Methods of
Algorithm Description

Second Edition

to accompany the
2 Unit (General)
2/3 Unit (Common)
3 Unit (Additional)
Computing Studies Syllabuses
Republished as a second edition with permission of the Director-General, Department of School Education.

The original *Methods of Algorithm Description* document was developed at a Computer Education Unit, Department of School Education writing workshop.

© Board of Studies NSW 1995

Published by
Board of Studies NSW
PO Box 460
North Sydney NSW 2059
Australia

Tel: (02) 9927 8111
ISBN 0 7310 3365 5
March 1995 95010
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>5</td>
</tr>
<tr>
<td>Structure of the Document</td>
<td>6</td>
</tr>
<tr>
<td>What is Programming? (OHT original)</td>
<td>7</td>
</tr>
<tr>
<td>What is an Algorithm? (OHT original)</td>
<td>8</td>
</tr>
<tr>
<td>Overview of Two Methods</td>
<td>10</td>
</tr>
<tr>
<td><strong>Programming Structures</strong></td>
<td>12</td>
</tr>
<tr>
<td>- Sequence</td>
<td>13</td>
</tr>
<tr>
<td>- Selection</td>
<td>15</td>
</tr>
<tr>
<td>- Repetition</td>
<td>20</td>
</tr>
<tr>
<td>- Subprograms</td>
<td>24</td>
</tr>
<tr>
<td><strong>Solved Problems</strong></td>
<td>29</td>
</tr>
<tr>
<td>- Lift</td>
<td>31</td>
</tr>
<tr>
<td>- Solution 1</td>
<td>31</td>
</tr>
<tr>
<td>- Solution 2</td>
<td>32</td>
</tr>
<tr>
<td>- Temperature Control</td>
<td>33</td>
</tr>
<tr>
<td>- Toll Gate</td>
<td>35</td>
</tr>
<tr>
<td>- ‘Squash’ Scoring</td>
<td>38</td>
</tr>
<tr>
<td>- Record Separation</td>
<td>41</td>
</tr>
<tr>
<td>- Solution 1</td>
<td>42</td>
</tr>
<tr>
<td>- Solution 2</td>
<td>44</td>
</tr>
<tr>
<td>- Solution 3</td>
<td>46</td>
</tr>
<tr>
<td>- Guess the Number</td>
<td>48</td>
</tr>
<tr>
<td>- Income Tax</td>
<td>52</td>
</tr>
<tr>
<td>- Telephone Dialler</td>
<td>54</td>
</tr>
<tr>
<td>- Auto Teller</td>
<td>58</td>
</tr>
<tr>
<td><strong>Algorithms for Searching and Sorting</strong></td>
<td>71</td>
</tr>
<tr>
<td>- Linear Search</td>
<td>72</td>
</tr>
<tr>
<td>- Binary Search</td>
<td>75</td>
</tr>
<tr>
<td>- Bubble Sort</td>
<td>77</td>
</tr>
<tr>
<td>- Selection Sort</td>
<td>79</td>
</tr>
<tr>
<td>- Insertion Sort</td>
<td>82</td>
</tr>
<tr>
<td><strong>A Collection of Problems with No Solutions Given</strong></td>
<td>85</td>
</tr>
</tbody>
</table>
Introduction

This revised document was written in conjunction with the Board of Studies Information and Communication Technologies Syllabus Committee. It contains suitable methods of algorithm description for use in the implementation of the following courses in Computing Studies:

- 2 Unit (General) Computing Studies
- 2/3 Unit (Common) Computing Studies
- 3 Unit (Additional) Computing Studies.

This document presents two methods for describing algorithms for use in Computing Studies courses in NSW schools. It shows, through the use of examples, how the methods of algorithm description of pseudocode and flowcharts can be used to describe solutions to problems.

There are many definitions of methods of algorithm description in existence, some with many special symbols and keywords defined for special purposes. This document attempts to define a minimum set that will be useful in terms of the syllabuses in Computing Studies.

In assessing the quality of algorithm descriptions, general criteria such as the correctness of the algorithm, the clarity of the description, the use of appropriate control structures and the embodiment of structured methods, rather than the specific features of any method, should be taken into consideration.

Clarity of description and consistency in the use of the components of the method chosen are of far more importance than the actual shape of a flowchart element or the specific wording of a pseudocode statement.

The document presents standards that students should aim for in publishing solutions to problems. The same standards should be used by teachers when presenting algorithms to students. In many cases there are alternatives that could be used and it should be noted that students can expect to see methods of algorithm description with many differences in detail published in books and magazines. Teachers should ensure that the approach presented in textbooks, worksheets and examinations does not contradict the standards that students use.
Structure of the Document

The preliminary section consists of a description of the programming process and some definitions of an algorithm presented in the form of overhead transparency originals. In the following section are descriptions of the main features of two methods of algorithm description: pseudocode and flowcharts. Descriptions and specific examples of the programming structures of sequence, selection, repetition and subprograms (procedures or subroutines) are given.

The most substantial section of the document contains sample problems and worked solutions that show the use of each of the methods of algorithm description. The solutions do not purport to describe the ‘ultimate’ solution, but rather one (or more) possible solutions to the problem.

It will become obvious from reading the document that not all solutions are directly suited to implementation on a computer. The reason for this is that an aim of the document is to show how solutions to problems can be expressed through the use of sample problems to which most people already know some solution.

There is a new section in this edition which provides possible descriptions of the searching and sorting algorithms required by Core Topic 2: Algorithm Design in the 2/3 Unit (Common) Computing Studies course.

It must be noted that alternative solutions are possible. In some cases different solutions are provided expressed in each of the two prescribed methods. These can be used as models for expressing the same solution in the other method and should extend the reader’s understanding of the topic.

The final section has a collection of problems for which no solutions are given. These are provided to give you some practice using the methods of algorithm description.
What is Programming?

Programming is the total creative process that involves these stages:

- clearly define the problem
- analyse the problem
- design a solution
- implement the solution
- test the solution
- document the solution.

In the appropriate circumstances we should also:

- compare alternative solutions.
What is an Algorithm?

An algorithm consists of a set of explicit and unambiguous finite steps which, when carried out for a given set of initial conditions, produce the corresponding output and terminate in finite time.

*How to Solve it by Computer*, RG Dromey, Prentice Hall UK, 1982

An algorithm is a finite, definite, effective procedure, with some output.

*Computer Science*, D Woodhouse et al, Jacaranda Wiley, 1984

The series of steps that you develop to solve a problem is known as a solution algorithm. There are many different algorithms for almost any problem.

*Understanding Information Technology*, K Behan and D Holmes, Prentice Hall Australia, 1986
Reprinted with the permission of Prentice Hall Australia Pty Ltd.
Algorithm: a step-by-step procedure for solving a problem; programming languages are essentially a way of expressing algorithms.


In order that a task be carried out on a computer, a method or technique for the task must be described very precisely in terms of the different steps. An algorithm is a description of the steps of a task, using a particular technique. Writing an algorithm is one of the first steps taken in preparing a task to be done by a computer.

