



Diploma in Software Development

CS105

Development Principles II

(NZQF Level 5, 15 credits, version 1)

Assessment 2

Practical Test

Weighting within the course 50%

Candidate's Assessment Instructions

- This assessment is an **open book activity**, you can use your own course and review notes as well as offline or online resources, such as textbooks or online journals.
- You will each submit the identical piece of work separately for the work that requires collaboration.
- You can always ask your online tutor if you need further explanation or if the instructions are not clear.
- The purpose of this assessment is to assess your knowledge. As part of your academic and professional integrity, you must work alone on this assessment. In the event Yoobee suspects collusion, this will be addressed. For more information on plagiarism, please refer to the Student Handbook.
- Submit your completed assessment online in the correct space provided.
- Marks and feedback will be returned within 15 days of the submission date.

Submission/Deliverables of your Work

Submissions are accepted only via a Yoobee Online and all files should be submitted in a single zip file including:

- A word document which consists of code, output screenshots, UML diagrams and any other explanations/content asked in the assessment tasks. The word document must have your name, student id on the headers. Page numbers must be included in the footers. Use a font of either Times New Roman or Calibri. The minimum font-size must be 12 with at least 1.15 line spacing. Please ensure all diagrams, tasks, figures, or tables (if any) are numbered.
- A file containing the C++ solutions of all programming tasks asked in this assessment. Every C++ file must have comments with question number and full name.

Success Criteria

- You need to achieve 50% or more of the marks to pass this course
- You are allowed a maximum of three attempts. The maximum percentage to be awarded on second and third assessment attempts is 50%.

Learning outcomes:

1. Explore key principles, methods, tools and frameworks used in the development of mobile games.
2. Describe the processes used to solve development problems.
3. Demonstrate developing literacy in key programming languages for mobile game development.

Note: This Practical test contains two sections, in both sections, you can choose the tasks to complete, as follows.

Section 1 Choose one out of the two tasks to complete.

Section 2 Choose two out of the three tasks to complete.

Tips to test your application: please assign values to the variable in the program during the development to save time. On completion, comment out the hard-coded values and enable the user input.

Objective

To solidify your understanding of the design principles and Object-Oriented Programming paradigm taught so far, you will solve problems using C++ Object Oriented Programming concepts and develop & deploy an application.

Section 1 (Choose one out of the two questions.)

Question 1: Pointer to Objects

Iphone 12 pro max has a native app Health. The Health app stores the activities of the user such as number of Steps and Walking + Running in kilometres per day.



You are required to create a class HealthActivity and enable it to store the name of the user, number of steps and kilometres of walkingRunning. Use appropriate data types, access specifier, constructor, setter and getter functions.

Consider taking input for 5 users. Declare a pointer to an array of HealthActivity objects. Pass the pointer to SetFunction() and GetFunction(), to set the user input and display the user data with appropriate units respectively.

Find the average steps and average distance of Walking + Running of all the users and display them with appropriate units within the GetFunction();

Sample output

```
Microsoft Visual Studio Debug Console

Enter the name, number of steps and walking + running distance:Barry 3457 2.8
Enter the name, number of steps and walking + running distance:Benny 7342 10
Enter the name, number of steps and walking + running distance:Carol 8620 12
Enter the name, number of steps and walking + running distance:Dixon 1734 3
Enter the name, number of steps and walking + running distance:Merlyn 5532 6.4

Name: Barry
Steps: 3457 steps
Walking + Running: 2.8 Kms
Name: Benny
Steps: 7342 steps
Walking + Running: 10 Kms
Name: Carol
Steps: 8620 steps
Walking + Running: 12 Kms
Name: Dixon
Steps: 1734 steps
Walking + Running: 3 Kms
Name: Merlyn
Steps: 5532 steps
Walking + Running: 6.4 Kms
Average steps of 5 users : 5337 steps
Average distance of walking + running for 5 users: 6.84 Kms
```

Question 2: Classes & Objects and UML Class Diagram

Demand for online shopping during this lockdown has skyrocketed. Users are ordering several items online. You have been asked to write a C++ program, in which you have a class **Cost** that contains three data members, *dollar*, *cents* and *count*.

