Name _____

CISC 3142 – Programming Paradigms in C++
Spring 2019
Exam 1

Part I. Answer all questions in the boxes to the left. (15 points)

D

1. Having individual source files that can be compiled to object files which can then be linked into an executable
a. improves the organization of large applications
b. minimizes the time necessary to rebuild an application's executable
c. allows code to be easily reused in several applications
d. all of the above
e. none of the above

B

2. After modifying the contents of a `.h` file
a. all the source files of the entire application must be recompiled
b. those source files that include the `.h` file must be recompiled
c. only the corresponding .cpp file must be recompiled
d. no recompilation is necessary, only linking need be performed
e. none of the above

B

3. Which is the usual situation for the members of a class?
a. data members are public and member functions are private
b. data members are private and member functions are public
c. all members are public
d. there are both public and private data members and member functions
e. it doesn't make much difference

D

4. Given a member function `mf` of class `C` and the declaration `C c1;`, which is the proper function call?
a. `C::mf(c1)`
b. `C.mf(c1)`
c. `mf(c1)`
d. `c1.mf()`
e. None of the above

A

5. Given the member initialization list `: name(n), balance(0) {}`
a. `name` and `balance` are data members; `n` is a constructor parameter
b. `name` and `balance` are constructor parameters; `n` is a data member
c. `name`, `n`, and `balance` are all constructor parameters
d. `name`, `n` and `balance` are all data members
e. any of the above is possible

B

6. Objects (variables declared to be a class type) are typically passed to a function
   a. by value
   b. by reference or const reference
   c. by value or reference
   d. by value or const reference
   e. it makes no difference how they're passed

D

7. To indicate that a member function does not modify the receiver
   a. there's no way to do this—the receiver is not present in the function's signature
   b. the keyword `const` is placed at the beginning of the function header
   c. the class is declared `const`
   d. the keyword `const` is placed after the member function's parameter list
   e. a member initialization list must be used

E

8. Given an integer variable `x`, for which of the following call/signature fragments might the value of `x` be changed upon returning from the function?
   a. `f(x)` with header `void f(int val)`
   b. `f(x)` with header `void f(int &val)`
   c. `f(&x)` with header `void f(int *valp)`
   d. *a* and *b*
   e. *b* and *c*

C

9. The insertion/output operator << for a class `C` is typically written as
   a. a member function of `C`
   b. a member function of `cout`'s class
   c. a non-member function declared as a friend
   d. all of the above are equally likely
   e. none of the above

C

10. The return type and value for the << operator
   a. is a reference to the object being printed to allow for the chaining of << calls
   b. is the number of characters printed
   c. is a reference to the output stream to allow for the chaining of << calls
   d. must be a friend of the class being printed
   e. << is a void function – it does not return a value

E

11. Given the declaration `C c1, c2;`, the operator + in `c1 + c2` could be
   a. a member function of `C` with one parameter
   b. a member function of `C` with two parameters
   c. a non-member function with two parameters
   d. *a* or *b*
   e. *a* or *c*

A

12. Given `C c1, c2;`, the operator `+=` in `c1 += c2` would typically be
   a. a member function of `C` with one parameter
   b. a member function of `C` with two parameters
   c. a non-member function with two parameters
   d. any of the above
   e. none of the above

B

13. When returning the receiver as the return value of a member function, one writes:
   a. `return &this;`
   b. `return *this;`
   c. `return this`
   d. `return this&`

B

14. The code `return os;` where `os` is an `ostream` is probably in a
   a. `>>` operator
   b. `<<` operator
   c. `[]` operator
   d. `+` operator
   e. `+=` operator

A

15. Which of the following is true?
   a. the effect of using default parameter can always be achieved by having overloading method/functions
   b. the effect of having overloaded methods/functions can always be achieved by using of default parameters
   c. both *a* & *b* are true
   d. overloading and default parameters have nothing to do with each other

Part II. Answer all of the following.

