

BOARD OF STUDIES
NEW SOUTH WALES

Software Design and Development

Stage 6

Software and Course Specifications

Higher School Certificate
2001

The Board of Studies has made all reasonable attempts to locate owners of third party copyright materials and invites anyone from whom permission has not been sought to contact the Copyright Officer, Board of Studies NSW. Ph: (02) 9367 8111; fax: (02) 9279 1482.

© Board of Studies NSW 1999

Published by
Board of Studies NSW
GPO Box 5300
Sydney NSW 2001
Australia

Tel: (02) 9367 8111

Internet: <http://www.boardofstudies.nsw.edu.au>

ISBN 0 7313 4347 6

99403

Contents

| | |
|---|----|
| Foreword | 5 |
| Introduction | 6 |
| General Specifications | 6 |
| Representational Tools..... | 6 |
| Software Tools | 10 |
| Metalanguages | 11 |
| Software Specifications | 13 |
| Language Specifications..... | 13 |
| Options..... | 16 |
| Methods of Algorithm Description | 17 |
| What is an algorithm?..... | 18 |
| Overview of two methods..... | 19 |
| Programming structures..... | 21 |

Foreword

The HSC software and course specifications for Software Design and Development contain information pertaining to the Higher School Certificate from 2001. This information is relevant to students studying Preliminary courses from 2000. Any amendments to requirements will be notified in the *Board Bulletin* Official Notices.

These HSC software and course specifications should be read in conjunction with:

- *Software Design and Development Stage 6 Syllabus* and support documents
- Official Notices in the *Board Bulletin*
- Examination, Assessment and Reporting Supplement
- examination and assessment reports.

The Board of Studies reserves the right to make changes to the software and course specifications. As they are reviewed, the amendments will be published on the Board of Studies website <http://www.boardofstudies.nsw.edu.au> and notified in the Official Notices in the *Board Bulletin*.

Curriculum advice may be obtained on:

Phone (02) 9367 8246 fax (02) 9367 8476

Board of Studies publications (syllabuses, support documents, *Board Bulletins*, specimen examination papers) may be obtained from Client Services on:

Phone (02) 9367 8495 fax (02) 9262 8178
(fax orders preferred)

Correspondence should be addressed to:

Board of Studies
GPO Box 5300
Sydney NSW 2001

Introduction

This document has been produced to provide clarification of the depth of treatment required for some concepts in the *Software Design and Development Stage 6 Syllabus*. Essential and desirable software features to be used are also identified. The document should be read in conjunction with the *Software Design and Development Stage 6 Syllabus*.

In addition to the software and concepts detailed in this document, students should be exposed to further software and concepts that illustrate syllabus content.

To fulfil syllabus requirements, students must have access to software that has all of the essential features. Examination questions can be set which require an acquaintance with such software. Particular features of specific software will not be examined.

The document is available on the Board's website so that it can be regularly updated.

General Specifications

Representational Tools

Context diagrams

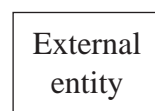
Context diagrams are used to represent entire information systems. The system is shown as a single process along with the inputs and outputs (external entities) to the system. The external entities are connected to the single process by data flow arrows. The symbols used are:



a single process representing the entire system as a circle



data flow representing the flow of data between the single process and external entities



any person or organisation that provides data to the system or receives data from the system.

Data dictionary

A comprehensive description of each field in the database. This commonly includes: field name, number of characters (field width), data type, number of decimal places (if applicable) and a description of the purpose of each field.

Data flow diagram

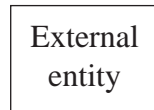
A data flow diagram provides more detail at a lower level for a context diagram. Data flow diagrams represent the information system as a number of processes that together form the single process of a context diagram. The source of data, its flow between processes and its destination along with data generated by the system is represented. The symbols used are:



a process represented by a circle. Processes are an action taking place transforming inputs to outputs



data flow, representing the flow of data between the single process and external entities



any person or organisation that provides data to the system or receives data from the system



a location where data is stored. It can be in computer format, such as a diskette, or in non-computer format, such as a filing cabinet or an answering machine.

IPO chart

These charts are used to document the inputs into a process, the general nature of the processes performed on this input, and outputs produced for each task or process in a system.

When used in conjunction with a structure chart, one IPO chart should be produced for each lowest level process on the structure chart.

The IPO chart is in the form of a table with 3 columns, one for each of Input, Process and Output.

