}
3. `sugarRequired`	Holds the sugar requirements for one drinker	0, 1, 2, 3
4. `coffeesPoured`	Holds the number of cups poured so far	0–20
5. `sugarsAdded`	Holds the number of sugars added so far	0, 1, 2, 3

- {} denotes set of all possible values
- Other ranges are indicative

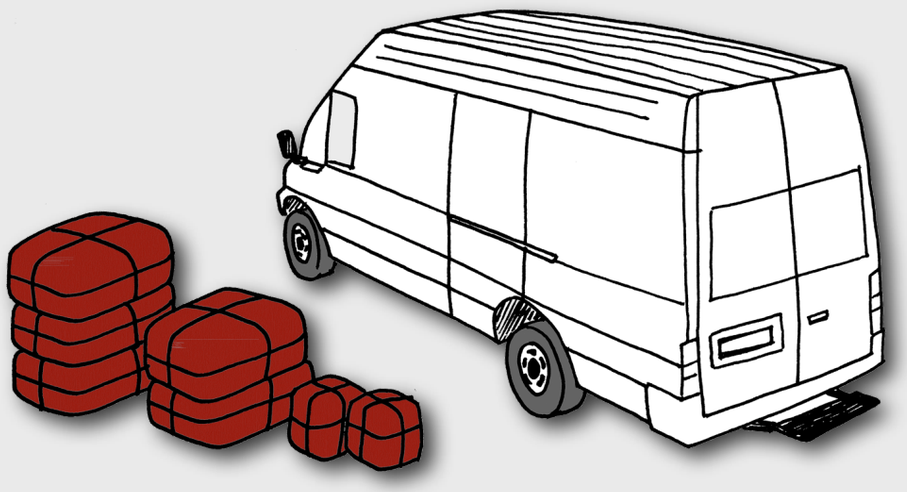
Solution with variables


```
1.  Find out how many cupsRequired ;
2.  IF (more than zero cupsRequired)
      2.1.  IF (more than six cups wanted)
              2.1.1.  limit cupsRequired to six ;
            ENDIF
      2.2.  Put water for cups required in coffee machine ;
      2.3.  Open coffee holder ;
      2.4.  Put filter paper in machine ;
      2.5.  Measure coffee for number required ;
      2.6.  Put coffee into filter paper ;
      2.7.  Shut the coffee holder ;
      2.8.  Turn on machine ;
      2.9.  Wait for coffee to filter through ;
      2.10. Initialize number of cupsPoured to zero ;
      2.11. WHILE (cupsPoured not equal to cupsRequired)
              2.11.1.  Initialize sugarsAdded to zero ;
              2.11.2.  Find out how many sugarsRequired ;
              2.11.3.  Find out whether milkRequired ;
              2.11.4.  WHILE (sugarsAdded not equal to sugarsRequired)
                         2.11.4.1.  Add spoonful of sugar ;
                         2.11.4.2.  Add 1 to number of sugarsAdded ;
                       ENDWHILE
              2.11.5.  IF (white coffee required)
                         2.11.5.1.  Add milk/cream ;
                       ENDIF
              2.11.6.  Pour coffee into mug ;
              2.11.7.  Stir coffee ;
              2.11.8.  Add 1 to number of cupsPoured
            ENDWHILE
      2.12. Turn off machine ;
    ENDIF
```