*Computing Science*, Peter Bishop, Thomas Nelson UK, 1982

Informally, an algorithm is a collection of instructions which, when performed in a specific sequence, produce the correct result. The study of algorithms is at the heart of computer science.

Overview of Two Methods

Pseudocode

Pseudocode essentially is English with some defined rules of structure and some keywords that make it appear a bit like program code. Some guidelines for writing pseudocode are as follows.

Pseudocode Guidelines

- The keywords used for pseudocode in this document are:

  for start and finish
  BEGIN MAINPROGRAM, END MAINPROGRAM

  for initialisation
  INITIALISATION, END INITIALISATION

  for subprogram
  BEGIN SUBPROGRAM, END SUBPROGRAM

  for selection
  IF, THEN, ELSE, ENDIF

  for multi-way selection
  CASEWHERE, OTHERWISE, ENDCASE

  for pre-test repetition
  WHILE, ENDOF

  for post-test repetition
  REPEAT, UNTIL

- Keywords are written in capitals.

- Structural elements come in pairs, eg for every BEGIN there is an END, for every IF there is an ENDIF, etc.

- Indenting is used to show structure in the algorithm.

- The names of subprograms are underlined. This means that when refining the solution to a problem, a word in an algorithm can be underlined and a subprogram developed. This feature is to assist the use of the ‘top-down’ development concept.
Flowcharts

Flowcharts are a diagrammatic method of representing algorithms. They use an intuitive scheme of showing operations in boxes connected by lines and arrows that graphically show the flow of control in an algorithm. The Australian Standards for flowcharting indicate that the main direction of flow is accepted as being top to bottom and left to right.

Flowchart Elements

Flowcharts are made up of the following box types connected by lines with arrowheads indicating the flow. It is common practice only to show arrowheads where the flow is counter to that stated above.

These should be thought of as the characters of flowcharts. Just as ordinary characters must be put together in certain ways to produce well-formed words, and words must be put together in certain ways to produce well-structured sentences, these flowchart elements must be connected in certain ways to form accepted structures and the structures connected in certain ways to form well-structured algorithms. The flowcharting structures for sequence, selection and repetition are given in the next section of this document.

It is considered good practice for a single flowchart never to exceed the bounds of one page. If a flowchart does not fit on one page, this is one instance in which the better solution is to use refinement which results in the creation of subprograms. Subprograms on separate pages are more desirable than using a connector to join flowcharts over more than one page. A flowchart expressing the solution to an involved problem may have the main program flowchart on one page with subprograms continuing the problem solution on subsequent pages. An example of this situation is given in the last solved problem in this document — the Auto Teller problem on page 54.
The Computing Studies syllabuses mention the programming structures of *sequence*, *selection*, *repetition* and *subprograms*. A description of each of these structures, together with examples of their use, follows.

**The Structures**

Each of the five acceptable structures can be built from the basic elements as shown below.
In all cases note there is only one entry point to the structure and one exit point as indicated by the dashed boxes.

Since each structure can be thought of as a process (as shown by the dashed boxes containing the structure), more complex algorithms can be constructed by replacing any single process by one or other of the structures.

**Sequence**

In a computer program or an algorithm, sequence involves simple steps which are to be executed one after the other. The steps are executed in the same order in which they are written.

In pseudocode, sequence is expressed as:

```
process 1
process 2
...
...
process n
```

In a flowchart, sequence is expressed as:

(The arrowheads are optional if the flow is top-to-bottom.)
An Example Using Sequence

**Problem:** Write a set of instructions that describe how to make a pot of tea.

**Pseudocode**

BEGIN
fill a kettle with water
boil the water in the kettle
put the tea leaves in the pot
pour boiling water in the pot
END

**Flowchart**
Selection

Selection is used in a computer program or algorithm to determine which particular step or set of steps is to be executed. A selection statement can be used to choose a specific path dependent on a condition. There are two types of selection: binary (two-way branching) selection and multi-way (many way branching) selection. Following is a description of each.

Binary Selection

As the name implies, binary selection allows the choice between two possible paths. If the condition is met then one path is taken, otherwise the second possible path is followed. In each of the examples below, the first case described requires a process to be completed only if the condition is true. The process is ignored if the condition is false. In other words there is only one path that requires processing to be done, so the processing free path is left out rather than included saying ‘do nothing’.

In pseudocode, binary selection is expressed in the following ways:

1. IF condition THEN
   process 1
   ENDIF
2. IF condition THEN
   process 1
   ELSE
   process 2
   ENDIF

In flowcharts, binary selection is expressed in the following ways:

1.

![Flowchart of Binary Selection](image)
Note: In a flowchart it is most important to indicate which path is to be followed when the condition is true, and which path to follow when the condition is false. Without these indications the flowchart is open to more than one interpretation.

Note: There are two acceptable ways to represent a decision in all of the structures.

1. The condition is expressed as a statement and the two possible outcomes are indicated by True, False.

2. The condition is expressed as a question and the two possible outcomes are indicated by Yes, No.

Either method is acceptable. For consistency, the former method is used throughout the document.
Multi-way Selection

Multi-way selection allows for any number of possible choices, or cases. The path taken is determined by the selection of the choice which is true. Multi-way selection is often referred to as a case structure.

In pseudocode, multiple selection is expressed as:

```
CASEWHERE expression evaluates to
  choice a : process a
  choice b : process b
  ...
  OTHERWISE : default process
ENDCASE
```

Note: As the flowchart version of the multi-way selection indicates, only one process on each pass is executed as a result of the implementation of the multi-way selection.

In a flowchart, multi-way selection is expressed as:
Examples Using Binary Selection

**Problem 1:** Write a set of instructions to describe when to answer the phone.

**Pseudocode**

IF the telephone is ringing THEN
  answer the telephone
ENDIF

**Flowchart**

![Flowchart for Problem 1](image)

**Problem 2:** Write a set of instructions to follow when approaching a set of traffic control lights.

**Pseudocode**

IF the signal is green THEN
  proceed through the intersection
ELSE
  stop the vehicle
ENDIF

**Flowchart**

![Flowchart for Problem 2](image)
An Example Using Multi-way Selection

**Problem:** Write a set of instructions that describes how to respond to all possible signals at a set of traffic control lights.

**Pseudocode**

CASEWHERE signal is
   red : stop the vehicle
   amber : stop the vehicle
   green : proceed through the intersection
   OTHERWISE : proceed with caution
ENDCASE

**Flowchart**
Repetition allows for a portion of an algorithm or computer program to be done any number of times dependent on some condition being met. An occurrence of repetition is usually known as a loop.

An essential feature of repetition is that each loop has a termination condition to stop the repetition, or the obvious outcome is that the loop never completes execution (an infinite loop). The termination condition can be checked or tested at the beginning or end of the loop, and is known as a pre-test or post-test respectively. Following is a description of each of these types of loop.

