Signature _____

Instructions: (Read carefully before the exam begins):

1. Before you begin, make certain that you have one Exam Booklet with pages numbered 1–18 printed double-sided.
2. This is a **closed-book exam.** The only permitted aid is the provided *Supplemental Materials* sheet.
3. There are a total of 12 questions, for a total of 117 marks.
4. The marks assigned to each question are shown at the beginning of the question; use this information to organize your time effectively.
5. Place all your answers in the spaces provided on these pages. There are 2 extra sheets at the end of the exam. **Do not write answers on the Supplemental Materials Sheet.** It will not be marked.
6. You do not need to write comments in your code unless it is specifically required by the question.
7. Questions will not be interpreted. Proctors will only confirm or deny errors in the questions. If you consider the wording of a question to be ambiguous, state your assumptions clearly and proceed to answer the question to the best of your ability. You may not trivialize the problem in your assumptions.
8. Cheating is an academic offense. Your signature on this exam indicates that you understand and agree to the University's policies regarding cheating on exams.

Question	out of	mark	marker's initials
1	8		
2	10		
3	9		
4	10		
5	7		
6	15		
7	6		
8	10		
9	13		
10	13		
11	6		
12	10		
Total	117		

Question 1 [8 marks]

a) [1 mark] What is a *scope qualifier* used for?

b) [1 mark] How would you declare a class A to prohibit the copying of objects of type A?

c) [6 marks] Consider the following global declarations.

```
class A {
  friend class C;
  class B {
     int i;
  };
protected:
  B ab;
public:
  void afunc();
};
```

```
class C {
public:
  void cfunc(A &a);
};

class D : public A {
public:
  void dfunc(A &a);
}

A &a = new D;
```

State whether each of the statements in the leftmost column of the table below is a *valid* or *invalid* code statement within each of the scopes listed at the heads of the other columns in the table.

statement	In main()	In A::afunc()	In C::cfunc()	In D::cfunc()
`B b;`				
`a.ab.i = 10;`				
`a.dfunc();`				

Question 2 [10 marks]

a) [1 mark] Give a variable declaration of type constant pointer to an integer (i.e., the pointer is constant, the integer is not).

b) [2 marks] Consider a function declaration:

```
void func (Object &o);
```

If the function does not modify the argument `o`, what is the disadvantage of *not* declaring `o` to be constant?

c) [2 marks] Explain why streaming operators `operator<<` and `operator>>` should *not* be member functions.

d) [5 marks] Rewrite the following class declaration so that the *implementation is hidden*. Your answer should be complete.

```
class Year {
    int year_;
public:
    Year (int);
    int year();
};
```

Question 3 [9 marks]

a) [2 marks] Amend the following declarations to break the dependency cycle

```
#ifndef A_H
#define A_H

class A{
   B *b;
};

#endif
```

```
#ifndef B_H
#define B_H

class B {
   A *a;
};

#endif
```

b) [2 marks] Write a *makefile rule* that (1) compiles the source code file (`A.cpp`) for the class `A` from question 3a, to produce an object file (`A.o`), but (2) compiles the file only if none of the files it depends on has changed since the last time the file was compiled. Assume only the above declarations.

c) [2 marks] Explain why one should never place a using directive inside of a header file.

d) [3 marks] Show three ways to access variable `i` in namespace `N`.

Question 4 [10 marks]

a) [3 marks] Consider the following declarations.

```
class A {
     B b;
public:
     B& afunc(C& c);
};
```

```
class B {
public:
     C& bfunc1();
     B& bfunc2();
     void bfunc3();
};
class C{
public:
     void cfunc();
};
```

The following are statements inside the implementation of `afunc(C&)`. For each, state whether it violates the *Law of Demeter*

i. `b.bfunc1.cfunc();`

ii. `b.bfunc2.bfunc3();`

iii. `c.cfunc();`

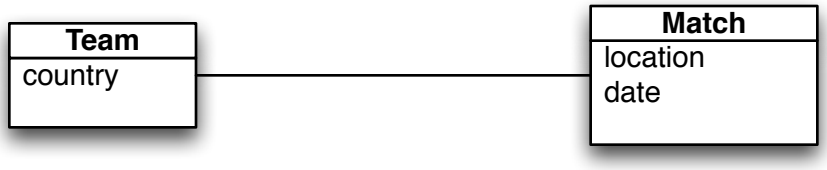
b) [2 marks] How does one test a module that depends on other modules that have not yet been implemented?

