Name: _____

Score: _____ / _____

HW3

Part 1

1

In the following code, identify the value of the specified variable at the Q* lines when the code is executing. (Therefore the value of each variable at the line marked Q1, is the value after that such line have been executed by the processor.)

```
#include <iostream>

int main() {
  int i = 7;
  int j = 8;
  int* p = &i; // Q1
  int& r = i;
  int& rpr = (*p); // Q2
  i = 20;
  p = &j; // Q3
  rpr = 13; // Q4

  r = 8;
  std::cout
    << "i=" << i << " j=" << j << std::endl; // Q5
}
```

- 1 *p: ____
- 2 *p: ____
- 3 *p: ____
- 4 i: ____
- 5 i: ____ j: ____

Answer Point Value: 10.0 points
Answer Key: 7, 7, 8, 13, 8, 8
Correct Feedback: --------
Incorrect Feedback: --------

2

In the following code, identify the value of the specified variable at the Q* lines when the code is executing. (Therefore the value of each variable at the line marked Q1, is the value after that such line have been executed by the processor.)

```
#include <iostream>

int main() {
  int i = 11;
  int j = 3;
  int* p = &i; // Q1
  int& r = i;
  int& rpr = (*p); // Q2
  i = 40;
  p = &j; // Q3
  rpr = 27; // Q4

  r = 99;
  std::cout
    << "i=" << i << " j=" << j << std::endl; // Q5
}
```

- 1 *p: ____
- 2 *p: ____
- 3 *p: ____
- 4 i: ____
- 5 i: ____ j: ____

Answer Point Value: 10.0 points
Answer Key: 11, 11, 3, 40, 99, 3
Correct Feedback: --------
Incorrect Feedback: --------

3 Does a static class variable member differ from a non-static class variable member?

- A. No. They are the same — *Feedback: --------*
- B. They differ. A static class variable member can be accessed from each class instance. A non-static class variable member can only be accessed from a specific instance; therefore the variable is unique per instance. — *Feedback: --------*
- C. They differ. A non-static class variable member can be accessed from each class instance. A static class variable member can only be accessed from a specific instance; therefore the variable is unique per instance. — *Feedback: --------*

Answer Point Value: 10.0 points
Answer Key: B
Correct Feedback: --------
Incorrect Feedback: --------

4 Explain the differences between static and non-static class variable members

!!!!Answer Point Value: 10.0 points
Model Short Answer: --------
Feedback: --------

5

FridgeClass is a class that have been developed by your collegue. You now have to use the FridgeClass in your program, therefore you are going to allocate an instance of this class calling the class constructor FridgeClass::FridgeClass(param1, param2). What is the main difference between a) and b) ?
a) FridgeClass fc(param1, param2);
b) FridgeClass *pfc = new FridgeClass(param1, param2);

!!!!Answer Point Value: 10.0 points
Model Short Answer: --------
Feedback: --------

6

FridgeClass is a class that have been developed by your collegue. You now have to use the FridgeClass in your program, therefore you are going to allocate an instance of this class calling the class constructor FridgeClass::FridgeClass(param1, param2).
a) FridgeClass fc(param1, param2);
b) FridgeClass *pfc = new FridgeClass(param1, param2);

- A. a) is an allocation on the heap therefore the class will disappear when the function in which it is declared returns; b) is an allocation on the stack, it is not deleted when the function that allocate the object returns — *Feedback: --------*
- B. a) is an allocation on the heap therefore the class will disappear when the function in which it is declared returns; b) is an allocation on the stack therefore it should be deleted by the user when he/she is not using it anymore — *Feedback: --------*
- C. a) is an allocation on the stack therefore the class will disappear when the function in which it is declared returns; b) is an allocation on the heap therefore it should be deleted by the user when he/she is not using it anymore — *Feedback: --------*
- D. a) is an allocation on the heap, when the function returns it is not automatically deleted b) is an allocation on the stack, it is automatically deleted when the function returns — *Feedback: --------*
- E. a) is an allocation on the stack therefore the class will disappear when the function in which it is declared returns; b) is an allocation on the heap, it is not deleted when the function that allocate the object returns — *Feedback: --------*
- F. a) is an allocation on the heap, when the function returns it is not automatically deleted b) is neither an allocation on the heap or on the stack — *Feedback: --------*

Answer Point Value: 10.0 points
Answer Key: C, E
Correct Feedback: --------
Incorrect Feedback: --------