Repetition: Pre-Test

A pre-tested loop is so named because the condition has to be met at the very beginning of the loop or the body of the loop is not executed. This construct is often called a guarded loop. The body of the loop is executed repeatedly while the termination condition is true.

In pseudocode pre-test repetition is expressed as:

```
WHILE condition is true
  process(es)
ENDWHILE
```

In flowcharting pre-test repetition is expressed as:

Although these two flowcharts are topologically the same and represent the same programming structure, the left-hand example above is used throughout this document.
Repetition: Post-Test

A post-tested loop executes the body of the loop before testing the termination condition. This construct is often referred to as an *unguarded loop*. The body of the loop is repeatedly executed until the termination condition is true.

An important difference between a pre-test and post-test loop is that the statements of a post-test loop are executed at least once even if the condition is originally true, whereas the body of the pre-test loop may never be executed if the termination condition is originally true. A close look at the representations of the two loop types makes this point apparent.

In pseudocode, post-test is expressed as:

```
REPEAT
  process
UNTIL condition is true
```

In a flowchart, post-test is expressed as:
An Example Using Pre-Test Repetition

Problem: Determine a safety procedure for travelling in a carriage on a moving train.

Pseudocode

WHILE the train is moving
    keep wholly within the carriage
ENDWHILE

Flowchart
An Example Using Post-Test Repetition

**Problem:** Determine a procedure to beat egg whites until fluffy.

**Pseudocode**

```plaintext
REPEAT
  beat the egg whites
UNTIL fluffy
```

**Flowchart**
Subprograms

Subprograms, as the name implies, are complete part-programs that are used from within the main program section. They allow the process of refinement to be used to develop solutions to problems that are easy to follow. Sections of the solution are developed and presented in understandable chunks, and because of this, subprograms are particularly useful when using the top-down method of solution development.

When using subprograms it is important that the solution expression indicates where the main program branches to a subprogram. It is equally important to indicate exactly where the subprogram begins. In pseudocode, the statement in the main program that is expanded in a subprogram is underlined to indicate that further explanation follows. The expanded subprogram section should be identified by using the keywords BEGIN SUBPROGRAM followed by the underlined title used in the main program. The end of the subprogram is marked by the keywords END SUBPROGRAM and the underlined title used in the main program.

When using flowcharts, a subprogram is shown by an additional vertical line on each side of the process box. This indicates that the subprogram is expanded elsewhere. The start and end of the subprogram flowchart uses the name of the subprogram in the termination boxes.

Example of Using Subprograms in Pseudocode

BEGIN MAINPROGRAM
process 1
process 2
process 3
process 4
END MAINPROGRAM

BEGIN SUBPROGRAM process 2
  do this
  do that
END SUBPROGRAM process 2
Example of Using Subprograms in Flowcharts

begin
  process 1
  process 2
  process 3
  process 4
end

begin process 2
  do this
  do that
end process 2
In many cases a subprogram can be written to do the same task at two or more points in an algorithm. Each time the subprogram is called, it may operate on different data. To indicate the data to be used one or more parameters are used. The parameters allow the author to write a general algorithm using the formal parameters. When the subprogram is executed, the algorithm carries out its task on the actual parameters given at the call.

The parameters to be used by a subprogram are provided as a list in parentheses after the name of the subprogram. There is no need to include them at the end of the algorithm.

**Example of Using Subprograms with one Parameter in Pseudocode**

BEGIN MAINPROGRAM
read (name)
read (address)
END MAINPROGRAM

BEGIN SUBPROGRAM read (array)
set pointer to first position
get a character
WHILE character is not the end of data AND there is room in the array
store character in the array at the position given by the pointer
increment the pointer
get a character
ENDWHILE
END SUBPROGRAM read

At the first call of this subprogram the characters are read into the array called ‘name’, at the second call the characters are read into the array called ‘address’.
Example of Using Subprograms with one Parameter in Flowcharts

begin

read (name)

read (address)

end

begin read (array)

set pointer to first position

get a character

character ≠ end of data and room in the array

store character in array at position given by the pointer

increment pointer

get a character

end read
Solved Problems
## Methods of Algorithm Description

### Table Indicating Programming Structures Used in the Sample Problems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Sequence</th>
<th>Binary Selection (if-then)</th>
<th>Multiple Selection (case)</th>
<th>Repetition: (while)</th>
<th>Repetition: (repeat-until)</th>
<th>Subprogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lift Solution 1</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lift Solution 2</td>
<td>✓</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature Control</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Toll Gate</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Squash Scoring</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record Separation Solution 1</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record Separation Solution 2</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Record Separation Solution 3</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guess the Number</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Income Tax</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Telephone Dialler</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Auto-Teller</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Lift Problem

Problem
A lift remains positioned at the ground floor level of a building with the doors shut whenever it is not in use. When a call button is pressed on any floor, the lift moves to the required floor and the lift doors open. Write an algorithm to express the logic of controlling the lift.

Solution 1
This solution uses sequence and repeat-until structures.

Pseudocode
An algorithm to express the logic of controlling a lift
BEGIN MAINPROGRAM
REPEAT
    check all buttons
UNTIL a button is pressed
    move to the required floor
    open the doors
END MAINPROGRAM

Flowchart
An algorithm to express the logic of controlling a lift
Solution 2

This solution uses *sequence* and *while* structures.

**Pseudocode**

An algorithm to express the logic of controlling a lift

```plaintext
BEGIN MAINPROGRAM
  check all buttons
  WHILE no button has been pressed
    check all buttons
  ENDWHILE
  move to the required floor
  open the doors
END MAINPROGRAM
```

**Flowchart**

An algorithm to express the logic of controlling a lift

![Flowchart diagram]

1. begin
2. check all buttons
3. no button is pressed
   - False
   - check all buttons
4. move to the required floor
5. open the doors
6. end
Temperature Control Problem

Problem

At a NSW coastal town the maximum annual temperature range is typically 12–34 degrees Celsius. An air conditioning company is installing a heating/cooling system in a new shopping centre in that town. The system checks the temperature every five minutes and adjusts the air temperature by using a combination of two heating and two cooling units. These units operate according to these temperature ranges:

- 0–15 degrees C – 2 heating units
- 16–20 degrees C – 1 heating unit
- 21–28 degrees C – 1 cooling unit
- > 29 degrees C – 2 cooling units

Write an algorithm that could be used to control the air conditioning system.

Solution

This solution uses sequence, while and case structures.

Pseudocode

An algorithm to describe the control of an air conditioning system. The input comes from sensors in the shopping centre.