The *count* variable records a count of how many objects have been created so far from class **Cost**. The class **Cost** also has four member functions and a constructor to initialize data member value:

readCost(), which gets values for Dollar and cents from the user at the keyboard;

showCost(), which displays the cost in \$345.65 format;

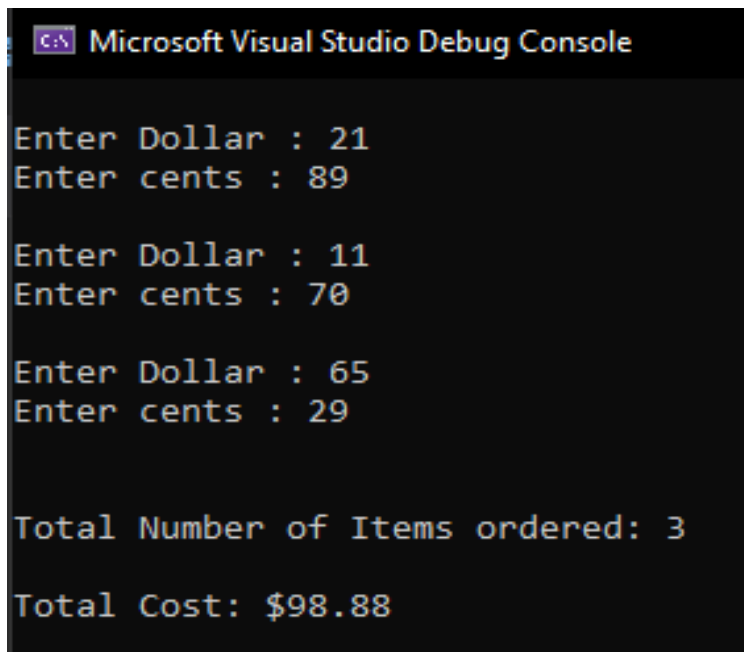
showCount(), which displays the count of objects created; and

computeTotalCost(), which sums all the items cost passes as an argument and returns total cost. The **computeTotalCost ()** function should not modify any of the data in the object for which it was called.

In the end the user would like to know total how many items were ordered using count and the total cost.

Also, prepare the UML class diagram for class **Cost**.

Sample output:



```
C:\> Microsoft Visual Studio Debug Console

Enter Dollar : 21
Enter cents : 89

Enter Dollar : 11
Enter cents : 70

Enter Dollar : 65
Enter cents : 29

Total Number of Items ordered: 3
Total Cost: $98.88
```

Section 2 (*Choose two out of the three questions*)

Question 1: Inheritance

MetLink is a public transport app that belongs to Wellington, NZ council. Wellington station displays information to the passengers as given below.

Details

Starting place: Wellington

Kapiti Line ends at: Waikane, Porirua, Kapiti

HuttValley Line ends at: Taita, UpperHutt, Melling

Johnsonville Line ends at: Johnsonville

Wairarapa Line ends at: Masterton

Train Status: Departed or Boarding

Cycle: Allowed, NotAllowed

Category: Express, AllStops

- a) Draw a UML diagram for the following reflecting the name of the class, access specifiers for data members, member functions, appropriate data types of data members and return types of getter functions. Clearly depict the relationship.

Class Name	Data Members (protected)	Member Function (public)	Relationship
Train	TrainNumber, Type (eg. Value - Passenger, Goods)	Setters, Getters	Parent
Trip	Line(eg. HuttValley, Kapiti), StartPlace, EndPlace, Driver, NumberOfCompartments, PlatformNumber, StartTime, EndTime	Setters, Getters	Child of Train

GoodsTrain	GoodsType	Setter, Getter	Child of Trip
PassengerTrain	DepartureTime, CycleAllowed, Category (eg. All stops, Express), Status (Departed, Boarding)	Setters, Getters	Child of Trip

b) Write down C++ program to perform the following.

- Create classes with its data members and member functions as shown above.
- Take input for all the data members depending on the type of train chosen.

Goods train does not need any input regarding cycle, category and status. Refer to the sample screen shown below:

```

=====
Metlink
=====

=====
Enter the Train number (eg. 123) and type of train (eg. Passenger / Goods): 378 Goods
Enter start place of train (eg. Wellington): Wellington
Enter end place of train(eg. UpperHutt): PalmerstonNorth
Enter driver's name(eg. Daniel): Stefanie
Enter number of compartments: 17
Enter platform number(eg. 3): 9
Enter start time(eg. 9:00): 10.05am
Enter end time (eg. 10:45): 12.17pm

=====
Enter the Train number (eg. 123) and type of train (eg. Passenger / Goods): 145 Passenger
Enter start place of train (eg. Wellington): Wellington
Enter end place of train(eg. UpperHutt): Kapiti
Enter driver's name(eg. Daniel): Marrito
Enter number of compartments: 5
Enter platform number(eg. 3): 2
Enter start time(eg. 9:00): 8:27am
Enter end time (eg. 10:45): 10:04am
Enter the line(eg. HuttValley): Kapiti
Enter whether cycle is allowed or not (eg. 0 for true / 1 for false): 0
Enter train status (eg. Departed / Boarding): Boarding
Enter the category (eg. Express / AllStops) : AllStops

