SULTAN QABOOS UNIVERSITY, COLLEGE OF SCIENCE
DEPARTMENT OF COMPUTER SCIENCE, FALL 2010
COMP3200: OBJECT ORIENTED PROGRAMMING

MIDTERM EXAM

November 14th, 2010 - Duration: 90 minutes

Solution

Question 1 (17 points): Mark the following statements as true or false:

	Statement	**True/False**
1.	The C++ compiler generates physical copies of a function member of a class for each class object	F
2.	If an object is declared in the definition of a member function of the class, then the object can access both the `public` and `private` members of the class	T
3.	If the heading of a member function of a `class` ends with the word `const`, then the function member cannot modify the `private` member variables, but it can modify the `public` member variables	F
4.	The constructor of a derived class specifies a call to the constructor of a base class in the body of the constructor definition	F
5.	A derived class can directly access the `protected` members of the base class	T
6.	In `protected` inheritance, `public` and `protected` members of the base class become the `protected` members of the derived class	T
7.	In the case of composition, you use the class name to invoke the base class's constructor.	F
8.	To overload a member function of the base class, the name of the function and the formal parameter list of the corresponding function in the derived class must be same.	F
9.	In C++, the dot operator has a lower precedence than the dereferencing operator	F
10.	Two pointer variables of the same type can be compared for equality	T

11.	Given the declarations
	
	`int list[10];`
	`int *p;`
	
	the statement
	
	`list = p;`
	
	is valid in C++.
		F

12. A pointer variable can be passed as a parameter to a function either by value or by reference.	**T**
13. Suppose that `p` and `q` are pointers of type `int`. The statement `p = q;` will result in shallow copying of data.	**T**
14. C++ does not allow the user to pass an object of a derived class to a formal parameter of the base class type.	**F**
15. The binding of virtual functions occurs at program execution time.	**T**
16. An abstract class does not need to provide the definitions of the member functions that are not pure virtual because you cannot create objects of the abstract class.	**F**
17. It is not necessary to include the copy constructor in classes with pointer member variables	**F**

Question 2 (15 points): select the most appropriate answer for each of the following questions

1. A class object can be ______________. That is, it can be created once, when the control reaches its declaration, and destroyed when the program terminates.

a. local

b. automatic

c. static

d. public

2. In C++, you can pass a variable by reference and still prevent the function from changing its value, by using the keyword ________________ in the formal parameter declaration.

a. `const`

b. `static`

c. `private`

d. `automatic`

3. To ______________ a `public` member function of a base class in the derived class, the corresponding function in the derived class must have the same name, number, and types of parameters.

a. overload

b. redefine

c. rename

d. reuse

4. If the derived class `classD` overrides a `public` member function `functionName` of the base class `classB`, then to specify a call to that `public` member function of the base class you use the ________________ statement.

a. `classD.functionName();`

b. `classB::functionName();`

c. `classB.functionName();`

d. `classD::functionName();`

5. Which of the following statements is true about protected inheritance?

a. The `private` members of the base become `protected` members of the derived.

b. The `protected` members of the base become `private` members of the derived.

c. The `public` members of the base become `protected` members of the derived.

d. The derived can directly access any member of the base.

6. ______________ is the ability to use the same expression to denote different operations.

a. Polymorphism

b. Inheritance

c. Composition

d. Encapsulation

7. C++ provides _______________ functions as a means to implement polymorphism in an inheritance hierarchy, which allows the run-time selection of appropriate member functions.

a. overloaded

b. overridden

c. redefined

d. virtual

8. In a _______________ copy, two or more pointers of the same type point to the same memory.

a. shallow

b. deep

c. dynamic

d. static

9. The _____________ constructor is called when an object is passed as a (value) parameter to a function.

a. copy

b. default

c. struct

d. class

10. What is the output of the following code?

```
int *p;
int x = 12;
p = &x;
cout << x << ", ";
*p = 81;
cout << *p << endl;
```

a. `81, 12`

b. `81, 81`

c. `12, 12`

d. 12, 81

11. If you overload the binary arithmetic operator + as a member function, how many objects must be passed as parameters?