BEGIN MAINPROGRAM
read the temperature
WHILE the system is turned on
    CASEWHERE temperature
        < 16 : run two heating units
        16 to 20 : run one heating unit
        21 to 28 : run one cooling unit
        OTHERWISE : run two cooling units
    ENDCASE
    wait five minutes
    read the temperature
ENDWHILE
END MAINPROGRAM
Flowchart

An algorithm to describe the control of an air conditioning system. The input comes from sensors in the shopping centre.

begin
read the temperature

system turned on
False
True

temperature range is

less than 16
16–20
21–28
otherwise

run two heating units
run one heating unit
run one cooling unit
run two cooling units

wait five minutes

read the temperature

end
Toll Gate Problem

Problem
When operational a toll gate operates by having a boom gate obstructing the road, and a sensor detecting when a vehicle is present. After coins to the value of $1.00 have been deposited in the basket, the boom gate opens and stays open until a vehicle has gone through. Amounts greater than $1.00 are accepted but no change is given. Individual coins less than 10 cents are ignored.

Write an algorithm to describe the control of the toll gate.

Solution
This solution uses sequence, if-then, repeat-until, while, and subprogram structures.

Pseudocode
An algorithm used to describe the operation of a toll gate that has a boom gate, a vehicle sensor, and a coin collection basket.

BEGIN MAINPROGRAM
REPEAT
   REPEAT
      wait
   UNTIL car has arrived
   get the money
   open boom gate
   repeat
      wait
   UNTIL car has passed
   close boom gate
UNTIL toll gate is not operational
END MAINPROGRAM

BEGIN SUBPROGRAM get the money
INITIALISATION
money collected is set to 0
END INITIALISATION
WHILE money collected is less than $1
   receive coin
   IF coin is less than 10 cents THEN
      ignore coin
   ELSE
      add the value of the coin to the money collected
   ENDIF
ENDWHILE
END SUBPROGRAM get the money
Flowchart

An algorithm used to describe the operation of a toll gate that has a boom gate, a vehicle sensor, and a coin collection basket.

```
begin
  wait
  False car has arrived
  True get the money
  open boom gate
  wait
  False car has passed
  True close boom gate
  False toll gate not operational
  True end
```
Subprogram

begin
get the money

set money collected to zero

money collected less than $1

receive a coin

coin smaller than 10c

add value of coin to money collected

ignore coin

end
get the money
‘Squash’ Scoring Problem

Problem
Write an algorithm to describe how to score a ball game, which is similar to squash. This ball game is scored as follows: the server gets one point for winning a rally. If the server loses the rally they lose the right to serve the next ball, but lose no points. The receiver gains the right to serve (but no point) if they win a rally. To win the game a player must win nine points.

Solution
This solution uses sequence, repeat-until, if-then and subprogram structures.

Pseudocode
An algorithm to describe the logic for scoring a ball game similar to squash.

BEGIN MAINPROGRAM
INITIALISATION
set RequiredPoints to 9
set each player’s points to 0
END INITIALISATION
toss and decide the server
REPEAT
    server serves the ball
    REPEAT
        play the rally
        UNTIL rally is won
        IF the server wins the rally THEN
            increment the server’s points by 1
        ELSE
            swap player status
        ENDIF
    UNTIL a player has won RequiredPoints
declare the winner
END MAINPROGRAM

BEGIN SUBPROGRAM toss and decide the server
toss a coin
    IF heads THEN
        player 1 is the server
        player 2 is the receiver
    ELSE
        player 2 is the server
        player 1 is the receiver
    ENDIF
END SUBPROGRAM toss and decide the server
An algorithm to describe the logic for scoring a ball game similar to squash.

begin
set RequiredPoints to 9
set each player's score to 0
toss and decide server
server serves the ball
play a rally
rally won
server wins the rally
increase server's points by one
swap the player status
player has RequiredPoints
end
begin toss and decide server

begin toss and decide server

begin toss and decide server

begin toss and decide server
Record Separation Problem

Problem

Let us assume that a particular database program manages a simple mailing list which consists of one record for each person on the list, and a number of fields containing information about each person (their name, address, etc). The program can read in data produced by a word processor provided that data is structured in the following way:

Each record to be read must be a single paragraph terminated by a return character, and each field within a record is separated by a tab character. For example:

Colin Jameson tab 33 Falcon Street tab Waverly tab NSW tab 2113 return

would be read as one record containing five fields. The end of the data is marked with a # (hash) character which immediately follows the return ending the last paragraph.

Assuming that there is at least one line of valid data at the start of the input file, describe an algorithm that the program might use to read such data one character at a time and place the information into separate fields and records. The algorithm reports the number of records read when all the records have been processed.
Solution 1

This solution uses sequence, while and case structures.

Pseudocode

An algorithm to describe the separation of a string of formatted data into fields and records to be used as input to a database.

BEGIN MAINPROGRAM

INITIALISATION
set record number to 0
set field number to 0
set field to empty
END INITIALISATION

read a character from the input file
WHILE character is not a hash

CASEWHERE character is

  tab: output the field to the database
       increment the field number
       set the field to empty

  return: output the field to the database
          increment the field number
          increment the record number
          set field number to 0
          set the field to empty

  OTHERWISE: append the character to the field

ENDCASE

read a character from the input file
ENDWHILE

report how many records were read

END MAINPROGRAM
Flowchart

An algorithm to describe the separation of a string of formatted data into fields and records to be used as input to a database.

begin
record number and field number are set to 0
field is set to empty
read a character from input file

character is

True

character is not a hash

False

increment the field number
set field to empty
output the field
increment the record number
set field number to 0 and field to empty
read a character from input file

tab

return

otherwise

append character to the field

increment the field number
output the field

report how many records were read

end
Solution 2

This solution uses sequence, repeat-until and if-then-else structures.

Pseudocode

An algorithm to describe the separation of a string of formatted data into fields and records to be used as input to a database.

BEGIN MAINPROGRAM

INITIALISATION
record number is set to 0
field number is set to 0
field is set to empty
END INITIALISATION

REPEAT
  read a character from the file
  IF the character is a hash THEN
    don’t do anything
  ELSE
    IF the character is a return THEN
      output the field to the database
      increment the field number
      increment the record number
      set the field number to 0
      set the field to empty
    ELSE
      IF the character is a tab THEN
        output the field to the database
        increment the field number
        set the field to empty
      ELSE
        append the character to the field
      ENDIF
    ELSE
      append the character to the field
    ENDIF
  ENDIF
UNTIL the character is a hash
report how many records were read
END MAINPROGRAM
Flowchart

An algorithm to describe the separation of a string of formatted data into fields and records to be used as input to a database.
Solution 3

This solution uses *sequence* and *while* structures.

**Pseudocode**

An algorithm to describe the separation of a string of formatted data into fields and records, which are to be used as input to a database. It assumes the data are correct.

```plaintext
MAINPROGRAM
INITIALISATION
set record number to 0
set field number to 0
set field to empty
END INITIALISATION
read a character from the file
WHILE the character is not a hash
    WHILE the character is not a return
        WHILE the character is not a tab
            append the character to the field
        read a character from the file
    ENDWHILE
    output the field to the database
    increment the field number
    set field to empty
    read a character from the file
ENDWHILE
output the field to the database
increment the record number
set field number to 0
set the field to empty
read a character from the file
ENDWHILE
report how many records were read
END MAINPROGRAM
```
Flowchart
An algorithm to describe the separation of a string of formatted data into fields and records, which are to be used as input to a database. It assumes the data are correct.
Guess the Number Problem

Problem

In a simple number game your opponent thinks of a secret number between 1 and 100. In no more than 10 guesses you have to try to guess the number. After each guess your opponent tells you if your guess was too high, too low or correct. Your opponent also keeps track of how many guesses you have had and tells you the game is over when you use all of your ten guesses or when you guess the number correctly.