```

```

Enter the Train number (eg. 123) and type of train (eg. Passenger / Goods): 103 Passenger
Enter start place of train (eg. Wellington): Wellington
Enter end place of train(eg. UpperHutt): Johnsonville
Enter driver's name(eg. Daniel): Robert
Enter number of compartments: 8
Enter platform number(eg. 3): 4
Enter start time(eg. 9:00): 2.10pm
Enter end time (eg. 10:45): 2.55pm
Enter the line(eg. HuttValley): Johnsonville
Enter whether cycle is allowed or not (eg. 0 for true / 1 for false): 1
Enter train status (eg. Departed / Boarding): Departed
Enter the category (eg. Express / AllStops) : AllStops

=====
Enter the Train number (eg. 123) and type of train (eg. Passenger / Goods): 911 Passenger
Enter start place of train (eg. Wellington): Wellington
Enter end place of train(eg. UpperHutt): Taita
Enter driver's name(eg. Daniel): Bryan
Enter number of compartments: 2
Enter platform number(eg. 3): 7
Enter start time(eg. 9:00): 11.45am
Enter end time (eg. 10:45): 1.00pm
Enter the line(eg. HuttValley): HuttValley
Enter whether cycle is allowed or not (eg. 0 for true / 1 for false): 0
Enter train status (eg. Departed / Boarding): Boarding
Enter the category (eg. Express / AllStops) : AllStops
Do you want to continue ? [y/n]y_

```

After taking the input, continue to display the main menu as shown below:

Option1: If “Goods Train” is chosen, the output should be shown as follows.

```

=====
Metlink
=====
a. Passenger Train
b. Goods Train
c. Exit
Please enter a or b to choose the type of train information you want or c to exit (eg. a,b,c):b_

```

```

=====
Metlink
=====
Sorry, it is under development! Press b to go back (b)b

```


Option 2: If Passenger Train is chosen then the following sub-menu must be displayed.

```
=====
Metlink
=====
a. Passenger Train
b. Goods Train
c. Exit

Please enter a or b to choose the type of train information you want or c to exit (eg. a,b,c):a
```

```
=====
Metlink
=====
a. Hutt Valley Line
b. Kapiti Line
c. Johnsonville Line
d. Wairarapa Line
e. Exit

Please enter a or b or c or d to choose the train line option or e to exit (eg. a,b,c,d,e):a
```

If HuttValley Line is chosen, then the information about HuttValley Train should be displayed.

```
=====
Metlink
=====

****HuttValley Line Train information****

=====
Train Number: 911      Platform number: 7
Wellington -> Taita
Start time: 11.45am    CycleAllowed

AllStops      Boarding
=====

Do you want to exit?[y/n]y_
```

After exiting and selecting b i.e. Kapital Line then it must show the following information:

```
=====
Metlink
=====
a. Hutt Valley Line
b. Kapiti Line
c. Johnsonville Line
d. Wairarapa Line
e. Exit

Please enter a or b or c or d to choose the train line option or e to exit (eg. a,b,c,d,e):b
```

```

=====
Metlink

=====
a. Hutt Valley Line
b. Kapiti Line
c. Johnsonville Line
d. Wairarapa Line
e. Exit

Please enter a or b or c or d to choose the train line option or e to exit (eg. a,b,c,d,e):c_

```

```

=====
Metlink

=====

*****Johnsonville Line Train information*****

=====
Train Number: 103      Platform number: 4
Wellington -> Johnsonville
Start time: 2.10pm      CycleNotAllowed

AllStops      Departed
=====

Do you want to exit?[y/n]

```

```

=====
Metlink

=====

*****Kapiti Line Train information*****

=====
Train Number: 145      Platform number: 2
Wellington -> Kapiti
Start time: 8:27am      CycleAllowed

AllStops      Boarding
=====

Do you want to exit?[y/n]y_

```

Option3: If “Exit” is chosen, then it must exit the application as below.

```
=====
Metlink
=====
a. Passenger Train
b. Goods Train
c. Exit

Please enter a or b to choose the type of train information you want or c to exit (eg. a,b,c):c_
```

```
C:\Users\092834\OneDrive - UP Education\Desktop\CS105\inheritanceQuestion\Debug\inheritanceQuestion.exe (process 19460) exited with code 0.
Press any key to close this window . . .
```

Question 2: Polymorphism

Consider a video games shop which sells video games of two types i.e., computer games and console games.

