ECE, Concordia University
COEN 244 W14 - Programming Methodology II
Dr. A. Hamou-Lhadj
Midterm Exam - February 25, 2014 - Duration: 75 min

Name: _____

Student #: _____

Closed book and no calculators allowed.
Clearly state all assumptions.

TYPE A

Circle the best answer. There is **only one** best answer. Each question is worth 3 marks.

1. **What is the value of x after executing the following code?**

```
int x;
int *p;
x = 20;
p = &x;
*p = 25;
cout << "The value of x is:" << x << endl;
```

a. 25 **(*p = 25 changes the value of x because p points to x)**
b. 26
c. 20
d. 19

2. **What is the output of the following program?**

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

int f(int &);
int g(int);

int main() {
   int a = 20;
   cout << f(a) << endl;
   return 0;
}
```

```
int f(int &x) {
   return (g(x) + x);
}

int g(int x) {
   return x++;
}
```

a. 40
b. 20
c. 41 **(my intent was this, but some students argued that x++ is not executed because of the return. I agree with them. The correct answer is then 'a. 40'. If I took marks off, please come see me).**
d. 21

3. **What is the main benefit of *Information Hiding* in Object-Orientation?**
   a. To make the program easier to maintain **(Information Hiding consists of keeping data members private. Only the class that defines them can access them. Changes inside the class won't affect other parts of the program that use the class as long as the class exhibits the same public member functions).**
   b. To simplify the writing of the program
   c. To make the program shorter
   d. To improve program performance

4. **In which of the following situations the copy constructor is called?**
   a. When a function returns a pointer to an object

b. When an object is passed by value (**Copy constructor is only called when you assign an object to another object while that other object is being created. Passing an object by value creates a copy object, hence the copy constructor is called. Answer c (most selected answer) is not correct because a simple assignment of objects that does not involve the construction of objects does not call the copy constructor).**

c. Whenever an object is assigned to another object

d. When an object is passed by reference

5. A destructor is a special member function that:

a. Is called before an object is destroyed **(The last function that is called before an object goes out of scope or deleted)**

b. Destroys an object that is created dynamically

c. Is used to access private members

d. Is used to set pointer variables to NULL

6. Consider the following code fragment:

```
class A {                       class B: public A {
      protected:                      protected:
        int var1;                       int var2;
        …                               …
};                              };
```

Consider the following main function, defined outside the classes A and B:

```
int main() {
    A obj1;                 // line 1
    B obj2;                 // line 2
    cout << obj1.var1;      // line 3
    cout << obj2.var2;      // line 4
    cout << obj2.var1;      // line 5
    obj1 = obj2;            // line 6
    return 0;
}
```

Which of the following lines will cause a compiler error?

a. 4 and 5 only

b. 4 only

c. 3 and 5 only

d. 3, 4, and 5 only **(var1 and var2 are not accessible outside the classes A and B. They are protected. Protected members are only accessible within the class hierarchy).**

7. In class composition, why is it important to use member initialization lists to initialize objects of the component classes?

a. To improve the performance of the program **(the objective is to speed up the assignment of component objects).**

b. This is the only way for initializing these members

c. To optimize memory space occupied by these objects

d. To improve the readability of the program

8. Which one of the following statements about a constructor <u>is true</u>?

a. Each class must have at least one constructor **(As I said in the class, even if you don't specify a constructor, a default constructor is created. A class must have at least one constructor. A constructor can be public, private, or protected)**
b. A constructor must always be declared public
c. A constructor function may have a return type
d. Data members can only be initialized within a constructor

9. How can we ensure that a function will not change an object passed to it as an argument without having to pass it by value?
a. Pass it by reference
b. Pass it as a pointer
c. Pass it by reference as a constant object **(It should be passed by reference so no duplicate object is created and as a constant so the function can't change it).**
d. Pass data members of the object

10. What does the term "multiple" in *multiple inheritance* mean?
a. Multiple derived classes
b. Multiple objects
c. Multiple base classes **(This is what it means)**
d. Multiple levels of inheritance

11. Suppose a class B derives from a class A. What happens when an object of B is created?
a. The constructor of A is executed first then the constructor of B
b. The constructor of B is executed first then the constructor of A
c. The constructor of B is executed but the constructor of A is not
d. The constructor of A is executed but the constructor of B is not

12. Which one of the following statements about friend functions <u>is false</u>?
a. A friend function of a class provides more efficient access to data members
b. A friend function of a class has direct access to data members of that class
c. A friend function is not inherited in class inheritance **(All the other answers are incorrect. A friend function has direct access to private members (answer b), which in turns improve efficiency (answer a). It can be called on an object just like any other member function, answer d).**
d. A friend function of a class is called on an object of that class

The next five questions are based on the following code:

```
/* The class Account represents a bank account, just like the one we saw in the
   Class. */

class Account {
    private:
        int accountNum;           // account number
        double balance;           // balance

    public:
        Account();                      // default constructor
        Account(int, double);           // regular constructor
        Account(const Account&)         // copy constructor
        ~Account();                     // destructor

        int getAccountNumber() const;   // returns account number
```

```
        double withdrawal(double);      // makes a withdrawal
        double deposit(double);         // makes a deposit
        void setBalance(double);        // sets the balance
        double getBalance() const;      // returns the balance
};

/*
   The implementation of the Account Class is as follows:
*/

#include <iostream>
#include "account.h"
using namespace std;

// default constructor
Account::Account() { accountNum = 0; balance = 0.0;}

// regular constructor
Account::Account(int an, double b) { accountNum = an; balance = b;}

// copy constructor
Account::Account(const Account& anotherAccount) {
    accountNum = anotherAccount.accountNum;
    balance = anotherAccount.balance;
}

// destructor
Account::~Account()  {
   // does nothing
}

// makes a withdrawal
double Account::withdrawal(double amount) {
     if (balance >= amount)  balance = balance - amount;
     else cout << "Insufficient funds" << endl;
     return balance;
}

// makes a deposit (assume amount is positive)
double Account::deposit(double amount)
{
     balance = balance + amount;
     return balance;
}

// sets the balance
void Account::setBalance(int b)
{
      balance = b;
}

//  returns the balance
double Account::getBalance() const
{
      return balance;
}

/*
   The class Person below represents the owner of an account. Note that a
   person can have many accounts (a maximum of 10). */

class Person {
```

```
  private:
     string name;          // name of the person
     Account *accounts;    // list of accounts
     int numAccounts;      // the number of accounts owned by a person

  public:
     Person(string)                    // regular constructor
     Person(const Person&);            // copy constructor
     ~Person();                        // destructor

     void addAccount(const Account&);  // adds an account to the person
     void removeAccount(int);          // removes  an account
     void listAllAccounts() const;     // displays all accounts
     bool searchAccount(int) const;    // searches for an account
     void deleteAllAccounts();         // deletes all accounts of a person
     double getTotalAmounts(); const   // returns sum of balances
     bool compareAccounts(const Person&) const; // compares accounts.
};
```

Part of the implementation of the Person class is provided in what follows:

```
#include <iostream>
#include "account.h"
#include "person.h"

using namespace std;

// regular constructor
Person::Person(string name1) {
    name = name1;
    accounts = new Account[10]; // creates an array of 10 accounts
    numAccounts = 0;            // so far the person owns no accounts
}
```

13. The function addAccount adds an account to the list of accounts of a person. Which one of the following implementations is a correct implementation of this function?

a.

```
   void Person::addAccount(const Account& a) {
     if (numAccounts < 10) {
        accounts[numAccounts]=a[numAccounts++];
     }
     else {
       cout << "Number of  accounts exceeded" ;
     }
     return;
   }
```

b.

```
   void Person::addAccount(const Account& a) {
      if (numAccounts < 10) {
        accounts[numAccounts] = a;
      }
      else {
       cout << "Number of  accounts exceeded";
      }
```

```
      return;
   }
c.
   void Person::addAccount(const Account& a) {
      if (numAccounts < 10) {
        a = accounts[numAccounts];
        numAccounts++;
      }
      else {
       cout << "Number of  accounts exceeded" ;
      }
      return;
   }
```

```
d.
   void Person::addAccount(const Account& a) {
      if (numAccounts < 10) {
        accounts[numAccounts] = a;
        numAccounts++;
      }
      else {
       cout << "Number of  accounts exceeded";
      }
      return;
   }
```

You need to assign a to accounts[numAccounts] and increment numAccounts.

14. **The removeAccount function takes an account number and removes it from the list of accounts of a person. The function is missing two statements: (1) and (2).**

```
   void Person::removeAccount(int acct) {
       // Search for the Account acct
       for (int i = 0; i < numAccounts; i++) {
         if (accounts[i].getAccountNumber()== acct) {
            // Shift the elements of the array
            for (int j = i; j < numAccounts-1; j++) {
                 (1) _ _ _ _ _ _ _ _ _ _ _ _
            }
            (2)  _ _ _  _ _ _ _ ___
            return;
         }
       }
       cout << "Account " << acct << " cannot be found " << endl;
   }
```

Complete the above implementation by choosing one of the options below:

```
a. (1) accounts[j] = accounts[j-1];     (2) numAccounts++;
b. (1) accounts[j] = acct;              (2) numAccounts--;
c. (1) accounts[j] = accounts[j+1];     (2) numAccounts--;
d. (1) accounts[j] = acct;              (2) numAccounts--;
```

This is straightforward. We are simply shifting the array and updating the numAccounts.

15. **The deleteAllAccounts member function removes all accounts owned by a person. The function is missing three statements (1), (2), and (3) that you need to complete by choosing one of the options below.**

```
void Person::deletAllAccounts() {
     (1)  _ _ _ _ _ _ _ _ _ _ _ _
     (2)  _ _ _ _ _ _ _ _ _ _ _ _
     (3)  _ _ _ _ _ _ _ _ _ _ _ _
     return;
}
```

```
a. (1) accounts = new Account[10];
   (2) numAccounts = 0;
   (3) delete[] accounts;
```

```
b. (1) delete[] accounts;
   (2) accounts = new Account[10];
   (3) numAccounts = 0;
```

Note that answer c is incorrect because accounts[i] is an object created statically. 'Delete' does not apply to static objects. Answer d is incorrect because you are creating new variables and not using the data members of the class.

```
c. (1) for (int i = 0; i < numAccounts; i++) delete accounts[i];
   (2) accounts = new Account[10];
   (3) numAccounts = 0;
```

```
d. (1) delete[] accounts;
   (2) Account* accounts = new Account[10];
   (3) int numAccounts = 0;
```

16. **The getTotalAmounts() member function returns the sum of all balances owned by a person. Recall that a person can have many accounts and each account has a balance. The function is missing three statements, (1), (2), and (3).**

```
double Person::getTotalAmounts() {
    double sum;
    (1) _ _ _ _ _ _ _ _ _ _ _ _
    for (int i = 0; i < numAccounts; i++)
```

```
        (2) _ _ _ _ _ _ _ _ _ _ _ _ _
    (3) _ _ _ _ _ _ _ _ _ _ _ _ _
}
```

Complete the above implementation by choosing one of the options below:

```
a. (1) sum = accounts[0].getBalance();
   (2) sum = sum + accounts[i].getBalance();
   (3) return sum;

b. (1) sum = 0.0;
   (2) sum = accounts[i].getBalance();
   (3) return sum;

c. (1) sum = 0.0;
   (2) sum = sum + accounts[i];
   (3) return sum;

d. (1) sum = 0.0;
   (2) sum = sum + accounts[i].getBalance();
   (3) return sum;
```

Answer a is incorrect because you assume you have one object in the array, which is not true. Answer b is incorrect because you are not summing up the balances. Sum will end up containing the balance of the last account in the array. Answer c is incorrect because accounts[i] returns an object and not the balance.

17. The member function compareAccounts (const Person&) takes an object of Person as input and returns true if the total amounts of this person is equal to the total amounts of the person on which the function is invoked. Otherwise, the function returns false.

```
bool Person::compareAccounts(const Person& o1) {
       (1) _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
       (2) _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
       (3) _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
       (4) _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
}
```

Complete the above implementation by choosing one of the options below.

```
a. (1) double a = getTotalAmounts();
   (2) double b = o1.getTotalAmounts();
   (3) if (a == b) cout << "Yes";
   (4) else cout << "No";

b. (1) for (int i = 0; i < numAccounts; i++)
   (2) for (int j = 0; j < o1.numAccounts; j++)
   (3) if (accounts[i].getBalance() == o1.accounts[j].getBalance())
         return true;
   (4) return false;
```

```
c. (1) double a = getTotalAmounts();
   (2) double b = o1.getTotalAmounts();
   (3) if (a == b) return true;
   (4) return false;
```

Answer a is incorrect because you don't return anything. Answer b is incorrect because it does not wait until the sum of all balances is computed. Answer d returns the correct answer but it is not efficient because it loops for no reason. Remember we are looking for the best answer.