Describe an algorithm which takes the role of your opponent in this game. Include in your solution a subprogram which checks for illegal guesses (those less than 1 or greater than 100). Include also the subprogram which generates a secret number between 1 and 100. Do not expand this but assume it is available.
Solution

This solution uses sequence, repeat-until, if-then-else and subprogram structures.

Pseudocode

An algorithm to describe a game in which the user tries to guess a number between 1 and 100, using no more than ten guesses.

BEGIN MAINPROGRAM
INITIALISATION
number of guesses is set to 0
GotIt is set to false
END INITIALISATION
generate a secret number using random number generator
REPEAT
   get a guess from the user
   IF the guess is in range THEN
      increment the number of guesses
      check the guess
   ELSE
      tell the user the guess is out of range
   ENDIF
UNTIL guess is correct (GotIt is true) or number of guesses is 10
IF the guess is incorrect (GotIt is false) THEN
   tell the user they have run out of guesses (=10)
   tell the user the secret number
ENDIF
END MAINPROGRAM

BEGIN SUBPROGRAM check the guess
IF guess > secret number THEN
   tell the user their guess is too big
ELSE
   IF guess < secret number THEN
      tell the user their guess is too small
   ELSE
      congratulate the user on a correct guess
      tell them how many guesses they took
      set GotIt to true
   ENDIF
ENDIF
END SUBPROGRAM check the guess
An algorithm to describe a game in which the user tries to guess a number between 1 and 100, using no more than ten guesses.

begin
set the number of guesses to 0 and GotIt to false

generate a secret number using random generator

get a guess from the user

guess in range

True

increment the number of guesses

check the guess

False

tell user that guess is out of range

GotIt is false or number of guesses is equal to 10

True

tell the user the secret number

GotIt is false

False

tell user they have run out of guesses

end
Subprogram

begin
check the guess

guess is bigger than secret number

True

tell the user the guess is too big

False

guess is less than secret number

True

tell the user the guess is too small

False

congratulate the user on guessing the secret number

tell the user how many guesses were taken

set GotIt to true

end
check the guess
Income Tax Problem

Problem
To calculate the income tax payable on any income based on the income tax scales shown below. The taxable income is to be entered and the tax payable calculated.

Tax Scales

<table>
<thead>
<tr>
<th>Taxable Income ($)</th>
<th>Tax payable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1–5400</td>
<td>Nil</td>
</tr>
<tr>
<td>$5401–20 700</td>
<td>Nil plus 20 cents for each $1 over $5400</td>
</tr>
<tr>
<td>$20 701–36 000</td>
<td>$3060 plus 38 cents for each $1 over $20 700</td>
</tr>
<tr>
<td>$36 001–50 000</td>
<td>$8874 plus 46 cents for each $1 over $36 000</td>
</tr>
<tr>
<td>$50 001 and over</td>
<td>$15 314 plus 47 cents for each $1 over $50 000</td>
</tr>
</tbody>
</table>

Solution
This solution uses sequence and if-then structures.

Pseudocode
An algorithm used to calculate the tax payable on any income using taxation rates set in the given table.

BEGIN MAINPROGRAM
  input income
  IF income greater than or equal to 50 001 THEN
    tax is 15 314 + (income – 50 000) * 0.47
  ELSE
    IF income greater than or equal to 36 001 THEN
      tax is 8874 + (income – 36 000) * 0.46
    ELSE
      IF income greater than or equal to 20 701 THEN
        tax is 3060 + (income – 20 700) * 0.38
      ELSE
        IF income greater than or equal to 5401 THEN
          tax is (income – 5400) * 0.20
        ELSE
          tax is nil
        ENDIF
      ENDIF
    ENDIF
  ENDIF
  ENDIF
  display income and tax payable
END MAINPROGRAM
Flowchart

An algorithm used to calculate the tax payable on any income using taxation rates set in the given table.

Another valid method of solving this problem is to use the multiple selection or ‘case’ structure.
Problem

A telephone dialler is connected between a computer and a telephone (see the diagram below). Its purpose is to dial a telephone number entered via the computer keyboard, establish a connection if it can and report on its progress and degree of success. The whole telephone number is entered via the computer keyboard at one time and is stored in a buffer in the computer.

The dialler ‘dials’ a digit by sending pulses along the telephone line. To dial 2 it sends two pulses, to dial 5 it sends five pulses etc. In the special case of a zero it sends ten pulses. There is a gap of 2 seconds between the set of pulses representing a digit of a telephone number.

The dialler will not operate unless the line is clear, in which case it will provide a message that it allows a dial tone then dials the number. Before sending the next digit it will check for a response, this takes into account the different lengths of phone numbers. The dialler will provide a message that the phone is ‘ringing’, ‘engaged’ or ‘answered’.

Write an algorithm to describe the operation of the telephone dialler.
Solution

This solution used sequence, repeat-until, if-then, while, case and subprogram structures.

Pseudocode

An algorithm to describe the control of a telephone dialler.

BEGIN MAINPROGRAM
REPEAT
  try for phone line
UNTIL the response is a dial tone
send a message to the computer that a clear telephone line is available
REPEAT
  REPEAT
    get a character from the computer
UNTIL the character is a digit
  IF the character is a 0 THEN
    set the digit value of the character to 10
  ENDIF
  assign the digit value of the character to a counter
  WHILE counter is greater than 0
    send a pulse
decrement the counter
  ENDWHILE
send no pulse for two seconds
UNTIL there is a response
determine outcome and send message (response)
END MAINPROGRAM

BEGIN SUBPROGRAM determine outcome and send message (response)
INITIALISATION
set maxtime to 60
END INITIALISATION
IF response is an engaged tone THEN
  send a message that the phone is engaged
ELSE
  IF response indicates the phone is ringing THEN
    send a message that the phone is ringing
    set a timer to 0
    REPEAT
      check to see if the phone has been answered
      increment the timer
    UNTIL the phone is answered OR timer is greater than maxtime
    CASEWHERE the phone was
      answered : send a message that a connection has been established
      unanswered : send a message that no connection has been established
    OTHERWISE : send an error message
    ENDCASE
  ELSE
    send an error message
  ENDIF
ENDIF
END SUBPROGRAM determine outcome and send message
An algorithm to describe the control of a telephone dialler.
Subprogram

begin determine outcome and send message (response)

set maxtime to 60

True

Response is engaged

False

send a message that the phone is engaged

send an error message

False

phone is ringing

True

send a message that the phone is ringing

set a timer to 0

check to see if the phone is answered

increment timer

False

phone is answered or timer > maxtime

True

phone was

False

answered

unanswered

otherwise

send a message that connection established

send a message that no connection established

send an error message

end determine outcome and send message
Problem

An automatic teller machine has a console as shown in the diagram below. The teller machine follows this sequence to assist a customer:

1. The customer will insert their card and enter a PIN. A customer is allowed at most three tries at their PIN. If they get it wrong three times the whole process ends without ejecting the card.

