Faculty of Computing, Engineering and Science

Assessment Cover Sheet and Feedback Form 2021-22

Module Code:	Module Title:	Module Team:
CS1D461	C++ Programming	Simon Payne, Shiny Verghese
Assessment Title and Tasks: Practical Coursework 2		Assessment No. 4
Date Set: 27-Sep-2021 08:00	Submission Date: 08-Apr-2022 17:00	Return Date: 06-May-2022 17:00

IT IS YOUR RESPONSIBILITY TO KEEP RECORDS OF ALL WORK SUBMITTED

Marking and Assessment
This assignment will be marked out of 100% This assignment contributes to 40% of the total module marks.
Learning Outcomes to be assessed (as specified in the validated module descriptor https://icis.southwales.ac.uk/): 1) To design, implement and test computer programs to solve a range of technical and mathematical problems. 2) To follow a secure design methodology and promote code re-use.
Provisional mark only: subject to change and / or confirmation by the Assessment Board

Marking Scheme:

	Fail	Narrow Fail	3rd Class / Pass	Lower 2nd Class / Pass	Upper 2nd Class / Merit	1st Class / Distinction
UML Class Diagram 10%	☐ Very Poor UML class diagram has been created. Many mistakes, using incorrect symbols and missing most of the objects in the requirements ☐ Very poor analysis of clients requirements.Very little of what the client asked for has been identified and represented ☐ Very untidy diagram. Not neat and not readable ☐ Lots of classes missing ☐ Very poor class names	☐ Poor UML class diagram has been created. It is missing some of the objects and relationships. Some mistakes in UML symbol usage ☐ Poor analysis of clients requirements. Less than half of what client asked for has been identified and represented ☐ Untidy diagram, Not neat and not very readable ☐ Quite a lot of classes missing ☐ Poor class names	☐ Satisfactory UML class diagram has been created. It uses correct UML symbols but the diagram is missing a few objects or relationships ☐ Satisfactory Analysis of clients requirements. Some of what the client asked for has been identified and represented ☐ Tidy diagram. Neat and readable ☐ Some classes missing ☐ Satisfactory class names	☐ Good UML class diagram has been created. It uses correct symbols and the majority of objects and relationships have been identified and correctly display on the UML diagram ☐ Good Analysis of clients requirements. Most of what the client asked for has been identified and represented ☐ Very tidy diagram, neat and easily readable ☐ 2 or 3 classes missing ☐ Good class names	☐ Very Good UML class diagram has been created. Uses correct symbols and the majority of objects and relationships have been identified, and correctly displayed on the UML diagram ☐ Very good Analysis of clients requirements. Almost everything the client asked for has been identified and represented ☐ Very tidy diagram, very neat and readable. Very good layout and attention to detail ☐ 1 class missing ☐ Very good class names	☐ Excellent UML class diagram has been created. It has identified all the objects and relationships asked for by the client. Solid OO relationships between the objects have been drawn in the class diagram ☐ Excellent Analysis of clients requirements. Everything the client asked for has been identified and represented in the diagram ☐ Excellent looking diagram. Very neat and tidy. Very easy to read. Good use or neat straight lines all touching the boxes they are pointing to ☐ All classes identified ☐ Excellent class names
Use Cases 10%	☐ Very poor use cases	☐ Poor use cases, little detail, hard to understand	☐ Satisfactory use cases, some detail, reasonable to understand	☐ Good use cases, good detail, easy to understand	☐ Very Good use cases, very good detail, easy to understand	☐ Excellent use cases, excellent detail, very easy to understand. Comprehensive in the coverage of client requirements
Other Design aspects 10%	☐ Very poor other design aspects accompanying the formal parts	☐ Poor other design aspects accompanying the formal parts	☐ Satisfactory other design aspects accompanying the formal parts	☐ Good other design aspects accompanying the formal parts	☐ Very good other design aspects accompanying the formal parts	☐ Excellent other design aspects accompanying the formal parts. Extra formal design may be included and are completed to a high standard
Code 30%	☐ Very poor. Code does not compile or run ☐ Very poor choice of data types and structures ☐ Very poor use of classes ☐ Very poor Relationships between classes	☐ Poor. Code compiles & runs, but the software does not fulfil the clients requirements ☐ Poor choice of data types and structures ☐ Poor use of classes ☐ Poor Relationships between classes	☐ Satisfactory. Code compiles & runs, but the software does only partially fulfil the clients requirements ☐ Satisfactory choice of data types and structures ☐ Satisfactory use of classes ☐ Satisfactory Relationships between classes	☐ Good. Code compiles & runs, but the software fulfils most of the clients requirements ☐ Good choice of data types and structures ☐ Good use of classes ☐ Good Relationships between classes	☐ Very good. Code compiles & runs, but the software fulfils all the client's requirements ☐ Very good choice of data types and structures ☐ Very good use of classes ☐ Very good Relationships between classes	☐ Excellent. Code compiles & runs, and the software fulfills all the clients requirements very well. Software is simple to use ☐ Excellent choice of data types and structures ☐ Excellent use of classes ☐ Excellent Relationships between classes
Testing 10%	☐ No or very little evidence of software testing	☐ Little evidence of software testing	☐ Some evidence of software testing	☐ Good evidence of software testing	☐ Significant testing of the program , well documented examples	☐ Comprehensive testing of the program , well documented examples
user guide 15%	☐ Very poor. There are gaping holes in the documentation	☐ Poor. The user guide is misleading	☐ Satisfactory. The user guide might contain minor omissions and errors	☐ Good. The user guide contains weaknesses in some areas	☐ Very good. The user guide was well written and contains useful relevant information	☐ Excellent. Well written documentation with very relevant screenshots, all