Validation of student information:

| Input | Process | Output |
|---|--------------------------|--------------------------------|
| Student id (read from magnetic card) | Read student record | Valid-student-flag |
| Password (read from keyboard) | Compare entered password | |
| | Check if correct | 0 if correct 1 if incorrect |
| | | Student id |

The same information can be expressed in a different format, as follows:

| | |
|---------------|---|
| System | School Library System |
| Process Name: | Validate student information |
| Input: | Student id (read from magnetic card) |
| | Password (read from keyboard) |
| Process: | Read student record |
| | Compare entered password |
| | Check if correct |
| Output: | Valid Student Flag (0 if valid, 1 if invalid) |

Storyboard

Storyboards are used to document the screens used in a system, and the flow between them. Screens should contain sufficient detail to indicate the general layout of text, fields, graphics and buttons included. The flow between screens is indicated by the use of arrows. Where a button click causes the movement to the next screen, the arrow should start from the appropriate button.

Structure Chart

Structure diagrams allow the representation of a system broken down into its separate subtasks or processes. The relationship between each of these processes should also be evident from the diagram.

Rectangles are used to represent tasks, with lines used to show the connections between tasks. The chart is read from top to bottom, with component subtasks on successively lower levels, and from left to right to show the order of execution of tasks at the same level.

The symbols used are:



Data movement between tasks (usually passed as parameters) is shown with the use of arrows.



A filled circle is used to indicate a flag.

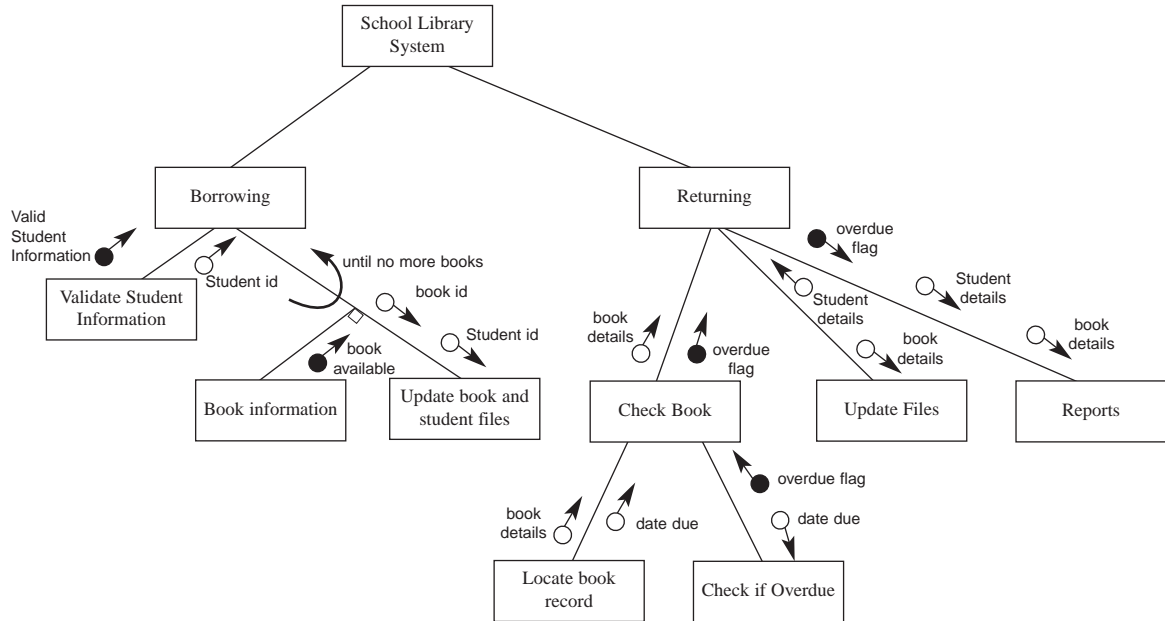


A decision (ie optional execution of a task) is indicated by use of a small diamond at the intersection of the line.



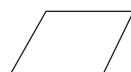
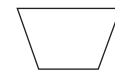





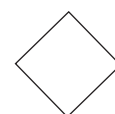

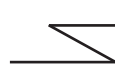

Repetition (execution of a particular task multiple times) is shown by a circular arrow.