7

You want to write a class and initialize one of the variable members to a specific value (in this case is an integer, and you want to assign to it the value 4). Are the following codes legal? (Legal in the sense that they are syntactically correct following the C++ standard.)
a)

```
class my_class {
public:
  int i(4);
};
b)
class my_class {
public:
  static int i(4);
};
c)
class my_class {
public:
  static const int i(4);
};
```

- A. a) is legal, but b) and c) are not legal — *Feedback: --------*
- B. Yes, they are all trying to define a variable and initialize it to 4 — *Feedback: --------*
- C. No, they are all trying to define a function instead of a variable — *Feedback: --------*
- D. Yes, they are all legal — *Feedback: --------*
- E. No, they are all wrong, therefore not legal — *Feedback: --------*
- F. b) is legal, but a) and c) are not legal — *Feedback: --------*

Answer Point Value: 10.0 points
Answer Key: C, E
Correct Feedback: --------
Incorrect Feedback: --------

8

You want to write a class and initialize one of the variable members to a specific value (in this case is an integer, and you want to assign to it the value 4). Are the following codes legal? (Legal in the sense that they are syntactically correct following the C++ standard.) Explain why.

```
a)
class my_class {
public:
int i(4);
};
b)
class my_class {
public:
static int i(4);
};
c)
class my_class {
public:
static const int i(4);
};
```

!!!!Answer Point Value: 10.0 points
Model Short Answer: --------
Feedback: --------

9 Which are the differences between function overloading and function overriding in C++?

!!!!Answer Point Value: 10.0 points
Model Short Answer: --------
Feedback: --------

10 Which are the differences between function overloading and function overriding in C++?

- A. With function overloading multiple function with the same name and return type, but different parameters list and implementations, can exists in the same program. Instead, function overriding allow redefinition of the same function in a subclass. *Feedback: --------*
- B. With function overloading multiple function with the same name but different parameters list and implementations, can exist in the same program. Instead, function overriding allow redefinition of the same function in a subclass. *Feedback: --------*
- C. With function overriding multiple function with the same name and return type, but different parameters list and implementations, can exists in the same program. Instead, function overloading allow redefinition of the same function in a subclass. *Feedback: --------*
- D. There are no differences, they are the same concept *Feedback: --------*
- E. With function overriding multiple function with the same name but different parameters list and implementations, can exist in the same program. Instead, function overloading allow redefinition of the same function in a subclass. *Feedback: --------*

Answer Point Value: 10.0 points
Answer Key: A
Correct Feedback: --------
Incorrect Feedback: --------

11 C++ allows operator overloading. Which of the following operators cannot be overloaded?

- A. == *Feedback: --------*
- B. <= *Feedback: --------*
- C. -= *Feedback: --------*
- D. + *Feedback: --------*
- E. << *Feedback: --------*
- F. The ternary operator *Feedback: --------*
- G. . *Feedback: --------*
- H. .* *Feedback: --------*
- I. - *Feedback: --------*

J. >> *Feedback: --------*

K. :: *Feedback: --------*

L. += *Feedback: --------*

Answer Point Value: 10.0 points
Answer Key: F, G, H, K
Correct Feedback: --------
Incorrect Feedback: --------

12 C++ allows operator overloading. Which are the operators that cannot be overloaded?

!!!!Answer Point Value: 10.0 points
Model Short Answer: --------
Feedback: --------

13 Considering the following class declarations, select the correct answer.

a)
```
class my_class {
public:
int get_int() { return my_int; }
private:
int my_int;
};
```
b)
```
class my_class {
protected:
int get_int() { return my_int; }
private:
int my_int;
};
```
c)
```
class my_class {
private:
int get_int() { return my_int; }
int my_int;
};
```

A. The member variable my_int in can be set at initialization time in implementations a) and c). In implementation b) the variable cannot be set at any time. *Feedback: --------*

B. The member variable my_int in all implementations can be set at initialization time. All implementations are equivalent. *Feedback: --------*

C. The member variable my_int in all implementations cannot be set by the user *Feedback: --------*

D. The member variable my_int in all implementations can be set at initialization time *Feedback: --------*

E. The member variable my_int in can be set at initialization time in *Feedback:*

implementations a) and b). In implementation c) the variable cannot be set at any time. --------

F. The member variable my_int in can be set at initialization time only in implementation a). In the other implementations the variable cannot be set at any time. --------

Feedback:

Answer Point Value: 10.0 points
Answer Key: C
Correct Feedback: --------
Incorrect Feedback: --------

14

Considering the following class declarations, discuss for each case if it is possible or not to set the value of my_int. If it is not possible explain why.