						functionality covered in an appropriate level of detail
Demonstration 15%	☐ Very poor. Code did not compile or other functional issues	☐ Poor. Code did not compile or work correctly. However, issues small and student can see how improvements code be made	☐ Satisfactory. A code compiles and operates generally to requirements. There may be some issues Student shows basic understanding of how issues could be fixed	☐ Good. Code compiles and operates to requirements. There may be some slight issues. Student shows good understanding of how issues could be fixed, or what improvements could be made to the code	☐ Very good. Code compiles and operates., very few issues. Interface should be well presented.Student can describe the code well	☐ Excellent. Code compiles and operates well. Interface is easily used and operation clear. May implement additional functionality where this doesn't go beyond the spirit of the task or impair functionality required. Student can explain the code excellently

Tasks

You are required to write an Object-Oriented C++ program using Classes and relationships between classes. to fulfil one of the client briefs listed below. You only need to write a program for one of the client briefs, the choice is up to you.

Client Brief 1: Rugby, Football, or another team sport Manager

You have been asked by your local sports club (up to you to choose the sport) to produce a simple application that can easily gather players availability for a match and produce a team sheet ready for the coach to look at. The application should be able to generate a team from the currently available players, putting them in the correct position, and choosing the best player in each position depending on the player average rankings. Ideally the application should be able to manage several teams (different age groups, 1st's 2nd's etc) but only for one sport. The coach has also asked if a history of matches could be stored, and stats generated from them. Before each match the coach will create a match in the system, and then input player availability, the system will then pick the best team from the average player ratings. The coach should be able to overrule the player ratings in the team selection and select his own players if he wants to. After each match the coach will enter the score, and give each player a rating out of 10, which will adjust their average rating. Data should be stored in a text file.

The application will store information about the **Teams**

- Team name
- Recent **Matches**
- **Head Coach**

The application will store information about **Matches**

- Opponent
- **Players**
- **Assistant Coach** (for the match)
- **Physio** (for the match)
- **Team Captain**
- Opponents Score
- Our Teams Score
- Result (Win, Loss, Draw)

The application will store information about **Players**

- Name
- Position
- Average Rating
- Current Availability
- Injured

The application will also store information about the support staff (There could be multiple of each)

- **Head Coach**
  - Name
  - email
  - Phone number
- **Assistant Coach**
  - Name
  - Email
  - Phone number
  - Coaching Expertise
- **Physio**
  - Name
  - Email
  - Phone number
  - Physio Specialty

Client Brief 2: Music Shop

You have been asked to build an application for a local music shop. The music shop mainly sells guitars, brass & woodwind instruments, and other accessories, but also provides lessons which customers can book. Customers can only book a lesson for an instrument that the shop sells. The application will be used by the employees at the music shop whilst talking to customers either on the phone or in person. The application should then be able to produce an invoice for the customer to pay for the booking of lessons, or for purchase of products. The application should be able to show a list of bookings for a given day. The application should have a feature where a user can search for an instrument or accessory. The application should also produce a sales report, that shows the total sales from instruments and the total income from the lessons. Data should be stored in a text file.