The following example illustrates these features:



System flowcharts

System flowcharts are a diagrammatic way of representing both the flow of data and logic through an information system. They were once the primary tool for documenting systems; however, data flow diagrams are often used instead. Standard flowcharting symbols are used along with symbols for representing physical devices that capture, store and display data. Many of these symbols have become out of date as a result of changes in technology.

| | | | |
|---|----------------|--|------------------------------|
|  | Input/output |  | Manual operation |
|  | Paper document |  | Magnetic tape |
|  | Online display |  | Direct access storage device |
|  | Online input |  | Decision |
|  | Punched card |  | Telecommunications link |
|  | Process | | |

Software Tools

CASE tools

The software must allow the production and maintenance of:

- Manuals, incorporating screen shots, table of contents, index
- Flowcharts
- Gantt charts
- Data dictionary
- System flowcharts
- Structure charts, data flow diagrams, and other appropriate modelling tools.

Recommended software:

Word processing, spreadsheet and drawing packages with appropriate graphics items.
CASE Tools such as Ascent, together-J, Rational Rose.

Meta Languages

BNF

Abbreviation for **Backus-Naur form**.

BNF is a metalanguage used to define the syntax of a programming language. It uses the following symbols:

:: = 'is defined as'

| (or) indicates a choice between alternatives

non-terminal symbol a symbol still to be defined

< > used to enclose non-terminal symbols

terminal symbol is used as written.

EBNF

Abbreviation for **extended Backus-Naur form**.

In this extended form the following symbols are used:

= 'is defined as'

| 'or' indicates a choice between alternatives

terminal symbol is used as written (may be a symbol enclosed in quotation marks or a reserved word written in upper case)

[] indicate an optional part of a definition

{ } indicate a possible repetition (0 or more times)

() used to group elements together.

Example:

Identifier = <Letter> {<Letter> | <Digit>}

Interpretation:

An identifier is defined to be a Letter followed by one or more Letters or Digits.

Letter is a non-terminal symbol and is defined elsewhere, and *Digit* is another non-terminal symbol defined elsewhere.

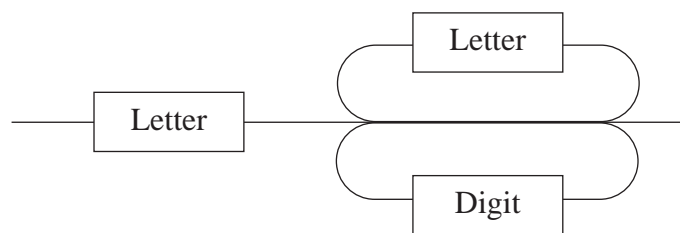
Railroad diagram

This is an alternative, graphical method used to define the syntax of a programming language.

Rectangles are used to enclose non-terminal symbols (that is, symbols that will be further defined). Circles or rounded rectangles are used to enclose terminal symbols. These elements are linked by paths to show all legal combinations. By starting at the left-hand side of the diagram and tracing any path in a forward direction to reach the right-hand side of the diagram, a syntactically correct construct will be covered.

(‘Railroad’ in this context means a branch in the diagram is legal if it is treated as a set of points in a railroad layout and a train can take the branch in a forward direction only).

Example:



Software Specifications

Language Specifications

The syllabus does not prescribe a single coding language for implementation of programs but advocates a range of high level languages.

Students are required to be proficient in using at least one of the currently approved languages but will not be asked to interpret or produce code as part of the external assessment.

The languages chosen for inclusion in the approved list are ones which are generally available and support structured programming concepts.

General language requirements

The programming language chosen **must** allow the students to:

- use numeric (integer and real), character and string data;
- use record and array data types including multi-dimensional arrays and arrays of records;
- use simple variables of type CHARACTER, REAL, INTEGER and STRING;
- use meaningful identifiers;
- use parentheses in creating expressions
- use the arithmetic operators of ADDITION, SUBTRACTION, MULTIPLICATION and DIVISION;
- use string handling operations to extract characters from a string
- use the logical operators of AND, OR and NOT;
- use relational operators which provide the equivalents of the following functions:

EQUAL TO
GREATER THAN
LESS THAN

NOT EQUAL TO
GREATER THAN OR EQUAL TO
LESS THAN OR EQUAL TO;

- use and create procedures (subprograms, subroutines) which may require parameters;
- input data from the keyboard and/or a data file;
- write output to the screen and/or a data file;
- use selection statements equivalent to:

(i) IF <condition TRUE > THEN

 <statement sequence 1>

ELSE

 <statement sequence 2>

ENDIF

(ii) CASE <control expression> OF

<case value list>

ENDCASE

- use repetition statements equivalent to:

(iii) REPEAT

<statement sequence>

UNTIL <condition TRUE>

(iv) WHILE <condition TRUE> DO

<statement sequence>

ENDWHILE

(v) FOR <control variable> taking <initial value> TO <final value> BY steps of 1 DO

<statement sequence>

ENDFOR

- include COMMENTS (REMARKS) in the code to document the program;
- include the use of a debugging facility such as single instruction stepping, trace and breakpoints;
- allow the reading and writing of sequential and random files
- experience the use of both a compiler and an interpreter.

