Test yourself by writing the answers out by hand – pen

*Your name:*______________________

Full-time study *Part-time study* ***(delete one)***

All questions are handwritten answer questions. C++ coding is needed. Theory/Concept questions would need C++ code demonstration of the theory/concept.

Handwrite the answers first, including code. Desk check your handwritten code using a carefully handwritten test plan. Do not type code onto a computer first, as you can get the illusion that you have learnt when you might not have. There is some research on handwriting vs keyboard. See work by Mangen et al; New York Times; Kiefer et al; The Guardian also reports on test subject who were university students; Mackenzie; Handwriting for developing cognitive skills; Advantages over typing; EEG studies.

Important advice:

- **When writing code to solve a given problem or code a data structure, you can also be asked to provide a test-plan, a test program and modular unit tests for data structures or classes. The test plan/program should check for normal operation and should also break any invalid module or data structure being tested. "Break" can also mean abnormal termination. Test plans and programs are handwritten.**
- **Just knowing the theory/concept has little to no mark allocation. Most of the marks are in the demonstration of the theory/concept using C++ code and explanation using code – explaining how the code fits the theory/concept.**
- **Just getting the code correct is not sufficient to pass. Proper design that applies the theoretical concepts is essential. As a worst case example, writing everything in one *main()* function would result in a mark of 0 even if the program works perfectly. The same result will be obtained if global variables are used.**
- **Answer the questions as asked. You will get no marks for regurgitation of memorised code when the code is not relevant to the question. It is quite a common mistake to believe that just because you wrote a lot in the exam, you will pass. So read the question carefully to work out what is required, then write your answer to address the requirements. The more you write a wrong answer, the more confident we are that you do not know the answer.**
- **Your mark will be halved for any coding answer that has implementation within the body of the class declaration. Only method prototypes allowed. Unlike Java, the implementation is outside the declaration of the class. If you feel you need to inline any method, provide written justification for doing so as part of your answer.**

- In the exam, rough answers can be written using a pencil. Answers that you want marked should be written using a black or blue pen.

1. Write a C++ program that prompts the user to input the elapsed time for an event in hours, minutes, and seconds. The program then outputs the elapsed time in seconds. An object-oriented solution is **not** needed for this question. What test data would you use to make sure that your program works. You need to provide reasons for selecting the test data.

2. Write an Object-Oriented C++ program that prompts the user to input the elapsed time for an event in seconds. The program then outputs the elapsed time in hours, minutes, and seconds. What test data would you use to make sure that your program works. You need to provide reasons for selecting the test data.

3. Write an OO C++ program that reads a list of integer numbers from a data file called cents.txt. The numbers in the data file are separated by spaces or newline characters. Each value in the data file represents an amount of money in cents.

   Your program prints out: the value in cents and what this value represents in dollars and cents.

   Sample output of the program when the file cents.txt has two numbers this time, but there can be more numbers in other situations.
   10 235

   Output:
   ```
   10 cents   is 0 dollars and 10 cents
   235 cents is 2 dollars and 35 cents
   ```

   Describe the data structure(s) to be used to solve the problem above. What alternative data structures could be used and what advantages do your chosen data structures have over the alternatives?

   What design pattern can be used in your solution? How would this pattern be used for other solutions that may involve different conversions? Use UML diagrams to explain.

   Explain how your solution is designed to fit the Model-View-Controller (**MVC**) pattern. Your explanation needs to clearly identify which class(es), subroutines (functions, procedures) fit into which part of MVC. You should also explain why they fit into the MVC components.

   Write a test plan for the application.

4. The manager of a football stadium wants you to write a program that calculates the total ticket sales after each game. There are four types of tickets with each type having its own price. After each game, data is stored in a data file called *sales.txt*. The data file format is as shown below:

   Ticket price   numberOfTicketsSold
   .....