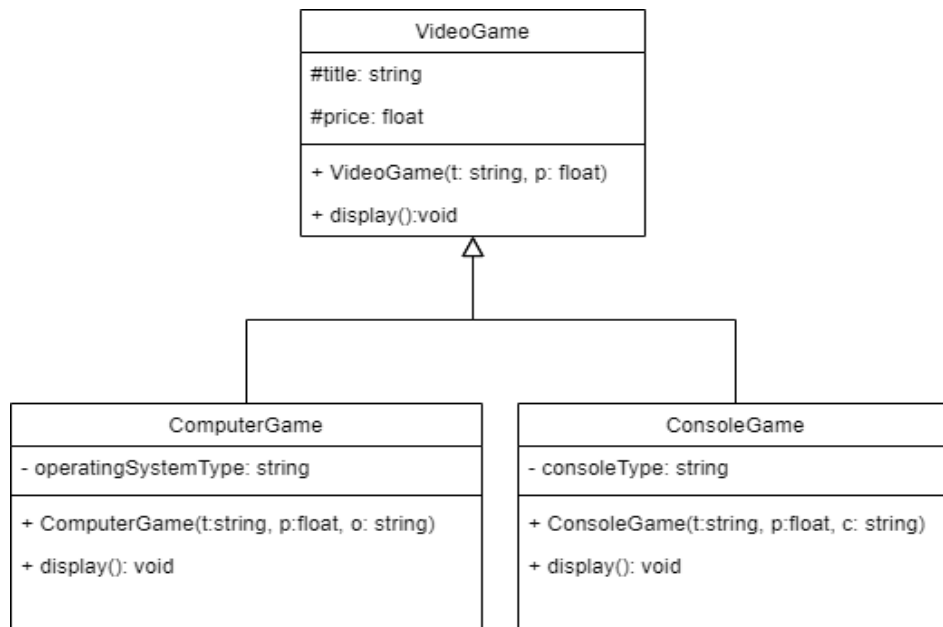
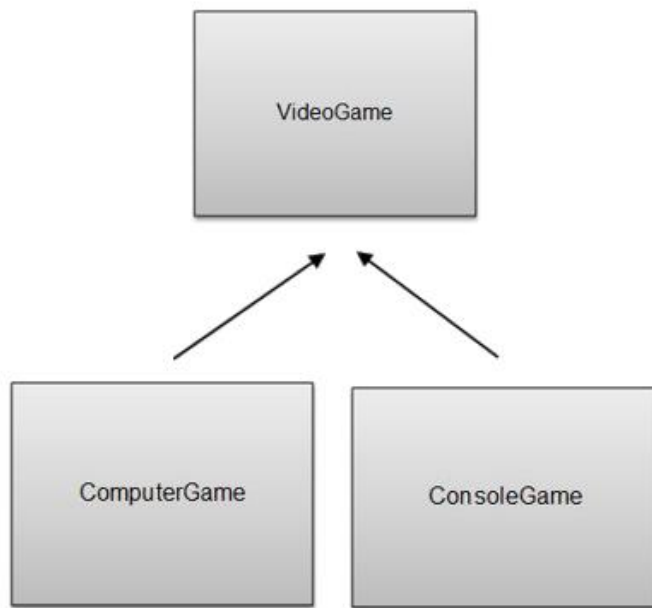
The shop records three data values for each computer game namely title, price, and operating system type.

For the console game, the shop records title, price, and console system type.

Suppose that you are hired as a C++ programmer to develop a simple data management system for the shop which will require you to do the following tasks:

- Create a class called VideoGame that stores the title and price of a game.
- Create two derived classes from the Base class VideoGame.
- The first derived class is called ComputerGame and is used for storing the Operating System (OS) type on which the computer game works i.e., Windows 10 or Mac OS for a computer game.
- The second derived class is called ConsoleGames and is used for storing the Console System type i.e., Xbox or PlayStation.
- Create a display() function that is used in all the classes to display the contents of video game objects.

Note: Refer to the class diagram given on the next page for the data members and member functions and their access specifications.



- Additionally, after creating the classes given above, also write a main program that creates an array of pointers to VideoGame objects. In an interactive loop, ask the store manager to record data about a particular computer game or console game and use new to create an object of class ComputerGame or ConsoleGame to hold the data.
- Store the pointer to the object in the array. When the store manager has finished entering the data for all the computer games and console games, display resulting data for all the games entered by the store manager, using a loop and a single statement such as:
videoGamesArray[i]->display();
 to display the data from each object in the array.

Sample Data Entry Screenshot:

```

Microsoft Visual Studio Debug Console
Video Games Data Entry
*****

Do you want to enter data for a Computer Game or a Console Game (o / c) : o
Please enter title of computer game: Age of Empires
Please enter price: 49.99
Please enter operating system type: Windows
Do you want to add another item : y
Do you want to enter data for a Computer Game or a Console Game (o / c) : o
Please enter title of computer game: 8th Wonder of the World
Please enter price: 10.99
Please enter operating system type: MacOS
Do you want to add another item : y
Do you want to enter data for a Computer Game or a Console Game (o / c) : c
Please enter title of console game: Arena Football
Please enter price: 9.99
Please enter console type: Xbox
Do you want to add another item : y
Do you want to enter data for a Computer Game or a Console Game (o / c) : c
Please enter title of console game: 007 Racing
Please enter price: 11.99
Please enter console type: PlayStation
Do you want to add another item : n

```

Sample Data Display Screenshot:

```

Microsoft Visual Studio Debug Console
Video Games List:
*****
Title: Age of Empires
Price: 49.99
OS Type: Windows

*****
Title: 8th Wonder of the World
Price: 10.99
OS Type: MacOS

*****
Title: Arena Football
Price: 9.99
Console Type: Xbox

*****
Title: 007 Racing
Price: 11.99
Console Type: PlayStation

*****

C:\Users\97009\source\repos\Virtual\Debug\Virtual.exe (process 13024) exited
Press any key to close this window . . .