Appropriate Languages:

- Pascal, a structured version of BASIC.

Event-driven Languages

The programming language(s) chosen **must** allow the students to:

- create scripts which respond to button presses and mouse actions, and make use of standard control structures
- should not be limited to simple linking of screens.

Appropriate Languages:

- Visual Basic, Hypercard, Delphi (limited functions only), REALBasic.

Prototyping and Rapid Applications Development

The software package(s) chosen **must** allow the students to:

- create several linked pages, cards or screens;
- create graphic and text elements on the pages, cards or screens;
- accept input from the keyboard;
- store and access data;
- perform mathematical operations on the data;
- perform sorting, searching and reporting operations on the data;
- use a scripting language which permits system events such as mouse button presses and keystrokes to be handled.

Appropriate languages:

- Visual Basic, Hypercard, Delphi (limited functions only), Access, Filemaker-Pro, REALBasic.

Software used to simulate CPU processing of instructions

The software package(s) chosen should allow the students to:

- simulate the processing of machine code instructions.

Appropriate languages:

- TIM, 'Under the Hood' including a CPU simulation.

Options

1. Evolution of Programming Languages

Students should design and create a simple expert system using an expert system shell, as an example of the logic paradigm.

The software must allow students to:

- enter simple IF-THEN rules
- add, remove and edit rules
- query the expert system
- display the rules that the system used to reach a conclusion.

Appropriate software:

Eshell, ESIE, Clixpert

2. The Software Developer's view of the hardware

The software package(s) chosen must allow students to:

- Drag and drop logic gate symbols to create a circuit
- Edit existing designs
- Print circuit designs
- Simulate the working of designed circuits to show outputs for selected input values.

Appropriate software:

LogicSim, LogicCircuits, Crocodile Clips

Methods of Algorithm Description

Introduction

There are many definitions of methods of algorithm description in existence, some with many special symbols and keywords defined for special purposes.

This document presents two methods for describing algorithms for use in the implementation of the Software Design and Development course.

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.

What is an Algorithm?

Algorithm: a step-by-step procedure for solving a problem; programming languages are essentially a way of expressing algorithms.

Understanding Computers: Computer Languages,
by the editors of Time-Life Books, © 1988
Time-Life Books Inc.

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.

Problem Solving and Computer Programming,
Peter Grogono & Sharon H Nelson,
© 1982 Addison-Wesley Publishing Company Inc.
Reproduced by the permission of the publisher.

Overview of Two Methods

It is expected that students are able to develop and interpret algorithms using both of these 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, ENDWHILE

for post-test repetition

REPEAT, UNTIL

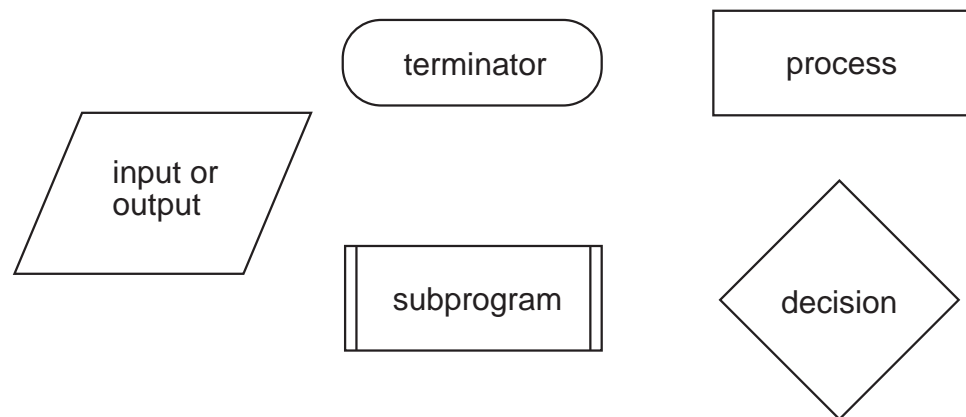
- Keywords are written in capitals.
- Structural elements come in pairs, eg for every **BEGIN** there is an **END**, for every **IF** there is an **ENDIF**, etc.
- Indenting is used to show structure in the algorithm.
- The names of subprograms are underlined. This means that when refining the solution to a problem, a word in an algorithm can be underlined and a subprogram developed. This feature enables the use of the ‘top-down’ development concept, where details for a particular process need only be considered within the relevant sub-routine.

