College of Science – Computer Science

CSC371 – Advanced Object Oriented Programming

May/June 2021

Release Time: **10:00 (Time Zone: BST)**

Deadline: **13:00 (Time Zone: BST)**

Alternative Assessment Information

- You ***MUST*** use your own copy of this assessment from Canvas. If you obtained this assessment document from a friend or elsewhere then delete this copy and use your own version from Canvas. If you experience difficulties to download the assessment, get in contact via the emails below.
- This is an open-book assessment. This means you may use your notes, textbooks, and other resources, including calculators. Copying from resources other than your notes requires referencing.
- You must submit before the deadline. Allow some spare time for technical submission issues.
- The total time for this assessment is 3 hours. This assessment is designed to be sat in a 2-hour window. An additional hour allows you time for accessing the paper, uploading your submission, and dealing with technical issues.
- It is suggested that you use Microsoft Word (or any other editor of your choice) to type your answers, then save as PDF when you are ready to submit. All submitted text (and code if present) must be word-processed, but you may include images (or photos of hand drawn images) as part of the document.
- This is an individual assessment. Under no circumstances are you to discuss any aspect of this assessment with anyone; nor are you allowed to share this document, ideas or solutions with others using email, social media, instant messaging, websites, or any other means. Your attempts at these questions must be entirely your own work. Those found to have collaborated with others will receive a mark of 0.

Special Instructions

Answer all questions.

Submission Instructions

- Please submit a **single PDF** file **named as your student number followed by the module code** (e.g. 123456-CSC789.pdf) via the submission link located on the module page in Canvas.

By submitting, electronically and/or hardcopy, you state that you fully understand and are complying with the university's policy on Academic Integrity and Academic Misconduct. The policy can be found at https://myuni.swansea.ac.uk/academic-life/academic-misconduct.

Originator(s): ***Martin Porcheron***

In case of queries email both: m.a.w.porcheron@swansea.ac.uk and h.elliot@swansea.ac.uk

Part A. Object oriented concepts

1. Explain two advantages of using template metaprogramming techniques in C++ in comparison to other polymorphic techniques.

[4 marks]

2. Explain what *multiple inheritance* is, describing a benefit of it, a commonly cited problem related to it, and how C++ overcomes this problem.

[6 marks]

3. Describe the following principles of object oriented programming and explain how they feature in C++.

(**a**) *Encapsulation*

[4 marks]

(**b**) *Runtime polymorphism*

[4 marks]

Questions continue on next page. . .

Part B. Classes in C++

4. Describe the meaning of the `static` keyword in the context of member variables within classes.

[2 marks]

5. The following questions relate to this sample code:

Exam.h:

```
class Exam {
  const unsigned int total_marks;

public:
  Exam();
  Exam(int total_marks);
  ~Exam();

  int get_total_marks();
  bool reveal_answers() const;
};
```

Exam.cpp:

```
Exam::Exam() {
  total_marks = 50;
}

Exam::Exam(int total_marks) {
  this->total_marks = total_marks;
}

Exam::~Exam() {}

int Exam::get_total_marks() {
  return this->total_marks;
}

bool Exam::reveal_answers() {
  return false;
}
```

(**a**) Identify two issues with the code that will prevent successful compilation. For each issue you identify, state the issue, provide a brief explanation and demonstrate the changes you would make to the code to resolve the issue.

[6 marks]

(**b**) Alice wants to be able to print `Exam` objects to the standard output stream in the format: "`Exam(`*`total_marks`*`)`" where *`total_marks`* is the value of the `total_marks` variable. An example output would be "`Exam(50)`".

Describe the feature they should use to accomplish this and give the code to implement it.

[3 marks]

(**c**) Bob has created a new class named `MockExam` that derives from `Exam` and includes additional member variables. The new `MockExam` class overrides the `reveal_answers` function to return `true`. Bob has not changed the `Exam` class at all.

Describe and explain, in your own words and with code, the changes Bob must make to the `Exam` class to ensure the correct implementations of functions are run and all resources handled appropriately when the `Exam` and `MockExam` classes are used.

Note: You are <u>not</u> being asked to provide any code for the `MockExam` class.

[5 marks]

Part C. Memory management in C and C++

6. Explain what `#pragma pack` is and give code in which `#pragma pack(2)` will have an effect. Show how this effect can be easily demonstrated. Explain why using this may not be ideal in practice.

[5 marks]

7. You are given C++ code which declares a struct called `Employee`. This has three member variables only: a reference to a `std::string` called `name`, a `const unsigned int` called `employeeNum`, and an `unsigned int` called `lineManagerEmployeeNum`.

(**a**) Describe situations in which the *copy constructor* and *copy assignment operator* functions get called. Write a few lines of code that demonstrate these functions being used.

[3 marks]

(**b**) Decide if it is possible to implement an overloaded *move assignment operator* such that should a move assignment be attempted, the original struct will no longer contain any member data.

- If it is possible, provide an implementation for this operator overload function.
- If it is *not* possible, explain why and show with code how you can explicitly flag any attempted use of the operator as a compiler error.

[3 marks]

8. Read the following valid C declaration for a struct:

```
struct Cat {
    struct Cat *mother;
    struct Cat **offspring;
    unsigned int num_offspring;
    unsigned int age;
};
```

Each `Cat` struct has a pointer to its `mother`, plus a pointer, `offspring`, to an array of dynamically allocated pointers, each of which point to a `Cat` struct. Cats can produce multiple offspring at the same time in a 'litter', and can produce multiple litters in their lifetime. Each `Cat` struct contains a variable that states the total number of offspring the cat has produced (`num_offspring`) in its lifetime plus an `unsigned int` of the cat's `age`.

Write the implementation for the following function that produces a litter of cats:
`struct Cat* produce_litter(struct Cat *mother, unsigned int litterSize);`

This function takes a pointer to the `mother` struct and the size of litter to produce (`litterSize`), and returns a pointer to the array of `Cat` structs created. You must allocate space for enough `Cat` structs for the size of the litter. For each new `Cat` you create, you must correctly set the `mother` variable to point to the `mother` struct. The age must be set to `0`. You must also update the `offspring` variable in `mother` to include a pointer to each of your new `Cat` structs, taking care not to delete or leak any existing data.

[5 marks]

End of Paper