```

Question 3: Operator Overloading

Write a program that overload three arithmetic operators + , - and * to perform addition, subtraction and multiplication respectively on two Complex numbers. You will need two complex numbers. Use a constructor to initialize 1st complex number values and ask users to enter 2nd complex number values.

Hint :

Formula
<p><u>Complex Number Arithmetic:</u></p> $(a + bi) + (c + di) = (a + c) + (b + d)i$ $(a + bi) - (c + di) = (a - c) + (b - d)i$ $(a + bi) \times (c + di) = (ac - bd) + (ad + bc)i$

Sample Output:

```

C:\> Microsoft Visual Studio Debug Console
1st Complex number :3 + 2i
Enter 2nd Complex number values:
Enter Real value : 1
Enter imaginary value : 7

Choose Operation from Menu:
1. Addition
2. Subtraction
3. Multiplication
4. Exit

Please enter your option:1
C1 : 3 + 2i
C2 : 1 + 7i
C3 : 4 + 9i

Please enter your option:2
C1 : 3 + 2i
C2 : 1 + 7i
C3 : 2 - 5i

Please enter your option:3
C1 : 3 + 2i
C2 : 1 + 7i
C3 : -11 + 23i

Please enter your option:4
  
```

Rubric

Assessment Criteria & Weighting		A Range	B Range	C Range	D Range
Section 1 – Choose 1 out of 2 tasks (LO1, LO2, LO3) 20%					
Task 1 – Pointer to Objects (LO1, LO2, LO3) 20%					
Class access specifiers, member data and data types (i.e. instance variables), getters and setters	4%	All member data variables and the access specifiers for member data have been correctly implemented along with meaningful and consistent variable naming conventions for the member data variables. The required getters and setters are present and appropriately named. Comments that provide clarity to the code are present.	All member data variables and the access specifiers for member data have been correctly implemented along with meaningful and consistent variable naming conventions for the member data variables. The required getters and setters are present but not appropriately named. Comments are present but have less clarity.	All member data variables and the access specifiers for member data have been correctly implemented but variable naming conventions the member data variables are inconsistent and not meaningful. Comments are missing or have less clarity.	The member data variables and the access specifiers for member data are incorrectly or incompletely implemented or have not been implemented at all. Comments are missing or have ambiguity.
SetFunction() and GetFunction() implementation	4%	Both member methods SetFunction() and GetFunction() have been correctly implemented with the correct return type and comments that provide clarity to the methods' functionality.	Both member methods SetFunction() and GetFunction() have been correctly implemented with the correct return type and comments that explain the methods' functionality are present with less clarity.	Both member methods SetFunction() and GetFunction() have been correctly implemented with the correct return type and no comments that explain the method's purpose are present.	The member methods SetFunction() and GetFunction() have been incorrectly or incompletely implemented or have not been implemented by the student.
Passing pointer to objects and functions	2%	Creating pointer to the required array of HealthActivity objects and passing it to the SetFunction() and GetFunction() is correctly implemented. Clear comments have been written to scaffold the code wherever necessary.	Creating pointer to the required array of HealthActivity objects and passing it to the SetFunction() and GetFunction() is correctly implemented. Comments have been written to scaffold the code wherever necessary but have less clarity.	Creating pointer to the required array of HealthActivity objects and passing it to the SetFunction() and GetFunction() is correctly implemented. Comments to scaffold the code are missing or have ambiguity.	Creating pointer to the required array of HealthActivity objects and passing it to the SetFunction() and GetFunction() is incorrectly implemented or not implemented at all. Comments are missing or have ambiguity.
Finding average number of steps and the average of walkingRunning distance	4%	Excellent implementation of the required functionality with appropriate handling of the input and return types. Clear comments have been written to scaffold the code wherever necessary.	A good implementation of the required functionality along with appropriate handling of the input and return types. Comments are present wherever necessary but have less clarity.	A reasonable implementation of the required functionality along with appropriate handling of the input and return types. Comments are absent or have ambiguity.	The required functionality is incorrectly or partially implemented or not implemented at all. Comments are missing or have ambiguity.
Program execution and output	4%	Output flows and screenshots exactly match the ones given in the assessment specifications. No bugs are present while executing the program.	Output flows and screenshots mostly match the ones given in the assessment specifications but with a few missing or incorrectly named labels. No bugs are present while executing the program.	Output flows and screenshots mostly match the ones given in the assessment specifications but with a few missing or incorrectly named labels. A few minor bugs are present while executing the program, but these bugs do not lead to program crashes or sudden termination.	Output flows and screenshots barely match the ones given in the assessment specifications. Moreover, there are serious bugs present while executing the program that cause the program to crash or suddenly terminate.
Use of Best practices	2%	An excellent understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A good understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	Little to negligible understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.
Task 2. Classes and Objects and UML diagrams (LO1, LO2, LO3) 20%					
Cost class creation – instance variables and member functions	4%	Cost class has been correctly implemented with the required constructors, member functions and instance variables. Student has demonstrated an excellent understanding of the data types	Cost class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated a good understanding of the data types	Cost class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated a reasonable	Cost class has not been implemented or partially implemented with the required constructors, member functions and variables. Student has