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.

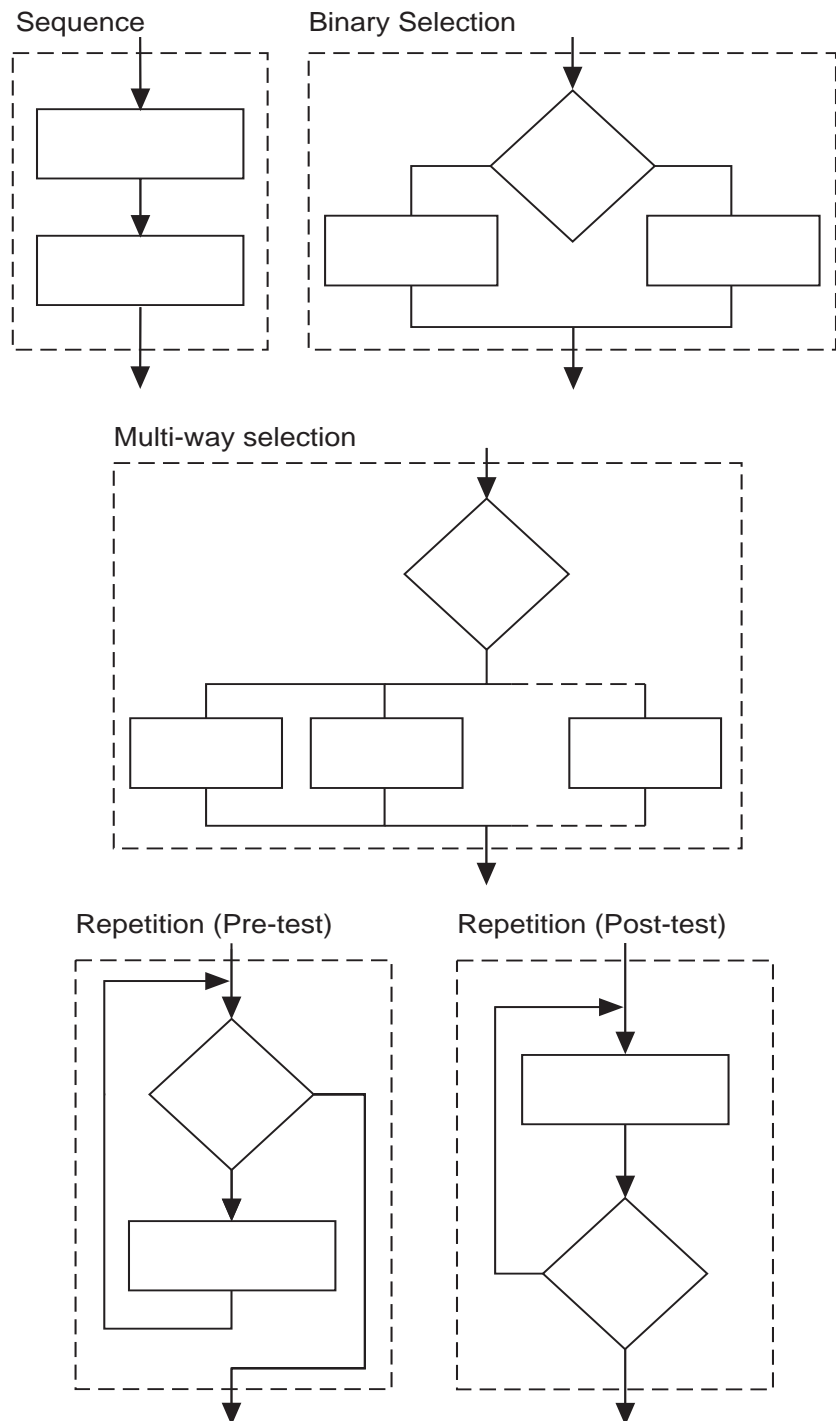
It is considered good practice for a single flowchart never to exceed the bounds of one page. If a flowchart does not fit on one page, this is one instance in which the better solution is to use refinement which results in the creation of subprograms. Subprograms on separate pages are more desirable 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. Regardless of page size, it is also important to start any complex algorithm with a clear, uncluttered main line. This should reference the required subroutines, whose detail is shown in separate flowcharts.