Sample data is shown below. You must not assume that this will be the data after every game. There is only data and no column headers. Data items on the same line are separated by white space.

```
250  5750
100  28000
50   3570
25   18750
```

The first line indicates that the ticket price is $250 and that 5750 tickets were sold at that price. Your program outputs the number of tickets sold and the total sale amount.

Explain how your solution is designed to fit the Model-View-Controller (**MVC**) pattern. Your explanation needs to clearly identify which class(es), subroutines (functions, procedures) fit into which part of MVC. You should also explain why they fit into the MVC components.

5. Using UML, design a birthday list to keep track of the birthdays of any person (friends, family, colleagues, etc) or pets. Think about what goes in a birthday list first before you start designing.

   Think of the classes, structures, data structures used, and what information is to be stored.

   What menu options would your program have?

   You should use the highest level of abstraction possible. Identify the different classes and their relationships using UML.

   Implement your birthday list in C++ and write a test program to show that your birthday list works. Using code comments indicate which are parts of your solution are in which components of **MVC**.

6. UML allows us to depict class relationships in Object Orientated design. Using UML, explain two such relationships: *Specialisation* and *Realisation*. You must draw the UML diagrams.

   a. Under what conditions would you use *Specialisation*?
   b. How is *Realisation* different to *Specialisation*?

7. **Using C++** code, implement an example <u>of each</u> of the following relationships from the previous question: *Specialisation*; *Realisation* - implement the UML designs.

8. Using any relevant C++ code example (or examples) of your choice, explain what is meant by “designing or coding classes in a minimal but complete manner”. Using appropriate C++ code example (or examples), explain how you would go about providing additional functionality to a class without creating additional methods or friends of the class?

   How would you namespaces to manage your design for the above?

9. Using C++ examples, explain how C++ templates provides a mechanism for abstraction.

10. Explain why languages such as C++ provide the template mechanism. Provide an example of a C++ template subroutine (must not be a method of a class) to demonstrate the utility of templates. You need to provide an explanation as well as code.

11. Discuss the validity of the statement "A class is a data type". Use C++ examples to illustrate your answer. Just writing code is not sufficient. You need to explain why the statement is valid or not valid.

12. Write a complete **minimal** but **complete** template Vector class using C++. For our purposes, a template Vector class is a dynamic array encapsulated inside the Vector class. The Vector class needs to be minimal but complete. Public methods should not provide overlapping or redundant functionality. You decide and defend what is necessary. Using C++, provide a full and modular unit test for the Vector class. STL data structures must **not** be used.

13. Write a complete template Vector class using C++. The Vector class needs to be minimal but complete. Public methods should not provide overlapping or redundant functionality. Using C++, provide a full and modular unit test for the Vector class. An appropriate STL data structure **must be used** inside your Vector class. You decide and defend what is necessary. Explain why this data structure is appropriate.

14. Write an application program in C++ that makes use of your Vector class. This application will first read all the book titles stored in a data file called *titles.txt*. The data is stored into your Vector. The application program then prints out the titles to the screen in the same order that was stored in Vector. Every line of the data file has exactly one title. There can be any number of titles in the data file. A sample of data that can be in the data file is shown below:

    ```
    Design Patterns
    Algorithms and Data Structures in C++
    Advanced Methods in Neural Computing
    ```

    Explain how your solution is designed to fit the Model-View-Controller (**MVC**) pattern. Your explanation needs to clearly identify which class(es), subroutines (functions, procedures) fit into which part of MVC. You should also explain why they fit into the MVC components.

15. Write a C++ subroutine called *fibonacci* that takes a number n as a parameter and returns the first n numbers in the Fibonacci series in an STL vector. So fibonacci(7) would return 0, 1, 1, 2, 3, 5, 8 in an STL vector. Write a test program to test your routine. You do **not** need to design an OO solution. The only object(s) needed would be the STL vector object(s).