CS105 Development Principles 2

		needed for the Cost class' instance variables and member functions.	needed for the Cost class' instance variables and member functions.	level of understanding of the data types needed for the Cost class' instance variables and member functions.	demonstrated a poor level of understanding of the data types needed for the Cost class' instance variables and member functions.
Proper usage of static class data and const keyword	2%	An excellent understanding of static class data and the const keyword has been showcased in the student's code. Comments to scaffold the code are clearly written.	A good understanding of static class data and the const keyword has been showcased in the student's code. Comments to scaffold the code are written with less clarity.	A minimal understanding of static class data and the const keyword has been showcased in the student's code. Comments to scaffold the code are written with less clarity or are missing.	Little to negligible understanding of static class data and the const keyword has been showcased in the student's code. Comments to scaffold the code are ambiguous or missing.
Implementing object as function argument and returning object from function	4%	An excellent understanding of implementing object as a function argument and returning an object from the function is showcased in the student's code. Comments to scaffold the code are clearly written.	A good understanding of implementing object as a function argument and returning an object from the function is showcased in the student's code. Comments to scaffold the code are written with less clarity.	A minimal understanding of implementing object as a function argument and returning an object from the function is showcased in the student's code. Comments to scaffold the code are written with less clarity or are missing.	Little to no understanding of implementing object as a function argument and returning an object from the function is showcased in the student's code. Comments to scaffold the code are ambiguous or missing.
Program execution and output	4%	Output flows and screenshots exactly match the ones given in the assessment specifications. No bugs are present while executing the program.	Output flows and screenshots mostly match the ones given in the assessment specifications but with a few missing or incorrectly named labels. No bugs are present while executing the program.	Output flows and screenshots mostly match the ones given in the assessment specifications but with a few missing or incorrectly named labels. A few minor bugs are present while executing the program, but these bugs do not lead to program crashes or sudden termination.	Output flows and screenshots barely match the ones given in the assessment specifications. Moreover, there are serious bugs present while executing the program that cause the program to crash or suddenly terminate.
Use of Best practices	2%	An excellent understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A good understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.
UML Class Diagram	4%	UML diagram has been drawn using the correct UML notations and appropriately labelled. An excellent understanding of the Cost class has been showcased in the diagram.	UML diagram has been drawn using the correct UML notations and appropriately labelled. A good understanding of the Cost class has been showcased in the diagram.	UML diagram has been drawn using the correct UML notations and appropriately labelled with a few missing details. An adequate understanding of the Cost class has been showcased in the diagram.	UML diagram is missing or a partial UML diagram has been drawn with poor use of UML notations and missing many details. A poor understanding of the Cost class has been showcased in the diagram.

Section 2 – Choose 2 out of 3 tasks (LO1, LO2, LO3) 80%

Task 1 – Inheritance (LO1, LO2, LO3) 40%

UML Class Diagram	8%	UML diagram has been drawn using the correct UML notations for all 4 classes. An excellent understanding of access specifiers, inheritance, data members and member functions (i.e. function arguments and return types) of all classes has been showcased in the diagram.	UML diagram has been drawn using the correct UML notations for at least 3 classes. A good understanding of access specifiers, inheritance, data members and member functions (i.e. function arguments and return types) of the classes has been showcased in the diagram.	UML diagram has been drawn using mostly correct UML notations for at least 2 classes. A minimal understanding of access specifiers, inheritance, data members and member functions (i.e. function arguments and return types) of at least 2 classes has been showcased in the diagram.	UML diagram has been drawn for one or zero classes. UML notations are mostly missing or incorrect. Little to no understanding of access specifiers, inheritance, data members and member functions (i.e. function arguments and return types) has been showcased in the diagram.
Implementation of classes	12%	A clear class structure with an excellent understanding of access specifiers, member functions, getters & setters, inheritance hierarchies, program flow, function and loop handling, naming conventions and clear commenting have been showcased.	A clear class structure with a good understanding of access specifiers, member functions, getters & setters, inheritance hierarchies, program flow, function and loop handling, naming conventions and clear commenting have been showcased.	A class structure with less clarity along with a minimal understanding of access specifiers, member functions, getters & setters, inheritance hierarchies, program flow, function and loop handling, naming conventions have been	A missing or vague class structure with little to no understanding of access specifiers, member functions, getters & setters, inheritance hierarchies, program flow, function and loop handling, naming conventions have been