Programming Structures

The Software Design and Development syllabus mentions 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.



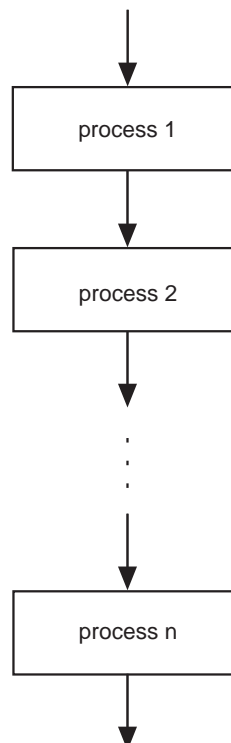
In pseudocode, sequence is expressed as:

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.

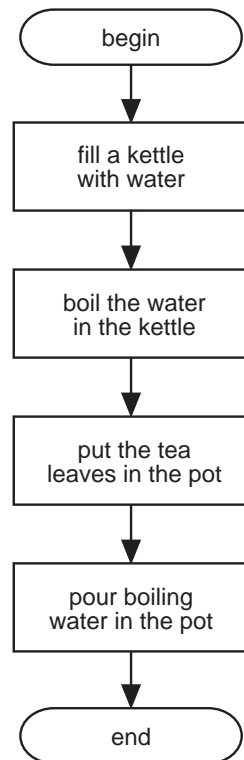


Pseudocode

```
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:

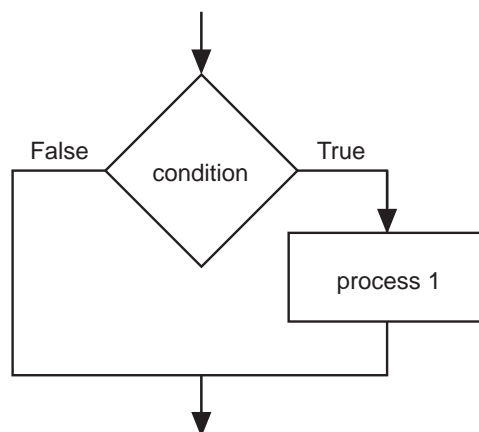
1. IF condition THEN
 process 1
 ENDIF

2. IF condition THEN
 process 1
ELSE
 process 2
ENDIF

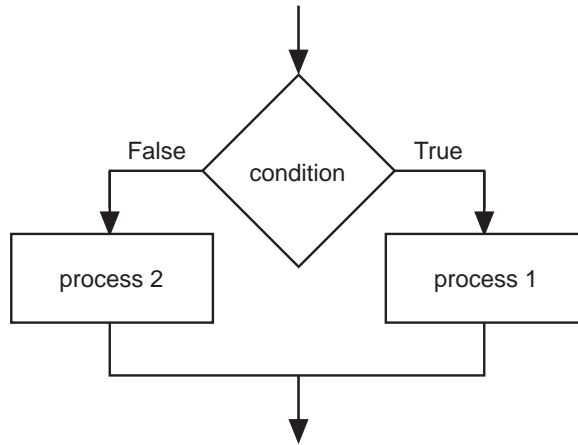


In flowcharts, binary selection is expressed in the following ways:

1.



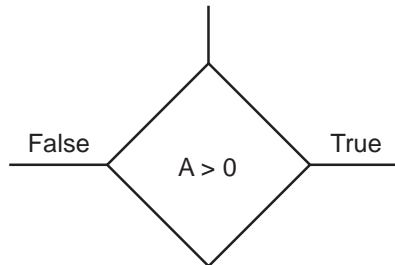
2.



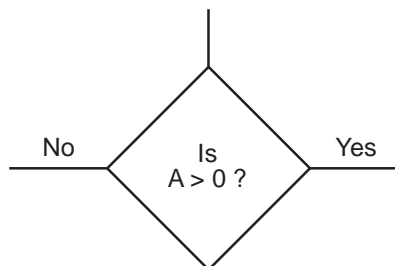
Note: In a flowchart it is most important to indicate which path is to be followed when the condition is true, and which path to follow when the condition is false. Without these indications the flowchart is open to more than one interpretation.

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:

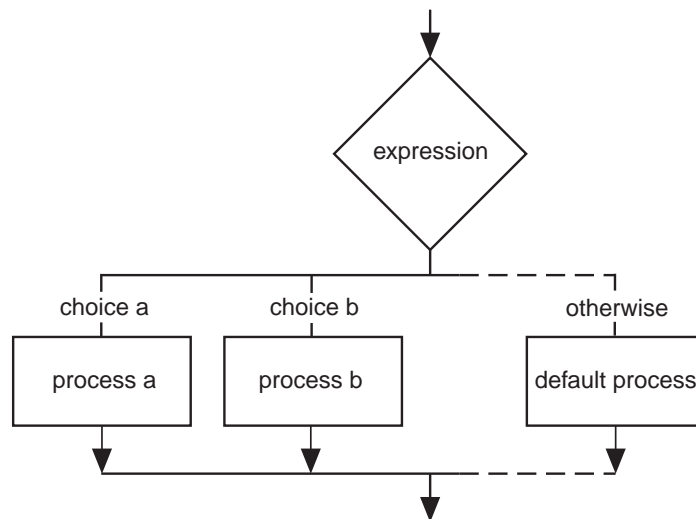
CASEWHERE expression evaluates to