2. If the PIN is correct they will then select an action button (withdraw, deposit or balance).

3. Next they will select the account type (savings or cheque).

4. Finally, they will enter the amount in whole dollars (if appropriate), and press OK to confirm it, and the transaction will be processed.

The auto teller will eject the customer’s card and stop the process if the customer presses the ‘Cancel’ button.

Describe an algorithm which the auto teller could use to accept the details from the customer and act on them.
Solution

This solution uses sequence, if-then, repeat-until, while, case and subprogram structures.

Pseudocode

An algorithm to describe the control of an automatic teller machine. Input comes from the buttons on the console, output through a small video screen.

BEGIN MAINPROGRAM
INITIALISATION
set Action to an empty string
set Account to an empty string
set Amount to 0
END INITIALISATION
wait for the card to be inserted
get the PIN and check it (Action)
IF action is 'cancel' THEN
eject card
ELSE
  IF action is not 'keep card' THEN
    get action required (Action)
    IF action is not 'cancel' THEN
      get account to be used (Action, Account)
      do the transaction
      ENDIF
    ENDIF
    eject card
  ENDIF
ENDIF
ENDIF
END MAINPROGRAM

BEGIN SUBPROGRAM wait for the card to be inserted
WHILE no card has been inserted
  wait
ENDWHILE
END SUBPROGRAM wait for the card to be inserted
BEGIN SUBPROGRAM get the PIN and check it (Action)

INITIALISATION
set OK to FALSE
set NumberOfTries to 3

END INITIALISATION

IF cancel button has been pressed THEN
set Action to ‘cancel’
ELSE
REPEAT
accept a four digit number
decrement NumberOfTries
IF correct PIN THEN
set OK to TRUE
ENDIF
UNTIL OK OR number of tries is 0
IF NOT OK THEN
set Action to ‘keep card’
ENDIF
ENDIF

END SUBPROGRAM get the PIN and check it

BEGIN SUBPROGRAM get action required (Action)

IF cancel button has been pressed THEN
set Action to ‘cancel’
ELSE
REPEAT
prompt user for action key
get a key press
UNTIL key press is an action key
CASE
WHERE keypress is
   Withdrawal : set Action to ‘withdraw’
   Deposit     : set Action to ‘deposit’
   Balance     : set Action to ‘show balance’
   Cancel      : set Action to ‘cancel’
ENDCASE
ENDIF

END SUBPROGRAM get action required
BEGIN SUBPROGRAM get account to be used (Action, Account)
IF cancel button is pressed THEN
    set Action to ‘cancel’
ELSE
    REPEAT
        prompt for account type key
        get a keypress
    UNTIL keypress is acceptable
    CASEWHERE keypress is
        savings account : set Account to ‘savings account’
        cheque account : set Account to ‘cheque account’
        cancel : set Action to ‘cancel’
    ENDCASE
ENDIF
END SUBPROGRAM get account to be used

BEGIN SUBPROGRAM get the amount in dollars (Action, Amount)
INITIALISATION
set OK to FALSE
END INITIALISATION
IF the cancel button has been pressed THEN
    set Action to ‘cancel’
ELSE
    REPEAT
        REPEAT
            prompt for a keypress
            get a keypress
        UNTIL keypress is acceptable
        CASEWHERE keypress is
            OK : set OK to TRUE
            cancel : set Action to ‘cancel’
            set OK to TRUE
            number : set Amount to 10 times the amount
            plus digit value of number
        ENDCASE
    UNTIL OK
ENDIF
END SUBPROGRAM get the amount in dollars
BEGIN SUBPROGRAM do the transaction (Action, Account)

IF the cancel button has been pressed THEN
    set Action to 'cancel'
ENDIF

CASEWHERE Action is
    cancel: set Amount to 0
            set Account to empty string
    deposit: get the amount in dollars (Action, Amount)
             IF Action is not 'cancel' THEN
                  IF Account is 'cheque' THEN
                      add Amount to cheque account
                  ELSE
                      add Amount to savings account
                  ENDIF
             ENDIF
    withdraw: get the amount in dollars (Action, Amount)
              IF Action is not 'cancel' THEN
                  IF Account is 'cheque' THEN
                      subtract Amount from cheque account
                  ELSE
                      subtract Amount from savings account
                  ENDIF
              ENDIF
    balance: IF Account is 'cheque' THEN
                 display balance of cheque account
            ELSE
                 display balance of savings account
            ENDIF
ENDCASE

END SUBPROGRAM do the transaction
An algorithm to describe the control of an automatic teller machine. Input comes from the buttons on the console, output through a small video screen.

```
begin
set Action to an empty string
set Account to an empty string
set Amount to 0
wait for card to be inserted
get the PIN and check it

Action is 'cancel'

get Action required (Action)

Action is not 'cancel'

get Account to be used (Action, Account)
do the transaction (Action, Account)

end
```

**Flowchart**

An algorithm to describe the control of an automatic teller machine. Input comes from the buttons on the console, output through a small video screen.
Methods of Algorithm Description

Subprograms

begin wait for card to be inserted
   card not inserted
      False
      True
      wait
   end wait for card to be inserted
begin get the PIN & check it (Action)

set OK to false

set number of tries to 3

cancel button is pressed

accept a four digit number

decrement number of tries

correct PIN

OK = true or number of tries = 0

set OK to true

set Action to 'keep card'

set Action to 'cancel'

end get the PIN and check it
Methods of Algorithm Description

begin get action required (Action)

cancel button is pressed

set Action to 'cancel'

True

set Action to ‘cancel’

False

prompt for action key

get a keypress

keypress is action key

False

keypress is

True

end get Action required

Withdrawal
Deposit
Balance
Cancel

set Action to ‘withdrawal’
set Action to ‘deposit’
set Action to ‘show balance’
set Action to ‘cancel’
begin get account to be used (Action, Account)

cancel button is pressed

prompt for account type key

get a keypress

keypress is acceptable

keypress is

set Account to 'cheque account'
set Account to 'savings account'
set Action to 'cancel'

end get account to be used
begin get amount in dollars (Action, Amount)

set OK to false

cancel button is pressed

set Action to 'cancel'

keypress is acceptable

prompt for a keypress

get a keypress

keypress is acceptable

keypress is

OK

set OK to true

set Action to 'cancel'

set OK to true

OK is true

True

end get amount in dollars

set Amount to 10 times the amount plus digit value of number
begin do transaction (Action, Account)

False
cancel button pressed
True

Action is

cancel
set Amount to 0
set Account to empty string

deposit
get Amount in dollars (Action, Amount)
False
Action is not 'cancel'
True
False
Account is 'cheque'
add Amount to savings account
add Amount to cheque account

withdrawal
get Amount in dollars (Action, Amount)
False
Action is not 'cancel'
True
False
Account is 'cheque'
subtract Amount from savings account
subtract Amount from cheque account

balance
False
Account is 'cheque'
True

display balance of savings account
display balance of cheque account

set Action to 'cancel'

end do transaction
Algorithms for Searching and Sorting
To give some assistance with understanding the standard algorithms for searching and sorting, as required in the 2/3 Unit (Common) Computing Studies Syllabus, the following examples are provided.