a. zero

b. one

c. two

d. three

12. Every object of a class maintains a (hidden) pointer to itself, and the name of this pointer is ____________.

a. this

b. `self`

c. `it`

d. `object`

13. A(n) ______________ function is a nonmember function that has access to all members of the class.

a. `virtual`

b. friend

c. `void`

d. `protected`

14. A class ____________ automatically executes whenever a class object goes out of scope.

a. pointer

b. exception

c. constructor

d. destructor

15. In _____________ binding, the necessary code to call a specific function is generated by the compiler.

a. static

b. shallow

c. dynamic

d. deep

Question 3 (7 points): write C++ code to declare a dynamic two dimensional array "triangle" that consists of 5 rows of different number of columns (as shown). Also initialize the array to the values shown in the figure? You must not use initializer lists.

1	**2**				
3	**4**	**5**			
6	**7**	**8**	**9**		
10	**11**	**12**	**13**	**14**	
15	**16**	**17**	**18**	**19**	**20**

```
int rows = 5;
int **triangle;
triangle = new int*[rows];

int value = 1;
for (int i=0; i<rows; i++) {
    triangle[i] = new int[i+2];
    for (int j=0; j<i+2; j++) {
        triangle[i][j] = value;
        value++;
    }
}
```

Question 4 (6 points): given the class definition and implementations of `firstClasss` **and** `secondClass` **below, what is the output of the following main program?**

```
class firstClass {
      int x;
public:
      virtual void print() const;
      virtual void changeNumber();
      firstClass(int a=0);
};
void firstClass::print() const {
      cout << "First Class  x = " << x << endl;
}
void firstClass::changeNumber() {
      x = 2*x;
}
firstClass::firstClass(int a) {
      x = a;
}


class secondClass : public firstClass {
      int y;
public:
      void print() const;
      void changeNumber();
      secondClass(int a=0, int b=0);
};
void secondClass::print() const {
      firstClass::print();
      cout << "Second Class y = " << y << endl;
}
void secondClass::changeNumber() {
      firstClass::changeNumber();
      y = y*2;
}
secondClass::secondClass(int a, int b) : firstClass(a) {
      y = b;
}


int main() {
      firstClass obj1(2);
      secondClass obj2(3, 5);

      firstClass * ptr = &obj1;

      ptr->changeNumber();
      ptr->print();

      ptr = &obj2;
      ptr->changeNumber();
      ptr->print();

      return 0;
}
```

Output:

```
First Class    x = 4
First Class    x = 6
Second Class  y = 10
```

Question 5 (15 points): write class definition and implementation for a class "Grades" that represents student's grades in an exam. The data members of the grades class are a dynamic array of integer grades, the maximum capacity of the array and the actual number of grades in the array. Please note the following:

- **You don't need to provide any accessor or mutator functions.**
- **Provide appropriate constructor(s), copy constructor and destructor**
- **Override the equality operator == for the Grades class. Two objects are considered equal only if they contain exactly same grade lists.**

```
class Grades {
   int *list;
   int size, max;
public:
   Grades(int m=100);
   Grades(const Grades& obj);
   ~Grades();
   bool operator==(const Grades& other) const;
};

Grades::Grades(int m) {
   max = m;
   size = 0;
   list = new int[max];
}

Grades::Grades(const Grades &obj) {
   max = obj.max;
   size = obj.size;
   list = new int[max];
   for (int i=0; i<size; i++)
        list[i] = obj.list[i];
}

Grades::~Grades() {
   delete [] list;
}

bool Grades::operator==(const Grades &other) const {
   if (size==other.size && max==other.max) {
        for (int i=0; i<size; i++)
             if (list[i] != other.list[i])
                  return false;
        return true;
   }
   return false;
}
```