16. Write a minimal but complete template Binary Search Tree (BST) using C++. Using C++, write a complete and modular unit test for your BST. For one of the tests, a BST of dates would be needed. Consequently, you will also need to write a Date class. Test your BST with a BST<Date>.

17. Implement recursive insert, delete tree and traversals for the BST.

18. What is the run-time performance (Big-O) of the BST's search method? What situation(s) would affect this run-time performance giving a worst-case Big-O? What is this worst-case Big-O performance and explain the structure of the tree that would give a worst-case scenario.

19. Provide recursive C++ implementations of methods to perform the following BST operations: in-order, post-order, and pre-order traversals; insert into tree and delete the whole tree.

20. Using UML, design a **minimal** but **complete** linked list data structure. Implement a template linked list in C++.

21. Using C++ examples, explain the concept of **inheritance**. Your code should demonstrate how inheritance works.

22. Using C++ examples, explain the concept of **dynamic** polymorphism. Your code should demonstrate how polymorphism works. Use code comments to explain the critical requirements for dynamic polymorphism to occur.

23. Using C++ examples, explain the essential difference between a stack and queue. Can a stack be implemented using a queue or vice-versa? Use C++ code to demonstrate how such implementations might occur.

24. Using UML notation, design a minimal but complete template class for the Stack data structure. The class design must show all details of the data structure. All public methods should be named appropriately. Parameters and return types must be shown. Two of the methods that your Stack class must have are "push" and "pop".

25. Using UML notation, design a minimal and complete template class for the Queue data structure. The class design must show all details of the data structure. All public methods should be named appropriately. Parameters and return types must be shown. Two of the methods that your Queue class must have are "join" and "leave" for joining and leaving a queue.

26. Using C++, write a **minimal** but **complete** template Stack class and a **minimal** but **complete** template Queue class. Do **not** use any STL data structure. You decide and defend what is necessary. Provide separate modular unit tests for each of your classes. The **unit test program should check for normal function and should be able to break** the data structure that has incorrect design or implementation. "Break" can also mean abnormal termination**.**

27. Using C++, write a **minimal** but **complete** template Stack class and a template Queue class. You **must use** the most suitable STL data structures inside each of the classes. You decide and defend what is necessary. Explain what STL data structures are the most suitable for each of your classes. Provide separate modular unit tests for each of your classes. The **unit test program should check for normal function and should be able to break** the data structure that has incorrect design or implementation. "Break" can also mean abnormal termination**.**

28. Explain the concepts of cohesion and coupling. What would a software designer be aiming to achieve in regard to these concepts? Use your own C++ examples to explain what needs to be achieved.

29. What is a memory leak? Using C++ example(s), demonstrate a memory leak.

30. What is a dangling pointer? Using C++ demonstrate how a dangling pointer may arise.

31. Using C++ examples, explain the concepts of deep and shallow copy.

32. If you need to write a destructor, what other methods or operators do you need to write? Explain your answer using C++ code. Why would you need a destructor?

33. If a class has pointer data, what methods (name them) must be provided by the class? Use your own C++ examples to explain why these methods are necessary, and, what these methods do. What happens if these methods are not provided? Use code comments to explain.

34. In OO design, what are the SOLID (S.O.L.I.D) principles? Explain each one and demonstrate how you used these in your lab exercises and assignments. C++ code examples are needed.

35. Explain the Open-Closed Principle. Using C++ example(s) demonstrate the Open-Closed Principle.

36. Explain the Liskov Substitution Principle (LSP). Using UML, show class examples that demonstrate:
    a. A violation of LSP
    b. Non-violation of LSP

    Explain the consequences of violating LSP. Use C++ code to demonstrate.