Arithmetic: loading vans

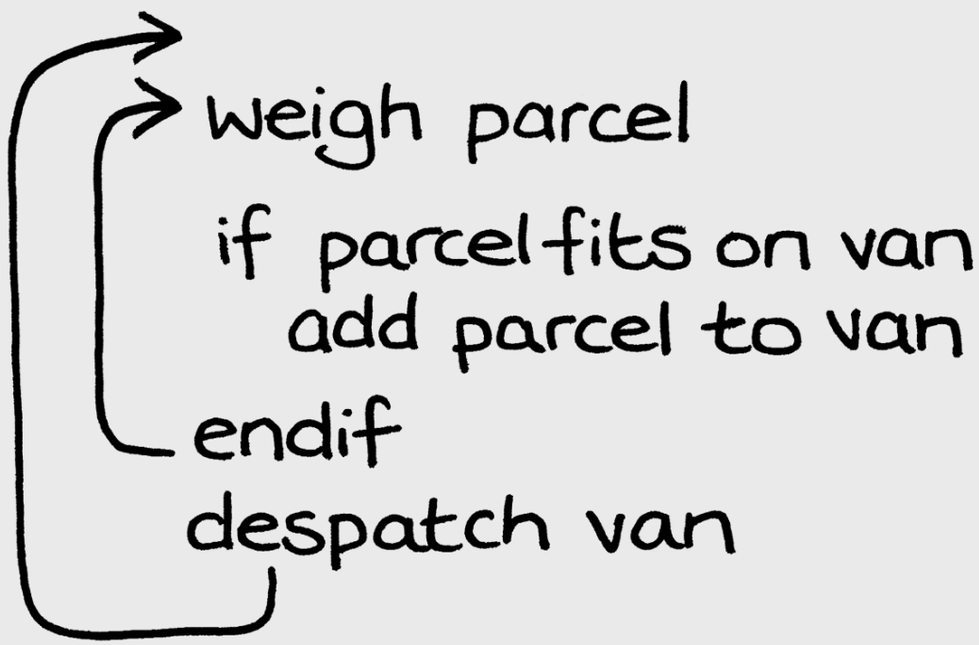
20 how to think like a programmer

Paul's Premier Parcels (PPP) is a delivery company. Each morning the parcels to be delivered for that day are lined up on a conveyor belt in the warehouse by the warehouse staff. The belt takes the parcels to the loading bay where a van waits. The loading crew weigh each parcel to see if it will fit on the van. If so, they put the parcel on the van and wait for the next parcel to arrive. As soon as a parcel arrives that would take the van over its maximum payload the van door is closed, the van sent off on its rounds and another empty van is brought up. The van is not kept waiting to see if any lighter parcels that will fit show up. This activity continues until all the parcels have been loaded onto vans. Happily, there are always enough vans for the waiting parcels. PPP wants to install a weighing machine on the conveyor belt linked to a computer so that the process of weighing parcels can be speeded up. Each van has the same maximum payload of 750 kg, and PPP only accepts parcels up to 120 kg in weight. After all the parcels have been loaded the company's managing director wants to know how many vans were needed that day and what was the heaviest payload that was sent out.



Principal parts

- The parcels, their weights, the vans, the maximum payload, a single van load of parcels.
- Sequence of actions for loading a single van:



- Here we see a selection nested within an iteration

Concept → Solution

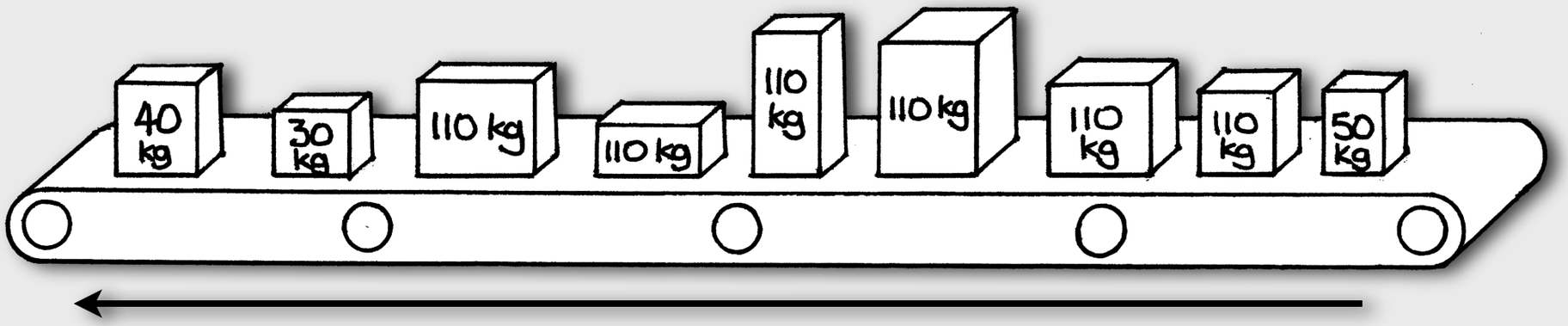
- Moving from high-level conceptual understanding to worked out pseudo-code solution can be tricky
- Some unanswered questions
  - Will likely need more than one van
  - How to tell if a parcel will fit on a van
  - Are the actions in the right order?
- Try visualizing the van loading process for real
  - Imagine being a warehouse loader

