THE UNIVERSITY OF WESTERN ONTARIO
LONDON CANADA

COMPUTER SCIENCE 3307A FINAL EXAMINATION DECEMBER 10, 2019 3 HOURS

NAME: ______________________________

STUDENT NUMBER: ______________________

Question	
1-20.	________
21-22.	________
23.	________
24.	________
25.	________
26.	________
27.	________
28.	________
29.	________
30.	________
31.	________
32.	________
33.	________
34.	________
35.	________
36.	________
TOTAL	________

(Out of 180 marks)

There are no cheat sheets, books, or other reference materials allowed for this exam. No calculators or other electronic devices are permitted either.

Part I -- Multiple Choice, True/False -- Choose the best answer from the choices given. Circle your answer on the paper. [40 marks total, 2 marks each]

1. In C++, a non-null pointer always "points to" data allocated on the heap.
   a. True.
   b. False.

2. If a variable `i` is declared as:

   ```
   const int i = 0;
   ```

   then the value of `i` cannot be changed in subsequent code.
   a. True.
   b. False.

3. The syntax `const int *bob` indicates that `bob` is a:
   a. Regular pointer to regular int type.
   b. Constant pointer to regular int type.
   c. Regular pointer to constant int type.
   d. Constant pointer to constant int type.

4. Under what circumstances should you declare a function as `const`?
   a. When the function modifies only the object's member data.
   b. When you want to call the function on references to objects of that class.
   c. When the function does not modify `*this`.
   d. When the function creates no local variables.
   e. None of the above.

5. Space is always allocated on the stack when:
   a. `new` is invoked.
   b. An assignment statement is executed.
   c. A function is called.
   d. A local object variable is modified.
   e. None of the above.

6. Space is always allocated on the heap when:
   a. `new` is invoked.
   b. A local object variable is modified.
   c. An object parameter is passed by reference.
   d. A constructor is called.
   e. None of the above.

7. Consider the following C++ code snippet:

```
int i = 4;
int j = 5;

int* k = &i;
int& l = j;

*k = 9;
l = 44;

cout << i << " ";
cout << j << " ";
cout << *k << " ";
cout << l << endl;
```

What will be the output of the snippet? Assume `cout` and `endl` have been imported.

   a. `4 5 9 44`
   b. `4 5 4 5`
   c. `9 5 9 44`
   d. `9 44 9 44`
   e. The snippet will not compile.

8. Consider the following function:

```
template <class T>
T average(T *atArray, int nNumValues) {
        T tSum = 0;
        for (int nCount=0; nCount < nNumValues; nCount++)
           { tSum += atArray[nCount]; }
        tSum = tSum / nNumValues;
        return tSum;
}
```

Which of the following statements about the class/type `T` must be true in order for the code to compile and run without crashing?

   a. It must be some kind of numeric type.
   b. It must have the < operator defined.
   c. It must have the `[]` access operator defined.
   d. All of the above.
   e. None of the above.

9. In C++, the destructor for an object is always called when the object is destroyed.
   a. True.
   b. False.

10. In C++, only operations that are declared to be virtual can be overridden in derived classes.
    a. True.
    b. False.

11. All member functions in a C++ class have a `this` pointer.
    a. True.
    b. False.

12. The story points of a user story:
    a. Have a work-hours equivalent defined specifically in the user story specifications.
    b. Cannot change once set.
    c. Are used to indicate the approximate size and complexity of a story.
    d. Are estimated by the customer.
    e. None of the above.

13. In a UML class diagram, there can never be more than one association or relationship between any two classes in the diagram.
    a. True.
    b. False.

14. Which of the following statements is false?
    a. Design principles define guidelines for "good" code.
    b. Extensibility is a trait of well-designed software.
    c. In specific coding situations, design patterns should be applied as they follow good design and make code more readable to other developers.
    d. Low coupling and high cohesion can go together.
    e. All of the above statements are true, not false.

15. Why does the Interface Segregation Principle aim to avoid fat interfaces?
    a. High level modules should not depend on lower level modules, because higher level interfaces are less likely to change.
    b. Because a fat interface prevents derived classes from being used as if they were the interface class.
    c. While fat interfaces reduce coupling between the subclasses and the interface, the large interface becomes difficult to use.
    d. Any class inheriting the interface must provide an implementation for all interface functions, which may be empty or meaningless.
    e. None of the above.

16. Which of the following are true about a successful implementation of the Singleton design pattern?
    a. It relies on public constructors.
    b. It must use the keyword `static`.
    c. It ensures that exactly one copy of the object always exists during execution.
    d. It prohibits refinement through subclassing.
    e. None of the above.

