**ASSIGNMENT OUTLINE**

**Achievement Standard 91906v1**

**Standard Number:**  91906v1

**Standard Title:** Use complex programming techniques to develop a computer program

**Credits:** 6

**Level:** 3

**Introduction/Kupu Arataki**

You are to create a complex computer program that uses variables of multiple data types, sequence, selection and iteration control structures, takes external input (like from a user) and produces an output.

It must also use at least two complex programming techniques from

1. A GUI,
2. Read or write to external file(s),
3. Object-oriented classes,
4. Define your own data types,
5. Incorporating third party APIs/libraries, - kinda
6. Complex data structures like stacks queues or trees.

Also, ensure your code is documented with comments, well tested and debugged.

You have 11 days to complete this assignment. It is due Monday 8th April at 4:30pm.

# Task/Hei Mahi

You may use the following ideas as starting points for your own project or you may come up with a unique idea of your own. **If you cannot come up with a project within two lessons, a project may be given to you.** Your idea, and the context it works in (i.e. how it is unique to you) must be discussed and agreed upon by your teacher before you start the work outlined in the planning section, below.

## Starting point ideas:

* **An ordering system for a take-away food shop**. A GUI menu takes input from the operator (menu items are chosen from a series of drop-down lists, including quantity) and interacts with an array of food items. Each order has a space for entering the name associated with the order. The program stores orders in an array. The program allows for take-away and delivery orders and calculates the final cost of the order. The menu drives all aspects of the ordering system.
* **A quiz program for testing knowledge of a particular topic**. A GUI based menu interacts with an array of questions. The questions are multiple choice. A record of correct/incorrect answers is kept, and statistics gathered about the responses. The menu allows for users to skip and retry questions as well as resetting the whole quiz. Multiple arrays are used to hold questions - attempted and not-attempted. The GUI includes drop-down lists and at least one of radio buttons or text-boxes.

Other projects that you may wish to consider as starting points are:

* **A sports analysis program** that takes GUI input of events/times/scores etc during a match or game or even series of games and allows the user to analyse the sports results.
* **A GUI resource-tracking program** that tracks what items have been given/borrowed to whom and when they are due for return. The program should be able to query the array of items for what is lent out, what is still in etc. This is a generic library program suitable for any type of resource.
* **A GUI based menu-planning program for a school camp** that allows users to plan a menu from a selection of common foods and determines if the menu is healthy or not. The program will need to use the calorie/KJ value of foods and compare it against ‘healthy’ criteria (e.g. total energy per day, contains vegetables etc). It will also need to be able to modify a menu if it does not meet the healthy criteria.
* **Find nearest emergency service.** This program would have coordinates stored for different emergency service buildings (fire, ambulance, police, fast food?) and when a user enters their coordinates (e.g. using a smartphone to be able to accept the input) and asks for a service, it would show the closest building for dispatch. Coordinates can be obtained from Google maps.
* **Gene sequence search program** that does a linear search through a gene sequence for a particular set of characters. The program could count how many occurrences there are in the substring and where they occur. A key to gene sequence files can be found at <http://www.genomatix.de/online_help/help/sequence_formats.html> and example gene sequence files can be found at <http://www.ncbi.nlm.nih.gov/genbank/samplerecord/> (scrolling to the bottom will reveal the gene sequence).

# Program requirements

## Development:

**Achieved:** Write code that performs your specified task and uses complex programming techniques. You should be able to show evidence of at least **TWO** complex programming techniques. Examples of complex programming techniques include writing code that:

1. Create a graphical user interface (GUI)
2. Read from, or write to, files or other persistent storage
3. Define class(es) and create objects
4. Define and uses custom type(s)
5. Use third party or non-core API, library or framework
6. Use complex data structures (e.g. stacks, queues, trees).

**Merit:** Follow the conventions of the programming language you have chosen

**Excellence**:

Ensure that the program is a well-structured, logical response to the task.

Make the program flexible and robust by doing at least two of the following

1. Use actions, conditions, control structures and, methods, functions or procedures effectively.
2. Check input data for validity.
3. Correctly handle expected, boundary and invalid cases. E.g. give feedback to the user when they enter an invalid input, and an option to retry.
4. Use constants, variables and derived values in place of literals

## Documentation

**Achieved:**

Programming code should be set out clearly.

Document the program with appropriate variable/module names and organised comments that describe code function and behaviour. Use appropriate variable/module names and follow conventions for your chosen programming language.

**Merit:** (Name things well)  
Appropriate variable/module names that describe their function.

Comments describe code function and behaviour with enough detail

## Testing and Debugging

**Achieved:**Test and debug the program to ensure that it works on a sample of expected cases.

**Merit**:   
Organise your test and debug of the program into logical order or sections. (This could possibly be shown in your versioning, or through screencasts.)

Test and Debug your program to ensure that it works on a sample of both expected cases and relevant boundary cases.