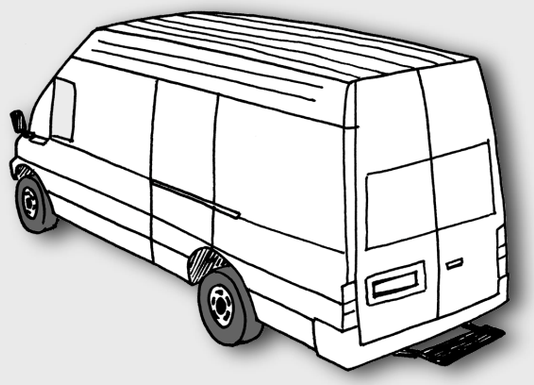
ACTIVITY

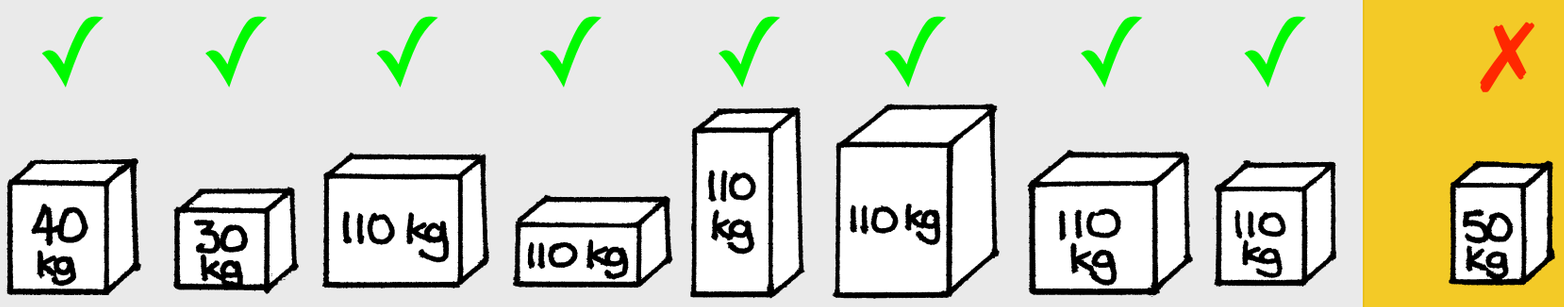
You have been tasked with the job of loading parcels on to vans. How do you do it? Walk through the process in your mind. Imagine yourself standing at the loading bay taking parcels off the conveyor belt. There is an empty van standing in front of you, and you have just weighed the first parcel. What do you do next, and why? When do you stop loading parcels and tell the van to drive off? When you have a clear picture of the process in your mind, write it down.

A visualization

- Imagine some parcels on a conveyor belt:







Load= 40 70 180 290 400 510 620 730 **780**

```
1.  Weigh first parcel ;
2.  WHILE (room on van)
      2.1.  Load parcel on van ;
      2.2.  Weigh next parcel ;
    ENDWHILE
3.  Despatch van ;
4.  Make note of van's payload ;
```

Enhancing the algorithm

- At the moment we have:

```
1.  Weigh first parcel ;
2.  WHILE (room on van)
      2.1.  Load parcel on van ;
      2.2.  Weigh next parcel ;
    ENDWHILE
3.  Despatch van ;
4.  Make note of van's payload ;
```

- How to express ‘room on van’?

```
1.  Weigh first parcel ;
2.  WHILE (payload + parcel weight less than or
           equal to capacity)
      2.1.  Load parcel on van ;
      2.2.  Weigh next parcel ;
    ENDWHILE
3.  Despatch van ;
4.  Make note of van's payload ;
```

Tracking the payload

- How do we know what the van's current payload is?
  - Keep a running total

```
1.  Initialize payload to zero ;
2.  Weigh first parcel ;
3.  WHILE (payload + parcel weight less than or
          equal to capacity)
      1.1.  Load parcel on van ;
      1.2.  Add parcel weight to payload ;
      1.3.  Weigh next parcel ;
    ENDWHILE
4.  Despatch van ;
```

ACTIVITY

Identify and write down the variables needed for Solution 5.13. Remember to write down their names, an explanation of their purpose, and an indicative range of possible values.

Identifier (variable's name)	Description	Range
1.		
2.		
3.		