c) [1 mark] What precisely is the purpose of *boundary testing*?

d) [4 marks] Annotate the following UML model, so that

i. the model specifies that each team plays in 3 matches, and that each match is a competition between 2 teams

ii. the model records the score of each team in each match



Question 5 [7 marks]

a) [5 marks] Consider the following declarations.

```
class Base {
public:
    void func();
    virtual void vfunc();
    virtual void vfunc2();
};
class Derived : public Base {
public:
    void func();
    void vfunc2();
    virtual void vfunc3();
};
Base &bPtr = new Derived;
```

The following are statements, state which function is invoked.

i. `bPtr.func();`

ii. `bPtr.vfunc();`

iii. `bPtr.vfunc2();`

iv. `bPtr.Base::vfunc();`

v. `((Derived &) bPtr).vfunc3();`

b) [2 marks] Draw the vtable for object `*bPtr` from question 5a.

Question 6 [15 marks]

Design an ADT for a Postal Code data type for use in Canadian postal addresses. Assume that the set of valid values is any string of six characters that starts with a capitalized alphabetic character, that alternates between capitalized alphabetic characters and digits, and that has a space between the third and fourth characters (e.g., N2L 3G1 is the postal code for UW). (Real Canadian postal codes avoid certain letters that can be confused with others -- we'll ignore this detail.)

Your ADT should be useful in any program that maintains Canadian addresses, and that wants a special data type for storing postal codes.

Your answer to this question is in several parts:

a) Provide a complete class declaration of your ADT, including nonmember functions, `const`, and default arguments.

b) State whether objects of your Postal Code ADT are mutable or immutable, and explain (briefly) the rationale of your decision.

c) If you are implementing your own default constructor, state your choice of default initial value for Postal Code objects and explain why. If you are not implementing your own default constructor, explain why. In either case, restrict your answer to 1 sentence.

d) Explain briefly your rationale for making the member functions in your ADT member functions, and your rationale for making nonmember functions in your ADT nonmember functions.

e) Which functions check that the value of a Postal Code object is valid?

Question 6 (cont.)

Questions 7-12 make use of the UML model on the Supplemental Sheet

On the supplemental sheet, there is a possibly incomplete UML model for a composite object Recipe.

A recipe consists of a list ingredients, whose amounts are specified either as a number (e.g., 2 apples) or as an amount of some unit of measure (e.g., 1/4 teaspoon salt); it also includes a set of instructions. Some ingredients are augmented with a description of how the ingredient should be prepared before starting the recipe (e.g., 2 onions, *chopped*); these are modelled as Prepared Ingredient objects. In our model, the ingredients of a recipe are represented by a collection of objects, and the instructions are stored in a single string value.

There are operations for adding ingredients, removing ingredients, and tweaking (i.e. modifying) ingredients in a recipe. There are also operations for printing the recipe and for printing a shopping list of ingredient names. For example, the following is a printed recipe. For demonstration purposes only, we use fonts to denote where each printed word comes from: ingredient names are in **bold** font, amounts are underlined, and the description of how an ingredient should be prepared is in *italics*:

Recipe Name: Olive Tapenade
Ingredients:
20 **Kalamata olives**, *coarsely chopped*
1 Tbsp **capers**, *rinsed, drained, and chopped*
1 tsp **fresh lemon juice**
2 tsp **olive oil**
0.5 tsp **anchovy paste**
Instructions:
Combine Kalamata olives, capers, lemon juice, olive oil, anchovy paste, and pepper. Mix well.

The following is the printed shopping list for this same recipe:

Kalamata olives
capers
fresh lemon juice
olive oil
anchovy paste

Note that some of the operations (e.g., Recipe::addIngredient) have default arguments.