a)

```
class my_class {
public:
int get_int() { return my_int; }
private:
int my_int;
};
```

b)

```
class my_class {
protected:
int get_int() { return my_int; }
private:
int my_int;
};
```

c)

```
class my_class {
private:
int get_int() { return my_int; }
int my_int;
};
```

!!!!Answer Point Value: 10.0 points
Model Short Answer: --------
Feedback: --------

15

For each of the following class constructors, namely a), b), and c), identify the type (default, copy constructor, normal), and briefly explain the usage (an example is fine).

```
class my_class {
public:
my_class(); // a)
my_class(const my_class& mc); // b)
my_class(const int a1, int a2, int* a3); // c)
};
```

!!!!Answer Point Value: 10.0 points
Model Short Answer: --------
Feedback: --------

16 For each of the following class constructors, namely a), b), and c), identify the type (default, copy constructor, normal).

```
class my_class {
public:
my_class(); // a)
my_class(const my_class& mc); // b)
my_class(const int a1, int a2, int* a3); // c)
};
```

- A. a) copy; b) default; c) normal; *Feedback: --------*
- B. a) copy; b) normal; c) default; *Feedback: --------*
- C. a) normal; b) copy; c) default; *Feedback: --------*
- D. a) normal; b) default; c) copy; *Feedback: --------*
- E. a) default; b) copy; c) normal; *Feedback: --------*
- F. a) copy; b) normal; c) default; *Feedback: --------*

Answer Point Value: 10.0 points
Answer Key: E
Correct Feedback: --------
Incorrect Feedback: --------

17 In the context of container (or collection) objects, such as QList, which are the differences between an aggregate and a composite relationships? (Focus your answer on the conteinee objects life management.)

!!!!Answer Point Value: 10.0 points
Model Short Answer: --------
Feedback: --------

18 In the context of container objects, such as QList, which are the differences between an aggregate and a composite relationships?

- A. There are no differences. Every container object has an aggregate as well as a composite relationship with the contained objects. *Feedback: --------*
- B. If the relationship is an aggregate, the container will manage the life-cycle of the contained objects, while in case of a composite relationship the container will not manage the life-cycle of the contained objects. *Feedback: --------*
- C. If the relationship is a composite, the container will manage the life-cycle of the contained objects, while in case of an aggregate relationship the container will not manage the life-cycle of the contained objects. *Feedback: --------*

Answer Point Value: 10.0 points
Answer Key: C
Correct Feedback: --------
Incorrect Feedback: --------

19 In the C/C++ programming language. When a class is defined abstract?

- ☐ A. When there is at least one virtual functoin — *Feedback: --------*
- ☐ B. When the declared constructors are all public — *Feedback: --------*
- ☐ C. When the declared constructors are all private — *Feedback: --------*
- ☐ D. When it cannot be directly instantiated — *Feedback: --------*
- ☐ E. When it can be directly instantiated — *Feedback: --------*
- ☐ F. When there is at least one pure virtual function — *Feedback: --------*

Answer Point Value: 10.0 points
Answer Key: C, D, F
Correct Feedback: --------
Incorrect Feedback: --------

20 In the C/C++ programming language. Explain whan a class is said to be abstract.

!!!!Answer Point Value: 10.0 points
Model Short Answer: --------
Feedback: --------

21 Describe the differences between the "base class-subclass" relationship and the "parent-children classes" relationship in the context of object oriented programming.

!!!!Answer Point Value: 10.0 points
Model Short Answer: --------
Feedback: --------

22 Which are the differences between the following class-to-class relationships?
a) parent-children
b) base class-subclass

- ○ A. The parent-children class relationship doesn't exist — *Feedback: --------*
- ○ B. Parent-children classes relationship identifies a run-time relationship, while base class-subclass identifies a compile time relationship. In fact in parent-children relationships all classes can be of the same class type, instead in base class-subclass relationship multiple different class types are involved — *Feedback: --------*
- ○ C. Parent-children classes relationship identifies a run-time relationship, while base class-subclass identifies a compile time relationship — *Feedback: --------*
- ○ D. There is no difference. They are the same — *Feedback: --------*
- ○ E. They can be both represented within UML diagrams — *Feedback: --------*

F. Parent-children classes relationship identifies a compile time relationship, while base class-subclass identifies a run-time relationship

Feedback: --------

Answer Point Value: 10.0 points
Answer Key: B, C, E
Correct Feedback: --------
Incorrect Feedback: --------