**Linear Search**

**Problem**

In a theatre there is a row of seats. Each seat is numbered consecutively starting at 1. A person occupies each of the seats. Describe an algorithm which an usher could follow to find the seat number of the first person in the row who is wearing a red jumper (indicated by the dark circle).
Solution

This solution implements a linear (sequential) search which will work whether or not the persons are seated in a given order or at random.

The solution uses sequence, while and if-then-else structures.

Pseudocode

An algorithm to describe a linear (sequential) search to find the seat number of the first person wearing a red jumper from a set of persons sitting in a row of seats.

BEGIN MAINPROGRAM
  INITIALISATION
  stand in front of the first seat
  set FoundIt to FALSE
  set MoreSeats to TRUE
  END INITIALISATION

  get the description of the wanted person

  WHILE FoundIt is FALSE AND MoreSeats
    IF the person in front of you is not the wanted person THEN
      stand in front of the next seat
    ELSE
      set FoundIt to TRUE
    ENDIF
  ENDDO WHILE

  IF FoundIt THEN
    report the seat number of the wanted person
  ELSE
    report that the wanted person is not present
  ENDIF

END MAINPROGRAM
Problem

A user has stored a set of numbers in a one-dimensional array. Each element of the array contains a unique number. Describe an algorithm the person could follow to find the element of the array which contains a particular number.

```
1  2  3  4  5  6  7  8
14 34 12 19 41 26 45 16
```

Solution

This solution implements a linear (sequential) search which will work whether or not the numbers are stored in order or at random.

The solution uses `sequence`, `while`, `if-then` and `if-then-else` structures.
Binary Search

The task is to determine whether or not a particular data value (the target) is present in a set of data. If the target is found the position of its first occurrence is reported and the search ends.

Note: that binary search will only work on sorted data.

General Idea of the Algorithm

Binary search divides the data set into two parts and determines in which part the element is likely to be found. The other part of the data set is discarded and the retained part is divided into two parts. The process is continued until either the value is found or there are no more elements in the data set to be checked. If a match is found then the position of the match is reported otherwise a message is written telling the user that the target is not present in the data.

At each division there are three possibilities for the target (if it exists in the data set):

(1) the target lies at the division point;
(2) the target lies to the left of the division point
(3) the target lies to the right of the division point.
Problem

A user wants to find the telephone number of a person in the Sydney Telephone Directory. The names are listed alphabetically. Each set of data for a person is stored as an element of a one-dimensional array. One way would be to carry out a linear search starting with the first name in the listing and checking each one until either the target is found or the end of the directory is reached. Describe a more efficient algorithm which the person could use to find the telephone number of the particular person.

Solution

This solution implements a binary search which will work only if the names are in strict alphabetical order.

The solution uses sequence, repeat-until, if-then-else and subprogram structures.

Pseudocode

BEGIN MAINPROGRAM

INITIALISATION
set Lower to first position
set Upper to last position
set FoundIt to FALSE
get the Target name
END INITIALISATION

REPEAT

calculate the Middle position

IF Target = data at Middle position THEN
set FoundIt to TRUE
set PositionFound to Middle
ELSE
IF Target < data at Middle position THEN
set Upper to Middle – 1
ELSE
set Lower to Middle + 2
ENDIF
ENDIF

UNTIL FoundIt OR Lower > Upper

IF FoundIt THEN
report PositionFound
ELSE
report ‘Target not present’
ENDIF

END MAINPROGRAM

BEGIN SUBPROGRAM calculate the Middle position

set Middle to (Upper + Lower) divided by 2
set Middle to integer part of Middle

END SUBPROGRAM calculate the Middle position
Algorithms for Sorting

Bubble Sort

The task is to sort a set of data into either ascending order or descending order as determined when the algorithm is written. In this case the order chosen is ascending order.

General Idea of the Algorithm

The data elements are compared in pairs and the larger of the pair ‘bubbles’ towards the top of the structure. On each pass, one element (the largest in the unsorted part) will be moved to its correct position in the sorted part.

```
1 2 3 4 5 6
42 32 23 12 19 54  # Compare first with second and Swap
1 2 3 4 5 6
32 42 23 12 19 54  # Compare second with third and Swap
1 2 3 4 5 6
32 23 42 12 19 54  # Compare third with fourth and Swap
1 2 3 4 5 6
32 23 12 42 19 54  # Compare fourth with fifth and Swap
1 2 3 4 5 6
32 23 12 19 42 54  # Compare fifth with sixth and leave
1 2 3 4 5 6
32 23 12 19 42 54  # Data set at end of first pass
```

Problem

A person wants to sort a set of marks into ascending order. Each mark is stored as an element of a one-dimensional array. Describe an algorithm which will sort the marks by ‘bubbling’ the largest mark in the unsorted part to the ‘top’ of the unsorted part. This will result in the sorted part getting larger each time through and the unsorted part getting smaller. The algorithm need not stop even if the person realises that the marks become sorted before the algorithm has finished.
Solution

This solution implements a bubble sort on an array of marks.

The solution uses sequence, while, if-then and subprogram structures.

Pseudocode

BEGIN MAINPROGRAM

INITIALISATION
set End to last position
END INITIALISATION

WHILE End > first position
  set Current to first position
  WHILE Current is less than End
    IF data at Current > data at (Current + 1) THEN
      Swap (Current, Current + 1)
    ENDIF
    increment Current
  ENDWHILE
  decrement End
ENDWHILE

END MAINPROGRAM

Note: that the parameters to the subprogram when it is called by the MAINPROGRAM are Current and Current + 1 but in the definition of the SUBPROGRAM general names are used for the parameters. When the SUBPROGRAM is called it substitutes the names of the actual parameters for those of the formal parameters used in the definition.

BEGIN SUBPROGRAM Swap (Position1, Position2)
  set Temp to data value at Position1
  set data at Position1 to data at Position2
  set data at Position2 to Temp
END SUBPROGRAM Swap
Selection Sort

In this algorithm a set of data is sorted into either ascending order or descending order as determined when the algorithm is written. In this case the order chosen is ascending order.

