Developing Software Introduction

H178 34

2021

# H173 34 Developing Software: Introduction

**Outcome 1**

Implement and test code to carry out tasks following a given design (page 2)

**Outcome 2**

Prepare technical documentation in line with good practice (page 3)

**Assessment task instructions**

This assessment will assess Outcomes 1 and 2. In this assessment you will write a piece of software in the language that you have been using throughout this Unit. The assessment is open-book and you are encouraged to refer to study notes and previously completed code. You are required to complete this work under supervised conditions so that your assessor can authenticate your work.

**You will be given a detailed design which you must follow when writing your code (APPENDIX 2).** **You will also be given a test plan which you must use to record the results of testing the software**.

Throughout this assessment you should adhere to good practice. A **statement of standard**s (APPENDIX 1) is included to demonstrate the required standard. You must ensure that you have indentation and internal comments throughout your code and that all variables and functions are appropriately named.

You will be assessed on your performance throughout the assessment and demonstration of the final product. You may demonstrate your performance at various stages of development. You will also be assessed on documentation produced and this will be presented in a technical manual.

# ASSESSMENT TASK 1

**OUTCOME 1**

**REQUIREMENTS SPECIFICATION**

You are required to write a program which carries out both string functions and number functions.

The program starts by asking the **administrato**r to enter their name.

The program then welcomes the administrator and presents a menu offering 3 choices:

**Option 1 —will generate a username.**

The program will display “This is option 1”, followed by the administrator’s name.

The **administrator** will then be prompted to enter the first and second name of a **member of staff** (**staff name**) **on one line. E.g Jack Russell**

The program will generate the username for the staff name by taking the first letter of the first name and the full second name.

e.g staff name is Jack Russell, username generated is JRussell

**Option 2 – calculate the factorial of a number**

The program will display “This is option 2”, followed by the administrator’s name.

The program will then calculate the factorial of a number entered by the administrator. The factorial of a number is defined as the product of that number and the number which is one less than it and the number which is one less than that, etc down to 1. For example, the factorial of 5 (or 5! as it is correctly written) is 5 \* 4 \* 3 \* 2 \* 1 = 120.

**Note** Output for printing the factorial should be in the format as below where the original number entered by the user is 5

**5 \* 4 \* 3 \* 2 \* 1 = 120**

**Option 3 – QUIT**

The program will display “This is option 3”, followed by the administrator’s name. The program will then quit.

The menu is **required to run until the user enters the quit option**.

You will write the program as a standalone application.

Please refer to statement of standards (APPENDIX 1), module design (APPENDIX 2) and data dictionary (APPENDIX 3) when writing.

# ASSESSMENT TASK 2

**OUTCOME 2 - Assessment task instructions**

Once you have completed your code and test logs you are required to prepare a **technical manual** which will include:

* Requirements specification (page 2)
* Detailed design (appendix 2)
* Code listing
* Test strategy (appendix 3)
* Completed test log (include screenshots of all tests)

**The bullet points listed above must be used as headings in your technical manual.**

You **will be provided with** the following documents which you require to complete the manual:

* Requirements specification (page 2)
* Detailed design (appendix 2)
* Test strategy (appendix 3)
* Test plan

**You** must provide:

* Code listing
* Completed test log with clearly labelled screenshots of all your tests

You are required to complete this work under supervised conditions so that your assessor can authenticate your work. You may refer to your notes and online help throughout the assessment.

# APPENDIX 1

**STATEMENT OF STANDARDS**

**Variables**

Variables should have names that are meaningful and represent the type of data that they are designed to hold.

e.g. int goalsFor, goalsAgainst; indicates that two integer values are to be used to represent goals for and goals against.

The scope of variables should be clearly defined. Where possible global variables should be avoided unless the variable is clearly going to be used throughout the code.

**Modularity**

Functions should be modular and carry out a single operation, eg getting numbers, calculating and printing results could be carried out in a single function but these are all discrete actions and should be carried out in their own function. Functions should have names that are appropriate to their functionality or operation on the code. For example a function that carries out string functions could be called string\_functions() or if it calculated the average of some numbers it could be called calculate\_average().

**Control**

The flow of a program will be determined by the functionality of the system and should be controlled during execution. Code should be written to allow selection of control paths based on decision making.

**Maintainability**

Each statement should be presented on a new line with appropriate termination. Code should be easy to read and make appropriate use of white space and indentation. In particular control structures must be indented and aligned with the begin and end indicators, this is most likely to be in the form of curly brackets to open and close sections of code. In addition it is essential that you **comment** your code to describe processes. This also provides evidence of your understanding.

**STATEMENT OF STANDARDS (CONT’)**

**Structure of a program**

All programs must follow the structure below. Dependant on the language environment some options may not be appropriate.

Opening comments

name

description of program

date

include libraries

declaration of constants

declaration of global variables

declaration of methods/functions

definition of methods/functions

description of methods

declaration of local variables

coding

**Testing**

All code must be thoroughly tested and the results of testing recorded. In addition this should be supported by screenshots of the tests. Testing will be carried out by you, the programmer, as the system is developed. This is known as glass or white box testing as you have knowledge of the code and what is expected. A test plan provides details of tests to be carried out and test data to be used. Once completed this becomes the test log for the system.