Questions 7 through 12 ask you to implement code fragments of the composite object Recipe. Your answers to these questions should agree with one another (e.g., they should compile, link, and work together).

Question 7 [6 marks]

Below is a partial C++ class declaration for the Recipe class.

a) Complete the class declaration with any additional private/protected members that you need to answer part (b)

b) Provide the C++ *definition* of the operation `printRecipe()` for the Recipe class. The output should resemble the example recipe on page 3. In particular, the output should include the recipe name, the list of ingredients. one per line, and the instructions. (White space and punctuation is not important.)

```
using std::string;

class Recipe {
public:
   Recipe();
   ~Recipe();
   Recipe& Recipe(const Recipe&);
   Recipe& operator= (const Recipe&);
   void addIngredient ( const string name, float amount, const string unit = "",
         const string preparation = "" );
   void removeIngredient ( const string name );
   void tweakIngredient ( const string name, float amount );
   void tweakIngredient ( const string name, const string preparation);
   void addInstructions ( const string instructions);
   void printRecipe ( std::iostream& ) const;
   void printShoppingList ( std::iostream& ) const;


private:
   string name;
   string instructions;
// ADD YOUR MEMBERS HERE




};
```

Question 8 [10 marks]

Provide the C++ *class declaration* and *definition* for the Food class. The class should declare and define only the members that are specified in the UML diagram.

- Food must be declared to be an *abstract base class*.
- `print()` prints the object. It must be declared so that it is overridable by derived classes.
- `printName()` prints the name of the object. It must be declared so that it is *not* overridable - it will be used to print the shopping list.

Assume that methods `printName()` and `print()` both have return types of `void`. If you need to invent variable names, make reasonable choices for names.

Question 9 [13 marks]

a) Declare the data members of the C++ class declaration for the Ingredient class. Your declarations must adhere to the class design given in the UML model. If you need to invent variable names, make reasonable choices.

b) Provide the C++ *definition* of the operation `print()` for the Ingredient class. Assume that the method has a return type of `void`. Your declarations must adhere to the class design given in the UML model. If you need to invent variable names, make reasonable choices.

c) Provide the C++ *definition* of the operation `print()` for the Prepared_Ingredient class. Your declarations must adhere to the class design given in the UML model. Assume that the method has a return type of `void`. If you need to invent variable names, make reasonable choices.

Question 10 [13 marks]

a) Provide the C++ *definition* of the constructor for the Ingredient class. If you need to invent variable names, make reasonable choices.

b) Provide the C++ *definition* of the assignment `operator=` for the Ingredient class. Your operator should adhere to *deep copy* semantics. If you need to invent variable names, make reasonable choices.

Question 11 [6 marks]

There are methods that are so important that the compiler will generate them for a class if the developer of the class does not indicate that he / she is providing them. Consider these methods in the Prepared_Ingredient class.

- Given the class design *as specified in the UML diagram*, state which methods (if any) would be generated for the Prepared_Ingredient class.
- For each method that would be generated by the compiler, describe briefly the generated method. (To receive any credit, your answer must be specific to a Prepared_Ingredient object.)
- For each method that would not be generated, state whether the developer should create one himself / herself, and explain briefly why or why not.

Question 12 [10 marks]

a) State whether an Ingredient object is substitutable for an Food object. Use the three rules of the *Liskov Substitutability Principle* to defend your answer. (Credit is awarded only for justification, not for correctly determining substitutability.)

Question 12 (cont.)

b) Consider the design principles regarding whether to use *Inheritance vs. Composition* when defining a new class in terms of an existing class, with respect to the relationship between class Ingredient and class Food.

- Use the principles to explain why, given the current design, it might be better if Food were a component of Ingredient (composition)
- Draw a UML diagram that shows the two classes in a *composition* relationship. Be sure to include the appropriate association type and multiplicities.
- Provide the C++ *class declaration* for the modified Ingredient class. If you need to invent variable names, make reasonable choices for names.

Extra Sheet #1

Extra Sheet #2