17. Which of the following statements about the Decorator pattern are true?
    a. It can be used to add and remove behaviour dynamically at run time.
    b. It is still viewed logically as a single unit, regardless of the number of decorator layers applied.
    c. It works by wrapping one object inside another with the same interface.
    d. All of the above.
    e. None of the above.

18. Creational design patterns:
    a. Are all based on some sort of factory pattern.
    b. Are used to encapsulate the creation of objects.
    c. Are only useful when constructors are statically dispatched.
    d. All of the above.
    e. None of the above.

19. The Adapter pattern:
    a. Can be implemented in C++ using multiple inheritance.
    b. Can be implemented in C++ without using multiple inheritance.
    c. Uses an adapter object to translate requests to an existing interfaces.
    d. Both a) and c) are true.
    e. All of the above are true.

20. The Builder pattern is used to:
    a. Create objects by merging classes with each other.
    b. Control the structure of complex, hierarchical class relationships, by building up from the bottom.
    c. Control the creation of objects where there are many constructors with long and complex parameter lists.
    d. Create objects through a similar process of steps, but allow the actual representation to vary.
    e. None of the above.

Part II -- Fill In The Blank -- Fill in the blanks as instructed. [34 marks total]

21. Consider the following class declarations [8 marks total, 1 mark per line]:

```
 1 class Person
 2 {
 3   void sleep();        // 1
 4   virtual void walk(); // 2
 5   virtual void eat();  // 3
 6 };
 7
 8 class Student : public Person
 9 {
10   void sleep();         // 4
11   void walk();          // 5
12   virtual void study(); // 6
13 };
14
15 int main()
16 {
17   Person* p1 = new Student();
18   Student s1;
19   Person p2 = s1;
20
21   // insert line here
22 }
```

For each of the following, indicate the method called (by its number in the comments above) if we were to replace line 21 with the line given. If the line would not compile, indicate WNC.

`p1->sleep();` ____

`p1->walk();` ____

`p1->eat();` ____

`p1->study();` ____

`s1.sleep();` ____

`s1.eat();` ____

`p2.sleep();` ____

`p2.walk();` ____

22. Given the following class definitions [26 marks total, 1 mark per line]:

```
#include<iostream>
using namespace std;

class X {
 public:
  virtual void a() { cout << "X::a"; }
  void b() { cout << "X::b"; }
  virtual void c() { cout << "X::c"; }
  virtual void c(int r) { cout<<"X::c(r)"; }
  void d() { cout << "X::d"; }
};
```

```
class Y : public X {
 public:
  void a() { cout << "Y::a"; }
  void b() { cout << "Y::b"; }
  virtual void d() {cout<<"Y::d";}
  void e() { cout << "Y::e"; }
};

class Z : public Y {
 public:
  void c() { cout << "Z::c"; }
};
```

Comment on any errors in these statements (and show output where requested).

```
X * xx = new X;    ____________________
X * xy = new Y;    ____________________
X * xz = new Z;    ____________________
Y * yx = new X;    ____________________
Y * yy = new Y;    ____________________
Y * yz = new Z;    ____________________
Z * zx = new X;    ____________________
Z * zz = new Z;    ____________________

xx->a();      Output: ____________________
xx->b();      Output: ____________________
xx->c();      Output: ____________________

xy->a();      Output: ____________________
xy->b();      Output: ____________________
xy->c();      Output: ____________________
xy->c(10);    Output: ____________________
xy->d();      Output: ____________________

xz->a();      Output: ____________________
xz->b();      Output: ____________________
xz->c();      Output: ____________________

yy->a();      Output: ____________________
yy->c();      Output: ____________________

yz->a();      Output: ____________________
yz->c();      Output: ____________________
yz->c(20);    Output: ____________________

zz->a();      Output: ____________________
zz->c();      Output: ____________________
```

Part III -- Short Answer -- For the following questions, write your answer in the space provided, using diagrams as appropriate. You do not need to write full sentences, and can use point form if that is your preference. [48 marks total]

23. There is a considerable memory leak in your software system. You suspect after hours of debugging that something is wrong with the following object usage. Here, `TimedSession` inherits from `Session` (i.e. your code contains the declaration: `class TimedSession: public Session`).

```
Session * currentSession = new TimedSession(2000);
...
delete currentSession;
```

Is it possible, when using the object in this way, for it to cause a memory leak? If so, what is the likely cause and why would that cause the leak? If not, why is it not possible? [6 marks]

24. Default parameters in C++ allow functions to be called without providing one or more trailing parameters, which can be convenient to software developers. Default parameters, however, are not without issues. Provide a scenario in which the use of default parameters leads to ambiguity when overloading a function in C++. [6 marks]