The system will need to store information about the **Instruments**

- Instrument Name
- Make
- Model
- Notes
- **Stringed Instruments**
  - Number of strings
  - Type (Acoustic, Electric, Bass etc)
  - colour
- **Brass Instruments**
  - Number of valves
  - Key B♭ E♭ etc
  - Material
  - Finish (Brass, Silver)
- **Woodwind Instruments**
  - Number of Keys
  - Material
  - Key B♭ E♭ etc

The system will need to store information about the **Customers**:

- Name
- Address
- Phone Number
- Notes

The system will create **bookings for lessons**:

- **Instrument**
- **Customer**
- Lesson required (Beginner, Intermediate, Advanced)
- Date
- Time
- Block booking (i.e. do they want a set of bookings e.g. for 5 weeks)
- Quoted Price
- Notes (for after the lesson)

The system will also keep an inventory of **Accessories**:

- Accessory Name
- Price
- Number in stock
- Suitable for **Instruments** (a vector or array of instruments the accessory is suitable for)

Client brief 3: Restaurant Take-away system:

You have been given an opportunity to build a system for your friend who is setting-up a restaurant in Treforest. The restaurant currently offers only take-away orders that could either be collected in-person or to be delivered within a 1-mile radius. There is an upper limit set by the owner of the restaurant for the take-away orders that can be booked per day. The restaurant offers two types of cuisines (for e.g. Italian and Chinese). The application should be able to generate a report of the following:

a. Total no of orders that have been booked.
b. Total income for the day.
c. Most popular cuisine.
d. Most popular item from the menu.
e. Loyal customers

The application will store information about the restaurant staff:

Chef
- Name
- Address
- Phone Number
- Cuisine specialist

Delivery Staff
- Name
- Address
- Phone Number

Manager
- Name
- Address
- Phone Number

Receptionist
- Name
- Address
- Phone Number

The application will store information about the Menu items:

Menu Item
- Cuisine
- Item name
- Availability
- Item price
- Popularity rating

The application will also store information about its customers and their orders

Customers
- Customer Name
- Contact
- Address

Order
- **Customer**
- **Menu Item**
- Type of order (in-person or delivery)
- Total price

Task 1 Report (40%)

Once you have chosen the client Brief you want to work on produce a report the content of which covers:

- The design process for the code you are going to write. This time you must produce a more formal design using Use Cases, UML class diagrams & a flow chart. A few paragraphs are also permitted.
- From you use cases, create test cases and list them in this report.

Then once you have completed your code:

- Results of the tests cases being carried out, with results and relevant evidence
- A brief review of your code highlighting areas for improvement

Task 2 Code (30%)

Write the software to fulfil the client's requirements. The code must be Object-Oriented. I.e., you must use classes and have relationships between the classes. Code must be written in C++. Make sure to write the code neatly and use comments as appropriate.

Task 3 User Guide (15%)

Write a user guide for your application. It should be aimed at the end user of the system. You can include annotated screenshots to make it easier to read.

Task 4 Demonstration or Video (15%)

You will be required to demonstrate your code, either in person in a timetabled lab session, or you could record a video of your application working with you narrating the actions being carried out. If you choose to create a video, be sure to show all aspects of the application working.

Advice

This assessment is designed to be as close to a real-world software project, with all the steps usually required when developing software in the real world. Think about your chosen client specification carefully. If you are not sure how to design the system, or you are not sure what is being requested then please ask. We can discuss the client briefs in the labs. We are not expecting the same system from everyone, software development is a creative process and there is no one perfect solution. Build the software how you feel is the best way to meet the clients' requirements.

Think about how it would work in the "real world". Feel free to be a bit creative with your solutions, if you think of extra features to add then please feel free to try them. But be sure to complete everything that has been specifically requested first. Writing software is a creative process and things can change, don't be afraid to go back and change your design. Another thing to consider, is how easy it would be to extend your application.

Be as detailed as possible in your design and do the design first. It will make your code easier to write. When creating your UML diagrams & writing you use cases, try to write them in a way that any programmer could potentially pick up the design and write the same code you would. Don't leave anything to assumption or chance when documenting your design.

Less is more in programming. Don't overcomplicate things and try to reuse code as much as possible. Read the marking RUBRIC carefully this will tell you how we will mark the assessment.

Submission

- Submit your report to Turnitin as a Word Document.

- Submit your code as a C++ file to CodeGrade The file should be called **practicalcoursework2.cpp**. Be sure to submit the C++ file containing the code and not any other visual studio file.

- Submit your user guide to Turnitin as a Word Document.

- If you decide to record a video submit that in the format it has been recorded in. Preferably mp4. This will be submitted through Panopto. Links will be available on blackboard.

Submission will be through blackboard, submissions via email will not be accepted.

Read the marking RUBRIC carefully, this will show you how we will mark the assessment.

Individual Assignment

This is an individual assignment and thus **the work submitted must be your own.**

Feedback

Feedback will be provided through the marking system SAFE.