# APPENDIX 2 : Detailed Design

**MODULE DESIGN**

**Level 1**

1. prompt for and read in administrator name

loop

2. prompt for and read value for choice

3 act on administrator choice

while choice is not 3 to end loop

**Level 2**

1. prompt for and read in administrator name

do

2.1 prompt for and read value for choice

2.2 switch choice

case 1

3.1 call staff name input method

break

case 2

3.2 call factorial calculation method

while choice is not equal to 3 (quit option)

staff name input method

3.1.1 print administrator name, “This is option 1”

3.1.2 prompt for and read value for staff name

3.1.3 call create username method passing staff name as parameter

factorial calculation method

3.2.1 print administrator name, “This is option 2”

3.2.2 initialise variables

3.2.3 prompt for and read in number

3.2.4 if number greater than zero

3.2.5 loop for count is less than or equal to number

3.2.6 if( number – count is greater than zero

3.2.7 calculate factorial (factorial \* count)

3.2.8 print output

end if

end loop

3.2.9 print factorial

else

3.2.10 print message “Enter a positive whole number greater than 0”

**MODULE DESIGN (CONT’D)**

4 create username method

4.1 assign first character of staff name to string one

4.3 find position of the space in staff name

4.4 assign substring of staff name to string two starting at character after

space to the end

4.5 concatenate string one and string two and store in username

4.6 print welcome message to username

# APPENDIX 2 – DETAILED DESIGN (CONT’D)

**DATA DICTIONARY**

|  |  |  |  |
| --- | --- | --- | --- |
| **NAME** | **TYPE** | **SCOPE** | **DESCRIPTION** |
| *$\_COOKIE[ 'user' ]* | string | Global | Used to store administrator name which is then used as output for each option |
| $action | integer | Local to main() method | Used to hold value of input from user from menu options and will be used in the case statement |
| $fullName | string | Local to staff name input method | Full staff name which will be converted to a username e.g. Jack Russell |
| $paramStaffName | string | Input parameter of create username method | Parameter which holds the value of staff name from staff name input method and used throughout create username method |
| $startOfFirstName | string | Local to create username method | Variable used to store substring – first character of username |
| $spacePosOfStaffName | integer | Local to create username method | Variable to hold the position of the space in staff name. |
| $substrAfterSpace | string | Local to create username method | Variable to hold substring of all characters after the space |
| $factorialNumber | Integer | Local to factorial calculation method | Variable, initialised to 0 to store value of user entered number |
| $factorialResult | integer | Local to factorial calculation method | Variable, initialised to 1 to store value of calculated factorial of number |
| $i | integer | Local to factorial calculation method | Variable used in loop to count number of times loop is executed and for performing factorial calculation |

# FUNCTIONS

|  |  |
| --- | --- |
| **NAME** | **Description** |
| main() | Displays options to the screen and displays a prompt for input. Repeats indefinitely until user selects the quit option. Methods must be called dependant on user input. |
| *generateStuffName()* | Displays an output to screen, printing the administrator name and message informing that this is option 1. Then prompts for a staff name to be entered. This is in the format of a first name and a last name. The create username method is then called passing the staff name as a parameter. |
| *getCalculatedFactorialString()* | Displays an output to screen, printing the administrator name and message informing that this is option 2. Then prompts user to enter a number and check that it is an integer greater than zero. Then calculates and outputs the factorial of the given number or prints an error message and returns to the **main()** |
| *generateUsername()* | Carries out string manipulation on the staff name input parameter. It extracts the first character then extracts the text after the space and joins the two to create the username. |

# APPENDIX 3

**TEST STRATEGY**

All testing will be white box as it will be tested by the developer.

The system will be tested as it is developed. Methods will be created and the control of the program will be tested before any functionality is created. This will make use of stub testing which will be a simple output message.

Functionality will be tested in isolation once the method is completed. The results of this testing will be documented on the test logs. Functions will be tested to ensure that they are reliable on a standalone basis.

Once the entire program is completed performance testing will be carried out to ensure that the program meets the requirements and also that the execution is acceptable.

# TEST LOG SHEETS

CANDIDATE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Subroutine name** | **Test case** | **Test data** | **Expected result** | **Actual result** | **Comments** |
| main() | Enter the name of the administrator at prompt | <administrator name> | Name accepted and displayed in welcome message | Name accepted and displayed in welcome message | The name is displayed in the banner with “Welcome”.  e.g.”Welcome Jack Russel!” |
|  | Menu option 1 | 1 | Message informing that this is option 1 with administrator name | Generate User Name(<administrator name>) | This is displayed in the header of the dialog for option 1 |
|  | Menu option 2 | 2 | Message informing that this is option 2 with administrator name | Calculate Factorial (<administrator name>) | This is displayed in the header of the dialog for option 2 |
|  | Menu option 3 | 3 | System exits | System exits and go to log in page | The user’s session is removed |
| *staff name input and create username* | Create a username from two words | Jack Russell | Prompted to enter a name which is then displayed as JRussell | Prompted to enter a name which is then displayed as JRussell | First name – Jack  Last name – Russell  Result - JRussell |
| *factorial calculation* | Invalid data | 0 | Message prompting administrator to enter number greater than 0 | Message prompting administrator to enter number greater than 0 | Min=”1” is defined in input tag |
|  | Normal Valid data | 5 | Factorial of 5 result displayed | 5 \* 4 \* 3 \* 2 \* 1= 120 | This is displayed in the bottom |
|  | Extreme data | -5 | Message prompting to enter number greater than 0 | “Value must be greater than or equal to1” | This is validated by the min value in the input tag |
|  | Extreme data | 8.4 | Error message prompting for a whole number | “Please enter a valid value, The two nearest values are 8 and 9” | can input integer only |