```
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.

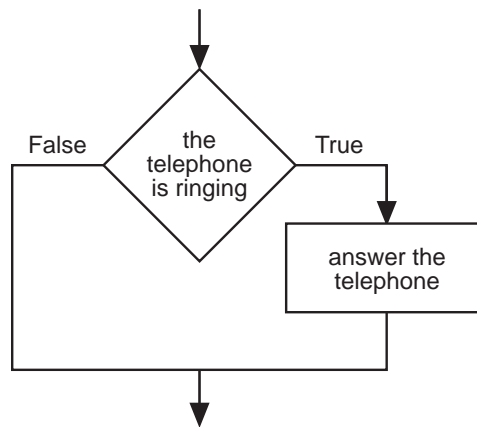


Pseudocode

```
IF the telephone is ringing THEN  
    answer the telephone  
ENDIF
```



Flowchart



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

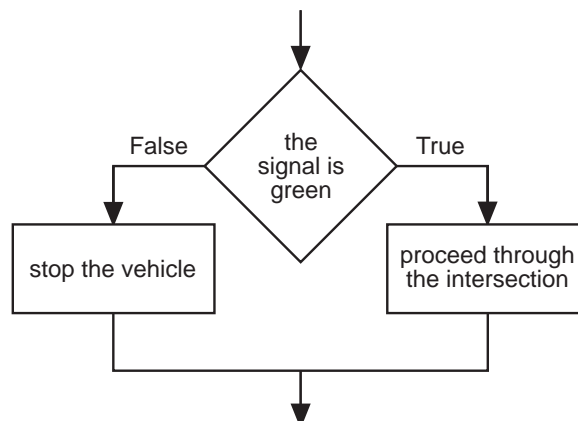


Pseudocode

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



Flowchart



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

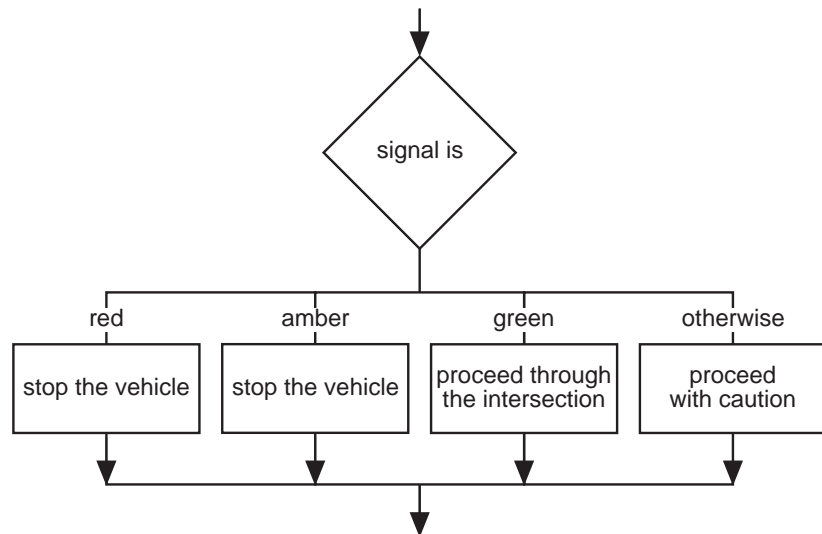
CASEWHERE signal is

```
red           : stop the vehicle
amber        : stop the vehicle
green        : proceed through the intersection
OTHERWISE    : proceed with caution
```

ENDCASE



Flowchart



Repetition

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.