```
d. (1) for (int i = 0; i < numAccounts; i++)
   (2) for (int j = 0; j < o1.numAccounts; j++)
   (3) if (getTotalAmounts() == o1. getTotalAmounts()) return true;
   (4) return false;
```

The next three questions are based on the following code:

The class SavingsAccount derives from the class Account. In addition to the attributes defined in the Account class, the SavingsAccount class has the following attributes:

privileges (string)	A list of privileges associated with the account such as the ability to write cheques, online access, etc.
overdraftLimit (double)	The overdraft limit is an amount of money that the bank lends to an account owner when the account is overdrawn (there is no more money in the account). In French, we call this "limite de découvert"

```
/* The class SavingsAccount inherits from Account */

class SavingsAccount: public Account {
    private:
        string privileges;       // account privileges
        double overdraftLimit;   // overdraft limit

    public:
       SavingsAccount();                    // default constructor
       SavingsAccount(int, double, string, double);  // regular constructor
       double withdrawal(double);           // makes a withdrawal
       void changePrivileges(string);       // changes the privileges
       string getPrivileges() const;        // returns privileges
       double getOverdraftLimit();const     // returns overdraft limit
       void setOverdraftLimit(double);      // modifies the overdraft limit
};
```

18. Which of the following member functions of the SavingsAccount class overrides functions of the Account class?

a. SavingsAccount();
b. SavingsAccount(int, double, string, double);
c. double withdrawal(double); **(An overridden member function must have the same signature in the base class as well as in the derived class)**
d. void changePrivileges(string);

19. Which of the following implementations of the SavingsAccount constructor is correct?

```
a. SavingsAccount::SavingsAccount(int acctNum1, double balance1,
            string privileges1, double overdraft1) {
        Account(acctNum1, balance1);
        privileges = privileges1;
        overdraftLimit = overdraft1;
   }

b. SavingsAccount::SavingsAccount(int acctNum1, double balance1,
                    string privileges1, double overdraft1) {
        accountNum = acctNum1;
        balance = balance1;
        privileges = privileges1;
        overdraftLimit = overdraft1;
   }

c. SavingsAccount::SavingsAccount(int acctNum1, double balance1,
            string privileges1, double
            overdraft1):Account(acctNum1, balance1) {

        privileges = privileges1;
        overdraftLimit = overdraft1;
    }
```

Answer a is incorrect because you can't call a constructor like this. Answer b is incorrect because accountNum and balance are private in the class Account. You can't access them from SavingsAccount. Answer d is incorrect because the constructor of Account does not understand privilege1 and overdraftLimit1.

```
d. SavingsAccount::SavingsAccount(int acctNum1, double balance1,
                    string privileges1, double overdraft1):
                    Account(acctNum1, balance1, privilege1,
                    overdraft1) {

          /* Nothing is needed here */
   }
```

20. Just like in the Account class, The withdrawal(double) member function of the SavingsAccount class is used to withdraw an amount of money from the account. The only difference is that, in addition to the balance, a savings account has also an overdraft limit. This said, the total amount of money in an account is the sum of the balance and the overdraft limit.

For example, consider a savings account with the following information

Balance = $100
Overdraft Limit = $300

Assume now that we want to withdraw the amount of $200. The resulting balance is $0 (100 –200 = - 100, meaning that the missing 100$ needs to be taken from the overdraft limit). The resulting overdraft limit is $200 ($300 – $100)

The implementation of the function is given in what follows. It misses three statements that you need to choose from the options below.

```
double SavingsAccount::withdrawal(double amount) {
    // check for sufficient funds
   if ((getBalance() + overdraftLimit) >= amount)
   {
      double diff = getBalance() - amount;

      if ((1) _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ ) {

        (2) _ _ _ _ _ _ _ _ _ _ _ _ _

        (3) _ _ _ _ _ _ _ _ _ _ _ _ _

      }
      else {
         setBalance(diff);
      }
   }
   else {
      cout << "Insufficient funds" << endl;
   }

   return getBalance();
}
```

a. (1) diff < 0
(2) overdraftLimit = overdraftLimit + diff;
(3) setBalance(0);

Answer b is incorrect because diff > 0. Answer c is incorrect because of getBalance and also the way the new balance is set. Answer d is incorrect because of (1) and (2). In fact, Answers c and d do not make any sense.

b. (1) diff > 0
(2) overdraftLimit = overdraftLimit - diff;
(3) setBalance(0);

c. (1) diff < 0
(2) overdraftLimit = overdraftLimit + getBalance();
(3) setBalance(getBalance() - amount);

d. (1) getBalance() - amount < 0
(2) overdraftLimit = overdraftLimit + getBalance();
(3) setBalance(0);