Title	**Apply essential knowledge and concepts of software development to create a program**		
Level	**4**	**Credits**	**10**

Purpose	People credited with this unit standard are able to: demonstrate and apply essential knowledge and concepts of software development tools and techniques; apply knowledge of a third-generation programming language (3GL) to demonstrate understanding of the language semantics and simple algorithms; apply programming knowledge and skills to design, code, and test a simple program based on a supplied specification. This unit standard has been developed primarily for assessment within programmes leading to the New Zealand Certificate in Information Technology Essentials (Level 4) [Ref: 2594]. It is intended for people who want to develop the essential skills for further study that will equip them to work in the field of Information Technology (IT) as a profession, and who will generally not have previous computer programming experience such as digital technologies achievement standards.

Classification	Computing > Software Development - Programming

Available grade	Achieved

Guidance Information

1 Recommended skills and knowledge:
Unit 18740, *Create a simple computer program to meet a set brief*, and Unit 29778, *Use the main features and functions of a schematic diagram application to create diagrams*, or demonstrate equivalent knowledge and skills.

2 Assessment, where applicable, will be conducted in and for the context of real or realistic situations and/or settings, and be relevant to current and/or emerging practice. The assessor may gather evidence over time from a range of scenarios rather than using one assessment where the learner has to demonstrate all of the required skills. The assessment context for this unit standard must be suitable to meet the criteria for level 4 in the NZQF Level Descriptors, which are available by searching for “level descriptors” at www.nzqa.govt.nz.

3 Specifications will be supplied to the learner. The specifications will outline the problem that the program is intended to solve, and will include a clear description of the desirable outcomes sought, the constraints to be met by the solution, and requirements against which the success or otherwise of the program can be evaluated. The specifications will include operating and testing procedures, and requirements for input, processing, and output.

The provided test data must cover both valid and invalid inputs, and the test plan will describe various test scenarios, and list the expected outcomes. The program specification will define what range of data is valid, and the action(s) that should be taken when invalid data is input.

4 Definitions

1GL refers to first-generation programming languages (also known as *Machine language*), the most basic of computer languages, which are based on a machine's architecture and arithmetic logic unit. The instructions are represented in binary code, but may also be depicted in octal or hexadecimal notation for easier human interpretation.

2GL refers to second-generation programming languages (also known as *Assembly/Assembler Language*), which attempt to make a computer's machine language more assessable to a programmer by replacing the binary op-codes and machine addresses with meaningful words or mnemonics.

3GL refers to third-generation programming languages, with enhancements intended to make the programming language more programmer friendly. Suitable languages may include Javascript, C, C++, C#, Python, Java, Ruby and Perl.

4GL refers to fourth-generation programming languages, which attempt to make programming easier, faster, and cheaper than is possible with a 3GL. A typical 4GL incorporates an integrated development environment (IDE), with a code editor, database, a query language (e.g. SQL), GUI and report generators, and an embedded 3GL to handle complex/special requirements. Essentially, it enables a programmer to state a functional requirement (e.g. produce a report), and the system will generate the necessary code, without a programmer needing to code the details of how to achieve the outcome.

An *algorithm* is a finite list of instructions or steps needed to solve a problem.

A *compiler* is a program used in software development to translate code written in a high-level language (e.g. 3GL), into assembly or machine code in preparation for machine execution.

A *design plan* describes the module composition and logic/structure of the program being developed, and outlines how the requirements of the specifications will be realised.

A *desk check* is a manual process where a programmer visually inspects code (usually off-line with hard copy) checking for syntax and logic errors. This may require devising some basic input data in order to predict the program behaviour. Errors are located and corrected without help of compiler diagnostics or computer generated output.

Good practice in software development refers to conventions used and recommended by an organisation involved in the software development, computing or IT industry in New Zealand. It includes coding standards such as using meaningful variable names, indenting code to reflect program logic, and inserting comments to explain program logic.

Integer arithmetic refers to the process of evaluating arithmetic expressions which contain integers (positive and negative whole numbers plus zero) where any fractional intermediate results are discarded, producing an integer result.
An *interpreter* is a program used in software development which scans a high-level language program line-by-line, giving a programmer instant feedback on syntax problems as the code is being written.
Organisation refers to a specific entity which may be – in private, public, or community and voluntary sectors; a business, a discretely managed unit within a larger entity, a Māori organisation, or other special-purpose body.
Problem decomposition means breaking the problem down into smaller manageable components.
A *program* with a user-interface is an application, but many programs are not applications. An *application* program (or app for short) is any software program designed for an end user.
Programming puzzles refer to a type of program construction tool, such as *Parson's programming puzzles*, where the learner is given a set of code fragments (such as blocks of single or multiple lines of code in random sequence) to piece together a program that produces a specified outcome.