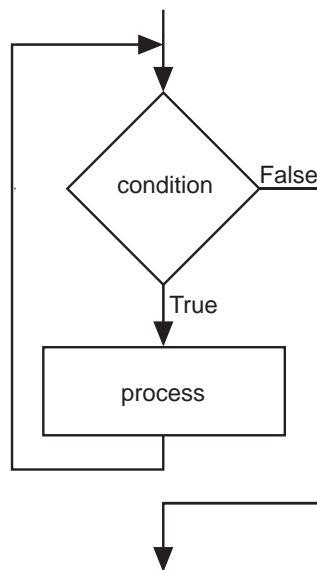


In pseudocode pre-test repetition is expressed as:

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



In flowcharting pre-test repetition is expressed as:



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.

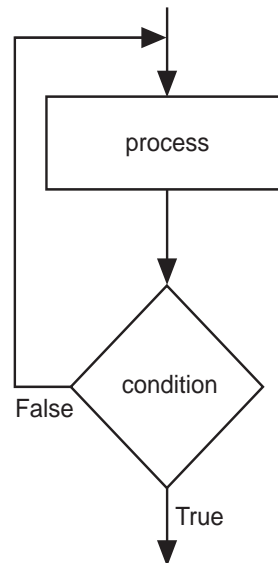


In pseudocode, post-test is expressed as:

```
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.

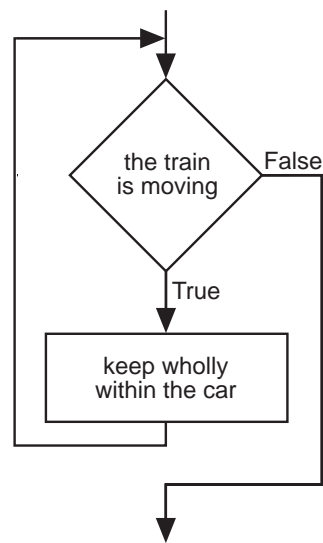


Pseudocode

```
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.

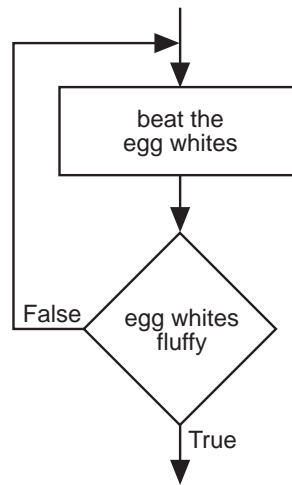


Pseudocode

```
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.

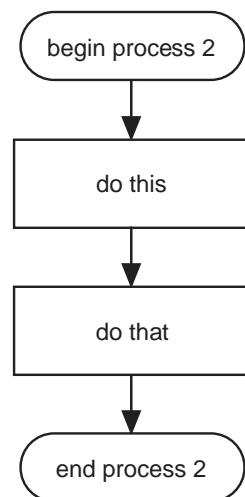
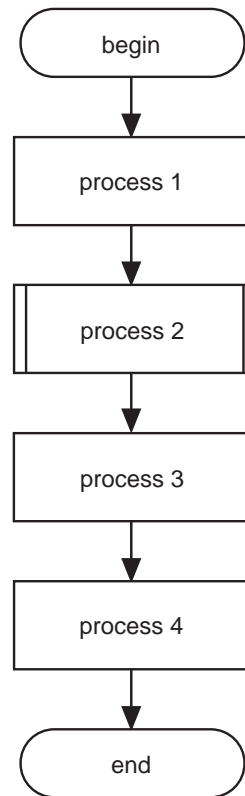


Example of Using Subprograms in Pseudocode

```
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



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 there is still more data AND there is room in the array  
    store data in the array at the position given by the  
    pointer  
    Increment the pointer  
    get data  
ENDWHILE
```

The first time that this subprogram is called, the character are read into the array called 'name' the second time, the data is characters are read into the array called 'address'.