The variables

- Here are the variables

Identifier (variable's name)	Description	Range
1. `parcelWeight`	Stores the weight of one parcel	{1 to 120}
2. `payload`	Stores the weight of the total load on the van	{0 to capacity}
3. `capacity`	Stores the maximum payload of a van	750

Solution with variables


```
1.  Initialize payload to zero ;
2.  Get first parcelWeight ;
3.  WHILE (payload + parcelWeight less than or
          equal to capacity)
      3.1.  Load parcel on van ;
      3.2.  Add parcelWeight to payload ;
      3.3.  Get next parcelWeight ;
    ENDWHILE
4.  Despatch van ;
```

- So that's one van loaded, what about multiple vans?
  Overall structure:

```
WHILE (parcels to be delivered)
   Load parcels on to vans,
   keeping count of the number and weight of vans used ;
ENDWHILE
Report on number of vans sent out ;
Report on heaviest payload ;
```

ACTIVITY

Write down an outline solution to the problem using the above structure and Solution 5.14. Remember, if you are finding it difficult to start, think about a similar problem you have already solved (Solution 5.9)

Loading multiple vans

- First attempt

```
1.  WHILE (conveyor not empty)
      1.1.  Initialize payload to zero ;
      1.2.  Get first parcelWeight ;
      1.3.  WHILE (payload + parcelWeight less than
                   or equal to capacity)
              1.3.1.  Load parcel on van ;
              1.3.2.  Add parcelWeight to payload ;
              1.3.3.  Get next parcelWeight ;
            ENDWHILE
      1.4.  Despatch van ;
    ENDWHILE
2. Report numberOfVans used ;
3. Report heaviestVan sent
```

- Not complete, but shows overall structure

Questions...

- How do `numberOfVans` and `heaviestVan` get their values?
  - Need some counting mechanisms
- The condition for the `WHILE` in Task #1.3: what happens when loading a van if the conveyor belt is empty?
- The position of Task #1.2 and Task #1.3.3. Task #1.3.3 gets a new parcel. If it will not fit on the van then the van is despatched and the `WHILE` in Task #1 is begun again which causes Task #1.2 to be obeyed again. What is the implication of this?

Finding heaviest van

- If we compare its weight with the heaviest value we wrote down last time we found a heavy van we can tell if this one is heavier

```
IF ( payload more than heaviestVan )
  Assign value of payload to heaviestVan ;
ENDIF
```

- To what value should `heaviestVan` be initialized?
  - 0 or 750?
- Fixing the loop

```
WHILE (payload + parcelWeight less than or equal
       to capacity) AND (conveyor not empty)
```

  - Note the AND -- makes a **compound** condition to stop when van is full or the conveyor becomes empty

Assigning values

- Programming languages have special operators for giving values to variables
- In pseudo-code we use an arrow

```
heaviestVan ← zero ;
```

- The value of expression on the right is assigned to variable on the left
- Value on the right **overwrites** the variable's current value
- To add to an existing value:

```
payload ← payload + parcelWeight ;
```

Updated solution


```
1.   capacity ← 750 ;
2.   numberOfVans ← zero ;
3.   heaviestVan ← zero ;
4.   Get first parcelWeight ;
5.   WHILE (conveyor not empty)
        5.1.  payload ← zero ;
        5.2.  WHILE (payload + parcelWeight less than or
                     equal to capacity) AND (conveyor NOT empty)
                 5.2.1.  Load parcel on van ;
                 5.2.2.  payload ← payload + parcelWeight ;
                 5.2.3.  Get next parcelWeight ;
              ENDWHILE
        5.3.  Despatch van ;
        5.4.  numberOfVans ← numberOfVans + 1 ;
        5.5.  IF ( payload more than heaviestVan )
                 5.5.1.  heaviestVan ← payload ;
              ENDIF
     ENDWHILE
6.   Report numberOfVans used ;
7.   Report heaviestVan sent ;
```

ACTIVITY

Simulate the loading process by following the algorithm in Solution 5.17. At the end of it you should be able to report the number of vans sent out and the weight of the heaviest payload. Use the following parcel weights for your run through

50, 90, 120, 110, 40,
30, 85, 85, 110, 100,
100, 100, 100, 120, 90,
50, 85, 120, 40

end of chapter 5