25. A regular polygon is a shape with any number of sides, where all sides are the same length and all angles are equal in degrees. Consider this class declaration:

```cpp
class RegularPolygon {
    public:
        RegularPolygon();
        RegularPolygon(int numSides, int sideLength);
        virtual ~RegularPolygon();
        virtual int numberOfSides();
        virtual int area();
    private:
        double interiorAngle;
        int sideLength;
}
```

Suppose you created a subclass of RegularPolygon called Square, overriding the above methods as necessary and providing a new constructor that takes a single parameter, the side length for the square in question. Would this scenario violate the Liskov Substitution Principle as we did in class when we derived a Square class from a Rectangle class? Give a basic description of the Liskov Substitution Principle and explain why there is or is not a violation in this case. [6 marks]

26. For memory management, does C++ have built-in garbage collection? Provide one benefit and one drawback of C++'s approach to memory management. [6 marks]

27. What type of error might you get if you omitted the include guard (example below) in your C++ header files? Why might such an error occur? [4 marks]

```
#ifndef SOMESOURCEFILE_H
#define SOMESOURCEFILE_H
...//header content
#endif
```

28. Why should you never place a `using namespace` directive inside of a header file? Why do some C++ programmers argue that you should avoid making use of `using namespace` directives altogether, even in code files? [4 marks]

29. For each of the following user stories, indicate whether or not the story is acceptable according to the INVEST criteria we studied in class. If not, indicate which criteria the story violates. Briefly justify your answer. [6 marks total]
    a. *An agent should be able to manage tickets.* [3 marks]

    b. *The GUI must be aesthetically pleasing to the user.* [3 marks]

30. Why might we want to pass a parameter as a constant reference? For example: `void myfunct(const MyClass &c)` [4 marks]

31. Describe high coupling, using at least one example or symptom of high coupling.
[3 marks]

32. Describe low coupling, using at least one example or symptom of low coupling.
[3 marks]

Part IV -- Long Answer -- For the following questions, write your answer in the space provided, using diagrams as appropriate. Again, you do not need to write full sentences, and can use point form if that is your preference. [58 marks total]

33. Recall the file utilities created in the individual assignment earlier in this course. This assignment entailed the creation of a class to manipulate files and directories in the file system, as well as a set of utilities using this class that could replace a number of standard command line file utilities. [20 marks total]

a. Craft a user story capturing one of the requirements for this assignment. Use the front-of-card and back-of-card areas below to capture and record all of the requisite information for this user story. [8 marks]

Front of Card

Back of Card

b. In class, we identified one software pattern that would have been of particular use to this assignment. Which pattern did we identify? How would the use of this pattern improve the design and quality of this software? [8 marks]

c. Provide a UML class diagram, like those given in class accompanying our design pattern examples, to illustrate the use of and integration of the design pattern from part b) above into the assignment. [4 marks]

34. Consider multiple inheritance in C++. [12 marks total]

a. Not every object-oriented language supports multiple inheritance, but C++ does. What are the advantages of supporting multiple inheritance? What are the drawbacks or potential problems with multiple inheritance? [6 marks]

b. In class, we used multiple inheritance in a number of our Structural patterns, with a mixture of public and private inheritance. For example, in our discussion of the Bridge pattern, we created a set list implementation that was declared as follows:

```
template <class T>
class SetListImpl : public ListImpl<T>, private std::set<T> {
   …
};
```

In this case, `ListImpl` was an implementor in the pattern, `SetListImpl` was a concrete implementor, and `set` was a standard container class used internally within `SetListImpl` as its underlying data structure. Why did we use a mixture of public and private inheritance? What purpose can such a mixture serve in general in the creation of C++ classes with multiple inheritance? [6 marks]

35. Compare and contrast the State and Strategy design patterns. Draw a UML diagram outlining each pattern, and briefly describe the context in which each are used. Include some additional points comparing the two patterns, and describe in what situation you would use each over the other. [14 marks]

36. Assume that you are using some class A, declared in a file called A.h. Consider the following code that makes use of this class.

```
#include "A.h"

int main (int argc, char*argv[]){
     A * p_a1 = new A();
     A * p_a2 = p_a1;
     delete p_a1;
     p_a1 = 0;
     delete p_a2;
}
```

Create a memory diagram depicting the execution of the above code step-by-step. Using your diagram, identify and explain in detail the memory management issue(s) that can be found in this code, if any. If there are no such issues present, explain why this is the case in detail. [12 marks]

This page has been left intentionally blank. Use it as additional workspace or extra space for answers if necessary.