				showcased. Comments have less clarity	showcased. Comments are either missing or ambiguous.
Usage of functions to take user inputs, create and display menus	6%	An excellent understanding of program flow and handling of the main menu is showcased in the student's code. The functions that facilitate the flow of the main menu and processing user input have been appropriately named, completely implemented, input and outputs of these functions are appropriately handled.	A good understanding of program flow and handling of the main menu is showcased in the student's code. The functions that facilitate the flow of the main menu and processing user input have been appropriately named, implemented with some missing details, input and outputs of these functions are appropriately handled.	A minimal understanding of program flow and handling of the main menu is showcased in the student's code. The functions that facilitate the flow of the main menu and processing user input have been appropriately named, completely implemented with some missing details, input and outputs of these functions are handled with some minor glitches.	Little to no understanding of program flow and handling of the main menu is showcased in the student's code. The functions that facilitate the flow of the main menu are missing or not been appropriately named, inputs and outputs of these functions are poorly handled. Many of the functions required to handle the flow in the main menu are missing or partially implemented.
Passing pointer of objects to functions	2%	An excellent understanding of passing pointer of objects to functions is showcased in the student's code. Clear comments have been written to scaffold the code wherever necessary.	A good understanding of passing pointer of objects to functions is showcased in the student's code. Comments have been written to scaffold the code wherever necessary but have less clarity.	A minimal understanding of passing pointer of objects to functions is showcased in the student's code. Comments to scaffold the code are missing or have ambiguity.	Little to no understanding of passing pointer of objects to functions is showcased in the student's code. Comments to scaffold the code are missing or have ambiguity.
Function prototypes	2%	Function prototypes of all member functions belonging to the classes are correctly defined in the header files.	Function prototypes of most member functions belonging to the classes are correctly defined in the header files.	Function prototypes of some member functions belonging to the classes are correctly defined in the header files. But some member functions happen to be directly implemented without a header file.	Function prototypes of only a few or no member functions are correctly defined in the header file.
Program execution and output	6%	Output flows and screenshots exactly match the ones given in the assessment specifications. The different output displays for the type of train chosen are appropriately handled. No bugs are present while executing the program.	Output flows and screenshots mostly match the ones given in the assessment specifications but with a few missing or incorrectly named labels. The different output displays for the type of train chosen are appropriately handled. No bugs are present while executing the program.	Output flows and screenshots mostly match the ones given in the assessment specifications but with a few missing or incorrectly named labels. The different output displays for the type of train chosen are appropriately handled with some minor glitches. A few minor bugs are present while executing the program, but these bugs do not lead to program crashes or sudden termination.	Output flows and screenshots barely match the ones given in the assessment specifications. The different output displays for the type of train chosen are partially handled or not handled at all. Moreover, there are serious bugs present while executing the program that cause the program to crash or suddenly terminate.
Use of Best practices	4%	An excellent understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A good understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.
Task 2. Polymorphism (LO1, LO2, LO3) 40%					
Implementation of Base class – VideoGame	8%	VideoGame class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated an excellent understanding and handling of the data types, instance variables, member functions needed in the VideoGame class.	VideoGame class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated a good understanding and handling of the data types, instance variables, member functions needed in the VideoGame class.	VideoGame class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated a reasonable level of understanding and handling of the data types, instance variables, member functions needed in the VideoGame class.	VideoGame class has not been implemented or partially implemented with the required constructors, member functions and variables. Student has demonstrated a poor level of and handling of the data types, instance variables, member functions needed in the VideoGame class.
Implementation of Derived classes – ConsoleGame and ComputerGame	10 %	Both derived classes are correctly and completely implemented. An excellent understanding of inheritance and polymorphism	Both derived classes are correctly and completely implemented. A good understanding of inheritance and polymorphism	At least one derived class is correctly and completely implemented. A minimal understanding of inheritance and polymorphism concepts	Both derived classes are partially or incompletely implemented or not implemented at all. Little to no understanding of