37. Discuss the various hashing approaches, including the advantages and disadvantages of each approach.

38. What is collision resolution when used in the context of hashing? Describe two collision resolution algorithms.

39. Write an Object-Oriented C++ program to meet the following requirement:

    A data file, *data.txt*, contains rows of weather data sorted by date. The data file may contain duplicate entries. For each row, there is a date (format: dd/mm/yyyy), followed by average ambient air temperature for the day (degrees C), average wind speed (km/h) for the day, average solar radiation (W/m^2) for the day. The weather data is separated by white space. A sample showing the format is shown below. Although only 3 rows are shown in the example, with one duplicate, there could a lot of rows of data representing weather data for many decades. It is also possible that duplicate data is not sorted by date. It can appear anywhere in the data file.

    ```
    05/01/2008  19  10 560
    05/01/2008  19  10 560
    06/01/2008  25  7   800
    ```

    Write a main program that reads the data into your Binary Search Tree (BST). You need to write your own BST as well as any other data structures needed for the program. Duplicate data is not stored in the BST. Once the data is loaded, the user can enter a date using a keyboard and your program prints the weather data on the screen (console).

As there can be a lot of data, searching for a particular date must be efficient. How efficient is you search for data for a specified date? Explain all (not just search) the efficiency considerations that your program is utilising. You may have code that would be comparing values. Are these comparisons efficient?

Explain how your solution is designed to fit the Model-View-Controller (**MVC**) pattern. Your explanation needs to clearly identify which class(es), subroutines (functions, procedures) fit into which part of MVC. You should also explain why they fit into the MVC components.

40. What is the PIMPL idiom? When would you want to use this idiom?

41. Explain “Information Hiding”. Why would you use this idea? Using C++ code examples, demonstrate how information hiding can be implemented.

42. Discuss the relationship between “Information Hiding” and Abstraction. Use UML and C++ examples to illustrate your answer.

43. What is a smart pointer?

44. Using pseudo-code, write an *efficient* algorithm that merges two **pre-sorted** arrays which are provided as input parameters to the algorithm. The merge algorithm creates and returns a new array containing all the values that were in the two original arrays with all values sorted. The algorithm does not modify the original arrays and duplicates are permitted in the resulting array. How would this algorithm change if duplicates are not permitted?

45. Implement the merge algorithm in a C++ template subroutine called *merge*. The template merge routine must be able to merge sorted arrays of any ordered data type.

46. Using C++, write an *efficient* subroutine called merge. This subroutine merges two pre-sorted vectors which are provided as input parameters to the *merge* subroutine. The *merge* subroutine creates and returns a new vector containing all the values that were in the two original vectors with all values sorted. The subroutine does not modify the original vectors and duplicates are permitted in the resulting vector.

    Explain why the algorithm is efficient.

    Provide integer test data to test your merge subroutine. The test data will be stored in the vectors to test your merge routine when a test program is written. You do not write the test program or put the test data into vectors for this question. You need to provide reasons for selecting the test data.

47. What algorithmic strategy does the merge sort algorithm use and how efficient is it (Big-O)? Using pseudo-code, write the merge sort algorithm.

48. Explain the Set data structure. What are the common set operations? Write *efficient*

algorithms for **each** of the common Set operations. What preconditions are needed for the algorithms to be efficient? Explain your answer.

49. Write a test plan for each of the set operations.

50. Using C++, implement each of the set algorithms to perform set operations.

51. Write pseudo-code for an efficient Set union algorithm that enables the following operation given that set1, set2 and set3 are class *Set* objects:

```
set3 = set1.union(set2);
```

    What preconditions are needed for the algorithms to be efficient? Explain your answer.

    Using C++, implement the UNION algorithm as a template function called *union*.

    How would your algorithm and implementation change if the three sets are parameters to *union*, where *union* is now a non-friend, non-member function?

52. Explain the use of the binary search algorithm. What are the pre-conditions of this algorithm and how does the precondition(s) affect the run-time performance (Big-O). Implement the algorithm in C++.

53. Using C++, implement the binary search algorithm as a subroutine. Binary search should be possible for any array of **any ordered data type**. Test your your search routine using an ordered Vector of Date objects. Vector and Date are the same ones you would have created in the labs and assignments.