General Idea of the Algorithm

The general idea behind the algorithm is to divide the array into two parts — the unsorted part and the sorted part. Each pass through the unsorted part finds the largest number and places it at the start of the sorted part. The array originally has an ‘empty’ sorted part and a ‘full’ unsorted part. So the largest number in the unsorted part is found and swapped with the last element in the unsorted part. The length of the sorted part is then increased by one and the length of the unsorted part is decreased by one.

<table>
<thead>
<tr>
<th>Unsorted</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>22</td>
<td>32</td>
</tr>
<tr>
<td>41</td>
<td>24</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>32</td>
<td>22</td>
</tr>
<tr>
<td>41</td>
<td>24</td>
</tr>
<tr>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>24</td>
</tr>
<tr>
<td>32</td>
<td>22</td>
</tr>
<tr>
<td>41</td>
<td>24</td>
</tr>
<tr>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>32</td>
<td>22</td>
</tr>
<tr>
<td>41</td>
<td>24</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>22</td>
</tr>
<tr>
<td>24</td>
<td>32</td>
</tr>
<tr>
<td>41</td>
<td>24</td>
</tr>
<tr>
<td>16</td>
<td>22</td>
</tr>
<tr>
<td>24</td>
<td>32</td>
</tr>
<tr>
<td>41</td>
<td>24</td>
</tr>
<tr>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>22</td>
<td>24</td>
</tr>
<tr>
<td>32</td>
<td>41</td>
</tr>
</tbody>
</table>

Methods of Algorithm Description
Pseudocode

BEGIN MAINPROGRAM
    INITIALISATION
    set EndUnsorted to last position
    END INITIALISATION
    WHILE EndUnsorted > first position
        set Current to first position
        set Largest to data at Current
        set PositionOfLargest to Current
        WHILE Current < EndUnsorted
            increment Current
            IF data at Current > Largest THEN
                set Largest to data at Current
                set PositionOfLargest to Current
            ENDIF
        ENDWHILE
        Swap(PositionOfLargest, EndUnsorted)
        decrement EndUnsorted
    ENDWHILE
    END MAINPROGRAM

BEGIN SUBPROGRAM Swap(Position1, Position2)
    set Temp to data value at Position1
    set data at Position1 to data at Position2
    set data at Position2 to Temp
    END SUBPROGRAM Swap
set EndUnsorted to last position

set Current to first position

set Largest to data at Current

set Position Of Largest to Current

increment Current

begin Swap
set Temp to data at Position1
set data at Position1 to data at Position2
set data at Position2 to Temp
end Swap

set Largest to data at Current
set PosOfLargest to Current

swap (Position Of Largest, EndUnsorted)

decrement EndUnsorted

end
Insertion Sort

In this algorithm a set of data is sorted into either ascending order or descending order as determined when the algorithm is written. In this case the order chosen is ascending order.

General Idea of the Algorithm

This is the method used by many card players to put their cards in order. The general idea of the algorithm is to divide the array into two parts — the unsorted part and the sorted part. To begin with, the sorted part contains only the right hand element. Each pass takes the last element from the unsorted part and then finds where it should be inserted in the sorted part. To find the proper place to insert the element a sequential (linear) search is used. As each element in the sorted part is checked, it is moved, if necessary, one place to the left to make room for the new element. At each pass the length of the sorted part increases by one and the length of the unsorted part decreases by one until its length is 0.
**Pseudocode**

BEGIN MAINPROGRAM

INITIALISATION
set First to first position
set Last to last position
set PositionOfNext to Last – 1
ENDINITIALISATION

WHILE PositionOfNext >= First
  set Next to data at PositionOfNext
  set Current to PositionOfNext
  WHILE (Current < Last ) AND (Next > data at (Current + 1))
    increment Current
    set data at (Current – 1) to data at Current
  ENDWHILE
  set data at Current to Next
  decrement PositionOfNext
ENDWHILE

END MAINPROGRAM

**Note:** Variations of the algorithm are possible. In some versions, the task of finding the correct place to insert is separated from the task of moving the elements to make room. By doing this, the task of finding the correct place to insert can be speeded up by using a **binary search** rather than a **linear search**.
Flowchart

begin

set First to first position

set Last to last position

set PositionOfNext to Last –1

Flowchart

PositionOfNext >= First

set Next to data at PositionOfNext

set Current to PositionOfNext

Flowchart

Current < Last AND Next > data at Current + 1

increment Current

set data at Current –1 to data at Current

Flowchart

set data at Current to Next

Flowchart

decrement PositionOfNext

end

Methods of Algorithm Description

84
A COLLECTION OF PROBLEMS WITH NO SOLUTIONS GIVEN
Use these problems to test your skills at using methods of algorithm description.

1. Refine the ‘Lift Problem’ (on page 27) to establish a more ‘realistic’ problem and solution. Consider the problem of sending the lift to specific floors. Determine when the lift doors should be opened and closed.

2. Refine the ‘Telephone Dialler Problem’ to cater for these suggested extensions.
   (a) Modify the dialler so that it rejects numbers beginning with 0 (for STD call bar).
   (b) As for (a), but allow access to Austpac 01922, 01923, 01924.
   (c) Restrict as in (a), but allow 008 numbers.

3. A one-lane bridge only allows traffic to travel in one direction at a time. A set of traffic lights controls the flow of traffic. It takes a slow vehicle 45 seconds at most to cross the bridge. Write an algorithm to specify the control of the traffic lights.

   Extension:
   Consider a ‘rush’ period in which traffic travelling north is three times as heavy as traffic travelling south.

4. Road work is being undertaken on a country road and traffic is being controlled by a player of mechanical ‘stop/slow’ paddle turners. Write an algorithm to describe the control of the paddle turners to safely direct traffic around the road works.

5. A program accepts as input dates in the form dd/mm/yy (eg 13/11/85) and later uses them in expanded form (eg 13 November 1985). Write an algorithm that could be used to do this conversion. Assume that there are no incorrect dates, and that all dates are in the 20th century.

6. Write an algorithm that will take today’s date and someone’s birth date as input and use the data to calculate the person’s age in years and full months.
7. A community group has decided to install an automatic sprinkler system in the local park. The sprinkler should come on when the ground moisture reduces to 20% of saturation level which is measured by a sensor. It should be turned off when the moisture reaches 55% of saturation level. Write an algorithm that could be used to control the sprinkler system.

8. A tourist bus has 53 seats. When tickets are booked, the next available seats are allocated according to number (ignoring where they are). Write an algorithm to decide what is the next seat available and to store the name of the person booking each seat.

Extension 1
The seats are arranged so that numbers 1 and 2 are together, 3 and 4 are across the aisle and so on up to 48. Seats 49 to 53 are across the back of the bus. Write an algorithm that can choose the seat wanted from what is available, then store the name of the person booked with the seat number.

Extension 2
Write an algorithm to print out the names of people booked on the bus in the order of the seats.

9. A subprogram of a grammar checking program checks for the use of apostrophes for the possessive case. Search a line of text for the symbol ‘. If it follows an s and is also followed by an s, remove the second s.