CS105 Development Principles 2

		concepts has been showcased in the student's code.	concepts has been showcased in the student's code.	has been showcased in the student's code.	inheritance and polymorphism concepts has been showcased in the student's code.
Implementation of main function	4%	The main function has been correctly implemented in a concise and professional manner. The student has taken great care to see best practices such as appropriate and consistent variable naming conventions. Meaningful comments are present wherever necessary.	The main function has been correctly implemented in a concise and professional manner. The student has some care to see best practices such as appropriate and consistent variable naming conventions. Comments are present wherever necessary but have less clarity.	The main function has been correctly implemented but in a coarse manner. The student has taken little care to see best practices such as appropriate and consistent variable naming conventions. Comments are absent or have ambiguity.	The main function has not been implemented as per the requirement or is not implemented at all. The student has given little to no regard for best practices. Comments are absent or have ambiguity.
Array of pointers	4%	Array of Pointers to VideoGame objects with the correct size has been created and appropriately utilized in a concise and professional manner. Meaningful comments are present wherever necessary.	Array of Pointers to VideoGame objects with the correct size has been created and appropriately utilized in a concise and professional manner. Comments are present wherever necessary but have less clarity.	Array of Pointers to VideoGame objects with the correct size has been created and appropriately utilized but in a coarse manner. Comments are absent or have ambiguity.	Array of Pointers to VideoGame objects has been incorrectly implemented or is missing. Comments are absent or have ambiguity.
Loop that takes user input	6%	The interactive loop that takes user input has been correctly implemented in a clear and professional manner. Student's code has demonstrated an excellent understanding of this requirement.	The interactive loop that takes user input has been correctly implemented in a clear and professional manner. Student's code has demonstrated a good understanding of this requirement.	The interactive loop that takes user input has been correctly implemented but in a coarse manner. Student's code has demonstrated a reasonable understanding of this requirement.	The interactive loop that takes user input has been incorrectly or partially implemented or is missing. Student's code has demonstrated little to no understanding of this requirement.
Displaying contents of video game stock	4%	Student has demonstrated an excellent implementation of this functionality via their code. Data display screens exactly match with those of the assessment specification.	Student has demonstrated a good implementation of this functionality via their code. Data display screens mostly match with those of the assessment specification but have a few minor variations.	Student has demonstrated a reasonably viable implementation of this functionality via their code. Data display screens are meaningful, but the output labels (or text) are not very consistent with those of the assessment specification.	Student has demonstrated little no understanding of this functionality via their code. Data display screens are vague or incomplete or missing.
Use of Best practices	4%	An excellent understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A good understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.
Task 3. Operator overloading (LO1, LO2, LO3) 40%					
Implementation of the Complex Class	10 %	Complex class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated an excellent understanding and handling of the data types, instance variables, member functions needed in the Complex class.	Complex class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated a good understanding and handling of the data types, instance variables, member functions needed in the Complex class.	Complex class has been correctly implemented with the required constructors, member functions and variables. Student has demonstrated a reasonable level of understanding and handling of the data types, instance variables, member functions needed in the Complex class.	Complex class has not been implemented or partially implemented with the required constructors, member functions and variables. Student has demonstrated a poor level of and handling of the data types, instance variables, member functions needed in the Complex class.
Implementation of the Overloaded + operator	6%	An excellent understanding of operator overloading has been demonstrated via the use of appropriately implemented and invoked functions.	A good understanding of operator overloading has been demonstrated via the use of appropriately implemented and invoked functions.	A minimal level of understanding of operator overloading has been demonstrated via the use of appropriately implemented and invoked functions.	Little to no understanding of function operator has been demonstrated in the student's code.
Implementation of the Overloaded - operator	6%	An excellent understanding of operator overloading has been demonstrated via the use of appropriately implemented and invoked functions.	A good understanding of operator overloading has been demonstrated via the use of appropriately implemented and invoked functions.	A minimal level of understanding of operator overloading has been demonstrated via the use of appropriately implemented	Little to no understanding of function operator has been demonstrated in the student's code.

CS105 Development Principles 2

				and invoked functions. implemented.	
Implementation of the Overloaded * operator	6%	An excellent understanding of operator overloading has been demonstrated via the use of appropriately implemented and invoked functions.	A good understanding of operator overloading has been demonstrated via the use of appropriately implemented and invoked functions.	A minimal level of understanding of operator overloading has been demonstrated via the use of appropriately implemented and invoked functions.	Little to no understanding of function operator has been demonstrated in the student's code.
Program execution and output	8%	Output flows and screenshots exactly match the ones given in the assessment specifications. No bugs are present while executing the program.	Output flows and screenshots mostly match the ones given in the assessment specifications but with a few missing or incorrectly named labels. No bugs are present while executing the program.	Output flows and screenshots mostly match the ones given in the assessment specifications but with a few missing or incorrectly named labels. A few minor bugs are present while executing the program, but these bugs do not lead to program crashes or sudden termination.	Output flows and screenshots barely match the ones given in the assessment specifications. Moreover, there are serious bugs present while executing the program that cause the program to crash or suddenly terminate.
Use of Best practices	4%	An excellent understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A good understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.	A minimal understanding and handling of C++ syntax, data structures, loops, problem solving, naming conventions, comments etc has been showcased.
Sum	100% *				