54. Using diagrams and C++ code, explain the following: value parameter, pointer parameter, reference parameter. If the keyword *const* is used on a parameter, what would be the design intention?

    Examine the following method prototype. What does each of the *const* mean in terms of the design intention or purpose? Use a diagram as appropriate.

```
const int& methodA(const Vector& v, int * const * p) const;
```

55. Given the following prototype of a subroutine, identify and explain using diagrams which part of memory is "read-only". Show which `ptr` or `ptr` de-reference cannot occur on the left hand side of an assignment statement.

```
void f1 (int const * const  * const ptr);
```

56. Using C++ code examples, explain the difference between a base class and an abstract base class. When would you use each of these? Show these classes in UML diagrams.

57. Explain with C++ examples what are pure virtual methods. In what software design situations would you use classes that have pure virtual methods? Defend your use in these situations.

58. Using C++ code examples explain the "Law of Demeter".

59. Using C++ code examples demonstrate violations of the Law of Demeter. For this question you should work that you had completed for the labs and/or assignments. If your labs/assignments did not violate this law, you may use other examples, but you need to declare (for this question) that you have never violated this law.

    Explain why these are violations of this law and the consequences of violation of this law.

60. What is STL?

    STL has a stack data structure. What are the other data structures available in STL? What code do you have to write to be able to use some of the STL data structures? To answer code question, think of what you have write to be able to use the STL std::stack in your main program.

61. Using C++ code examples, explain the comparative advantages/disadvantages of iteration vs recursion.

62. In a recursive algorithm, how is the base(s) and general case related to each other?

63. Explain, using suitable examples, the following terms from algorithm analysis:

    a. Running time of an algorithm
    b. Worst case running time.
    c. Order (rate) of growth of the running time
    d. Asymptotic upper bound

64. Arrange the following Big-O values in ascending order of their growth rates: O(n), O(1), $O(n^3)$, $O(n^2)$, $O(2^n)$, O(2n), O(nlog n), O(log n).

65. Unsorted data is inserted in a vector, linked list and a binary search tree so that each of these data structures has the same data. What are the orders (Big-O) for the search/find algorithms for each of these data structures? Explain your answer.

66. When trying to minimise space and time complexity, you had to store large amounts of data, what deliberations would you carry out when deciding on what data structure(s) to use? What are the trade-offs in your choices?

67. The following terms are encountered in complexity analysis. Explain each term.

    a. NP
    b. P
    c. Deterministic algorithm
    d. Non-deterministic algorithm

68. What are software Design Patterns?

    Explain the *Strategy* Design Pattern using C++ code examples. Describe the utility of this design pattern and what advantages are gained by using it.

Using a demonstration example of your choice, design a complete program using UML that uses the *Strategy* Design Pattern.

Implement your design using C++. Using code comments, highlight where this pattern is being used.

69. Using UML, explain the Model-View-Controller (**MVC**) pattern. Your explanation needs to clearly identify which class(es), subroutines (functions, procedures) fit into which part of MVC. You should also explain why they fit into the MVC components.

    Implement your design as a C++ program and highlight as code comments which parts of the code matches which part(s) of the MVC pattern.

70. There is a rule “Make data members private, except in behaviourless aggregates”. Looking back on your lab and assignments, explain where you broke this rule and where you followed this rule. Use your own code to illustrate your answers. When answering this question highlight where your class design pretended to encapsulate behaviour.

71. Discuss if it would make sense to have a rule “Consider making virtual functions non-public, and public functions nonvirtual”.

72. Explain the Nonvirtual Interface (NVI) pattern.

73. Explain what Cyclomatic complexity is and how it relates to program logic complexity. How would Cyclomatic complexity relate to the number of test cases. Provide C++ code examples to illustrate your answer – use your own work from the labs and assignments.