5 Legislation relevant to this unit standard includes but is not limited to the:
Copyright Act 1994
Copyright (New Technologies) Amendment Act 2008
Harmful Digital Communications Act 2015
Health and Safety at Work Act 2015
Privacy Act 1993
Unsolicited Electronic Messages Act 2007
and any subsequent amendments.
Current legislation and regulations can be accessed at http://legislation.govt.nz.

6 References
ACC5637 Guidelines for Using Computers - Preventing and managing discomfort, pain and injury. Accident Compensation Corporation - Department of Labour, 2010; available from Worksafe New Zealand, at http://www.business.govt.nz/worksafe/information-guidance/all-guidance-items/guidelines-for-using-computers.
Parsons, D and Haden, P; *Parson's programming puzzles,* Otago Polytechnic, Dunedin; Proceeding ACE '06 Proceedings of the 8th Australasian Conference on Computing Education - Volume 52, Pages 157-163 available at http://dl.acm.org/citation.cfm?id=1151890

Outcomes and performance criteria

Outcome 1

Demonstrate and apply essential knowledge and concepts of software development tools and techniques.

Performance criteria

1.1 Software development tools are identified, described, and compared in terms of their usage and application.

Range logic diagrams and pseudocode; assemblers, compilers, and interpreters; programming languages – 1GL, 2GL, 3GL, 4GL.

1.2 Small blocks/sections of provided code are desk-checked to locate syntax and logic errors.

Range up to 20 lines of provided procedural (3GL) code with a stated purpose;
provided code includes at least 5 syntax errors and 5 logic errors;
constructs must include sequence, iteration, and decision (if-then-else) elements;
at least 80% of the errors must be located and corrected.

1.3 A logic diagram is translated into logically correct and syntax-error-free 3GL code.

Range at least 10 elements in a provided logic diagram which include a loop and one or more decision elements;
may include the use of a compiler/interpreter to assist with syntax and logic error detection.

1.4 Logically correct, syntax-error-free code is translated into a well-formatted logic diagram which correctly reflects the code's logic.

Range a block/section of code is provided;
10 to 20 lines of undocumented 3GL code which contains decision and iteration statements.

Outcome 2

Apply knowledge of a third-generation programming language (3GL) to demonstrate understanding of the language semantics and simple algorithms.

Performance criteria

2.1 Simple 3GL logical and arithmetic expressions are correctly evaluated, given initial values for the expression variables.

Range includes at least five logical and five arithmetic expressions;
expressions include a mixture of standard operators (+ - * /; ^ (exponentiation); AND, OR, NOT) and brackets "()";
understanding of the respective operator precedence;
at least two variables and two operators per expression (arithmetic expressions only require integer arithmetic).

2.2 Correct solutions to programming puzzles based on simple algorithms are produced.

Range up to 10 lines of 3GL code which employ known language constructs and algorithms;
algorithms may include but are not limited to – exchanging the values stored in two variables; re-ordering the values stored in three variables into ascending or descending sequence; finding the largest or smallest element in an array; finding the mean value of a set of numbers inputted by a user; determining whether an array of numbers is in ascending/descending sequence; determining if a year, input by the user, is a leap year;
three algorithms are required, with at least one employing an array.

Outcome 3

Apply programming knowledge and skills to design, code, and test a simple program based on a supplied specification.

Range program includes but is not limited to 3GL and at least five of the following – modular structure, keyboard input, iteration, decision elements, input from a text file (comma separated data), simple built-in functions (e.g. date/time), text output to the screen and/or file, variables and expressions, arrays, nested decisions, simple repetitions (loop or iteration), files on disk.

Performance criteria

3.1 A design plan for the program is documented according to good practice to meet requirements of the supplied specifications.

Range design plan includes but is not limited to – problem decomposition diagram; key steps required in creating the program; logic diagrams or pseudocode;
plan may be written and/or graphic.

3.2 Program is coded and compiled to provide a solution using good practice.

3.3 Testing follows the test plan in the supplied specifications, and the test outcomes are documented.

3.4 The code is debugged and re-tested to eliminate errors and meet design plan and specifications.

Range all modifications are documented, including reasons for them.

Replacement information	This unit standard was replaced by unit standard 32946. This unit standard replaced unit standard 2795.

This unit standard is expiring. Assessment against the standard must take place by the last date for assessment set out below.

Status information and last date for assessment for superseded versions

Process	Version	Date	Last Date for Assessment
Registration	1	19 January 2017	31 December 2024
Review	2	28 April 2022	31 December 2024

Consent and Moderation Requirements (CMR) reference	0113

This CMR can be accessed at http://www.nzqa.govt.nz/framework/search/index.do.