**Excellence:**Comprehensively test and debug the program.

**Planning Checklist / Rārangi Arowhai**

Task I’m going to program: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  | **Achieved** | **Merit** | **Excellence** |
| --- | --- | --- | --- |
| **Development** | Programming Language I will use: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  My main variables and their types: (e.g. Score – Float) *Note: must have two different types \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  A place I will use selection  (e.g. if/else):  *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  A place I will use looping  (e.g. do while/ for) *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  Main inputs from user/file/external source: *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  Main output produced: *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*  Two+ of the following I will use:  ● Program a graphical user interface (GUI)   ● Read from, or write to, files  or persistent storage   ● Define and use class(es) and objects   ● Define and use data types   ● Use third party / non-core API, library or framework   ● Use complex data structures e.g. stacks, queues, trees  | Main conventions of this programming language I need to follow  (e.g. uppercase/lowercase, punctuation, layout)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Do two of the following  ● Use actions, conditions, control structures and, methods, functions or procedures effectively   ● Check inputs for validity,  specifically, I will check for the following invalid inputs: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ● Correctly handle expected, boundary and invalid cases  (e.g. input < 3 characters) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  ● Use constants, variables and derived values in place of literals  *Name at least three:* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| **Documentation** | Program code set out clearly   Add comments  (e.g. a comment for every major block of code, and every complex line of code) | Appropriate variable/module names that describe their function.   Detailed comments describe code function / behaviour  |  |
| **Testing** | Program tested and debugged to work on normal expected cases. List at least six expected inputs to test: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Tests are grouped and organised logically and all fixed / successful   Boundary inputs are tested.  List at least two boundary inputs to test:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Comprehensively test and debug the program  e.g. invalid inputs to test: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

**MARKING SCHEDULE**

**Achievement Standard 91906v1**

|  | **Achievement** | **Achievement with Merit** | **Achievement with Excellence** |
| --- | --- | --- | --- |
|  | *Use complex programming techniques to develop a computer program.* | *Use complex programming techniques to develop an informed computer program.* | *Use complex programming techniques to develop a refined computer program.* |
| **Development** | Write code for a program that performs a specified task   Use variables with at least two types of data  e.g. numeric, text, boolean   Use sequence, selection and iteration structures   Take input from a user, file, sensors, or other external source   Produce output   Use two or more complex programming techniques e.g:   1. Program a graphical user interface (GUI)  2. Read from, or write to, files  or persistent storage  3. Define and use class(es) and objects  4. Define and use data types  5. Use third party / non-core API, library or framework  6. Use complex data structures e.g. stacks, queues, trees    Use a suitable programming language  | Follow conventions for the chosen programming language *including naming, layout, code checking*  | Ensure that the program is a well-structured, logical response to the task   Make the program flexible and robust. For example, two of   1. Use actions, conditions, control structures and, methods, functions or procedures effectively  2. Check input data for validity  3. Correctly handle expected, boundary and invalid cases  4. Use constants, variables and derived values in place of literals  |
| **Documentation** | Set out the program code clearly   Documenting the program with comments  | Document the program with: appropriate variable/module names   Organise comments that describe code function and behaviour  |  |
| **Testing** | Test and debug the program to ensure that it works on a sample of expected cases  | Test and debug the program in an organised way   Ensure that it works on a sample of both expected and relevant boundary cases  | Comprehensively test and debug the program  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Resubmitted** | **Not Achieved** | **Achieved** | **Merit** | **Excellence** |

**Assessment schedule/Mahere Aromatawai: Digital Technologies & Hangarau Matihiko 91906 – Developing for Purpose**

|  |  |  |
| --- | --- | --- |
| **Evidence/Judgements for Achievement/Paetae** | **Evidence/Judgements for Achievement with Merit/Kaiaka** | **Evidence/Judgements for Achievement with Excellence/Kairangi** |
| *Use complex programming techniques to develop a computer program.* | *Use complex programming techniques to develop an informed computer program.* | *Use complex programming techniques to develop a refined computer program.* |
| The student has:   * written code for a program that performs a specified task * used complex techniques in a suitable programming language   ***For example (partial evidence):***  *The student’s program allows users to enter typical data and outputs on expected cases.*  *Program has a graphical user interface and custom classes (e.g. one class might ‘hold’ the interface and a second class might include help text).* | * followed conventions for the chosen programming language   **For example (partial evidence):**  If the student has used Python, class names are in CapWords, variable and function names are lowercase, and classes appear before the main routine.  The student has used an automated tool to check that their code follows conventions. | The student has:   * ensured that the program is a well-structured, logical response to the task   ***For example (partial evidence):***  *Program code is easy to read/understand and has been set up in a logical fashion.*  *Functions and classes have been used to keep distinct tasks separate, and to avoid duplicate code.*  *The program explicitly passes data between classes and functions and avoids the use of global variables.*  *Where the program uses a GUI, the GUI and the underlying code are kept separate, and communicate via a well-defined interface.*   * made the program flexible and robust * comprehensively tested and debugged the program   ***For example (partial evidence):***  *Student provides evidence of comprehensively testing their program to show that it works correctly for expected, unexpected and boundary values.*  *It has been structured so that making changes to it is easy. For example, the code uses named constants, clearly defined in a ‘constants’ area.*  *The code uses derived values, such as the length of a list, in place of literals*.  *The examples above are indicative samples only* |
| * set out the program code clearly and documented the program with comments   **For example (partial evidence):**  Layout is clear, and whitespace has been effectively used.  Student has included comments stating what the code does. | The student has:   * documented the program with variable/module names and organised comments that describe code function and behaviour   Code has clear commenting throughout that describes function and behaviour. |  |
| * tested and debugged the program to ensure that it works on a sample set of expected cases   **For example (partial evidence):**  Student has provided evidence of testing their program. The testing might be missing some of the expected detail. It might miss some testing showing that the program works for unexpected/invalid values.  *The examples above are indicative samples only* | * tested and debugged the program in an organised way to ensure that it works on a sample of both expected and relevant boundary cases   **For example (partial evidence):**  Student provides evidence of systematically testing their final outcome to confirm that it works for expected, and relevant boundary values.  *The examples above are indicative samples only* | * comprehensively tested and debugged the program |

Final grades will be decided using professional judgement based on a holistic examination of the evidence provided against the criteria in the Achievement Standard.