16. You are to write a module `average.h/.cpp` containing a single function `avg` that accepts an array of integers, and an integer containing the number of elements; and returns the average as an **integer**.

a. Code the `.h` file. Please make sure you include the preprocessor code(include guard) to prevent multiple inclusions of the file. **(5 points)**

```
#ifndef AVERAGE_H
#define AVERAGE_H

int avg(int arr[], int n);  // or whatever function you wrote

#endif
```

b. Code the `.cpp` file. Please make sure you include all necessary `#includes`. **(5 points)**

```
#include "average.h"

int avg(int arr[], int n) {      // or whatever function
     int total = 0;
     for (int i = 0; i < n; i++)
          total += arr[i];
     return total / n;
}
```

c. Code the file `averageDemo.cpp` containing a `main` function that declares and initializes an array, and prints out the average obtained by calling `avg`. Please make sure to include all relevant #includes. **(5 points)**

```
#include <iostream>

#include "average.h"

using namespace std;

int main() {
     int a[5] = {10, 20, 30, 40, 50};
     cout << avg(a, 5) << endl;        // or whatever function

     return 0;
}
```

d. You now realize that the function should return a double rather than an integer. Which files need to be modified, and which file(s) **must** be recompiled? (No need to make the changes) **(2.5 points)**

The signatures of the declaration and definition of the function must be changed to have a return type of double. (The declaration is the function header in average.h and the definition is the implementation of the function, i.e., the function body in average.cpp). average.cpp must therefore be recompiled; in addition, since the header (.h) file is changing as well, all code #includ'ing that file must be recompiled as well, and thus averageDemo.cpp must be recompiled.

e. Finally, you probably did not check for an empty array (i.e., a 0 number of elements, which would result in a division by zero), and decide you should add that code. Once again, which file(s) need to be modified and which **must** be recompiled? (Again, no need to make the changes) **(2.5 points)**

A conditional should be added to the body of the function to test for n == 0 (and throws the exception or exit the program if the condition is true), and thus average.cpp (which contains the function body) must be recompiled. Since the .h file is not modified by this change, files #includ'ing average.h (in particular averageDemo.cpp) need not be recompiled

17. You are to define a class `ClockInteger` that is similar to the Integer class presented/assigned in class, but the value of the class is restricted to the range 1-12. The class should contain the following:

- An integer data member to hold the value
- A constructor that accepts an integer value, and *normalizes* it to a value between 1 and 12: 13 for example would become 1, 14 would become 2, 0 would become 12. etc. If no value is sent to the constructor, the value should be initialized to 1.
- `+`, `+=`, `<<`, `>>`, and `==` operators (you should follow the usual conventions for parameter passing, return types, and member/non-member).
- A `get` function that returns the value

a. Code the `clock_integer.h` file (you can skip the include guard). Please do not make any of the functions inline (**5 points**)

```
#ifndef CLOCKINTEGER_H
#define CLOSKINTEGER_H

#include <iostream>

class ClockInteger {
     friend ClockInteger operator +(const ClockInteger &c1, const ClockInteger &c2);
     friend bool operator ==(const ClockInteger &c1, const ClockInteger &c2);

     friend std::ostream &operator <<(std::ostream &os, const ClockInteger &c1);
     friend std::istream &operator >>(std::istream &os, ClockInteger &c1);
public:
     ClockInteger(int v = 1);

     ClockInteger &operator +=(const ClockInteger other);
     int get() const;
private:
     int val;
};
#endif
```

~

b. (**10 points**) Now provide the `clock_integer.cpp` file. For full credit, you must use the `+=` operator in your definition of your `+` operator.

```
#include <iostream>

#include "ClockInteger.h"

using namespace std;

void normalize(int &val) {
   val = val == 0 ? 12 : val > 0 ? (val-1) % 12 + 1 : 12 + val % 12;
}

ClockInteger::ClockInteger(int v) : val(v) {normalize(val);}

int ClockInteger::get() const {return val;}

ClockInteger &ClockInteger::operator +=(const ClockInteger other) {
     normalize(val += other.val);
     return *this;
}

ClockInteger operator +(const ClockInteger &c1, const ClockInteger &c2) {
   return ClockInteger(c1) += c2;
}

bool operator ==(const ClockInteger &c1, const ClockInteger &c2) {
   return c1.val == c2.val;
}

ostream &operator <<(ostream &os, const ClockInteger &c) {
     os << c.val;
     return os;
}

istream &operator >>(istream &is, ClockInteger &c) {
      is >> c.val;
      normalize(c.val);
      return is;
}
```