DALHOUSIE UNIVERSITY
DEPARTMENT OF ENGINEERING MATHEMATICS
ENGM3282

ASSIGNMENT # 8, Due date: Tuesday November 6, 2018, 1:00 PM

1. In this question we model a database of employees of a company. Assume that there are only three types of employees: managers, scientists and labourers. The database is to store a name, an employee number, and a salary for each employee. In addition, for managers a title is stored, for scientists the number of publications is stored, for labourers a department is stored.

   We will use a base class called employee and three derived classes: labourer, scientist and manager as shown below.

   Write the required member functions for implementing these classes in the source file `employeetypes.cpp`.

   For marking purposes run the program and use the following data.

```
/* data to be entered

   labourer
   name: white
   id: 3434
   salary: 41000
   department: shipping

   manager
   name: black
   id: 565
   salary: 54000
   title: accountsmanager

   scientist
   name: Green
   id: 234
   salary: 79000
   number of publications: 23
*/

// File: employeetypes.cpp
// Program to illustrate derived classes

#include <iostream>
#include <fstream>
#include <string>
using namespace std;

class employee {
private:
    string name;  // last name
    int id;       // employee id number
    float salary; // employee salary

public:
```

```
    employee(string n="none", int i=0, float s=0.0);
    void write(ostream &out);
    void read(istream &in);
};

class manager : public employee {
private:
    string title;       // vice-president etc
public:
    manager(string n="none", int i=0, float s=0.0, string t="none");
    void write(ostream &out);
    void read(istream &in);
};

class scientist : public employee{
private:
    int pubs;           // number of publications

public:
    scientist(string n="none", int i=0, float s=0.0, int p=0);
    void write(ostream &out);
    void read(istream &in);
};

class labourer : public employee {
private:
    string dept;

public:
    labourer(string n="none", int i=0, float s=0.0, string d="none");
    void write(ostream &out);
    void read(istream &in);
};

int main()
{
    string type;            // employee type
    ofstream fout ("employeetypesout.txt");

    while(1) { // loop until break occurs
        cout << "\n\nType of employee: "; cin >> type;
        if (type == "q") break;

        if(type == "labourer") {
            labourer lab;
            lab.read(cin);

            cout << "\n";
            lab.write(cout);
            fout << "\n";
            lab.write(fout);

        } else if (type == "scientist") {
            scientist sci;
```

```
            sci.read(cin);

            cout << "\n";
            sci.write(cout);
            fout << "\n";
            sci.write(fout);

        } else if (type == "manager") {
            manager man;
            man.read(cin);

            cout << "\n";
            man.write(cout);
            fout << "\n";
            man.write(fout);

        }
   }
   fout.close();
   return 0;
}

////////////////// Member functions of employee ////////////////////////////

employee::employee(string n, int i, float s)
{
    name = n;
    id = i;
    salary = s;
}
void employee::write(ostream &out)
{

    out << "name  : " << name <<   "\n";
    out << "id    : " << id  <<    "\n";
    out << "salary: " << salary << "\n";
}
void employee::read(istream &in)
{
    if(in == cin) { // give a prompt to the user
        cout << "name  : ";
    }
    in >> name;

    if(in == cin) { // give a prompt to the user
        cout << "id  : ";
    }
    in >> id;

    if(in == cin) { // give a prompt to the user
        cout << "salary: ";
    }
    in >> salary;
}
```

```
////////////////////// Member functions of labourer //////////////////////////


////////////////////// Member functions of manager ///////////////////////////


////////////////////// Member functions of scientist ////////////////////////
```

2. In this question we modify the bank account classes discussed in class so that the base `class account` has a method called `write` which sends the values of the members to a stream. This method is overridden in the derived classes to send any new members to the stream.

   `class savingsaccount` has the data member `interestrate` and the method `addinterest`.

   `class chequingaccount` has the data member `remainingfreewithdrawals` and overrides the `withdraw` method so that the withdrawal fee is not charged if `remainingfreewithdrawls` is positive.

   Complete the implemenation of the derived classes. To achieve code reuse, methods of the derived classes should utilize methods of the base classes.

   For marking purposes, run the program and capture the screen output.

   Your output should look like:

```
savings account:
owner: joe
balance: 100
withdrawfee 0.5
interest rate: 0.02

chequing account:
owner: bob
balance: 100
withdrawfee 0.5
remaing free withdrawals: 2

savings account:
owner: joe
balance: 64.77
withdrawfee 0.5
interest rate: 0.02

chequing account:
owner: bob
balance: 64.5
withdrawfee 0.5
remaing free withdrawals: 0
```

```
/* File: accountwrite.cpp
   Shows how to derive classes*/

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

class account {
private:
    string owner;
    float balance;
    float withdrawfee;

public:
    account(string name, float fee);
    void write(ostream &out);
    void deposit(float amount);
    bool withdraw(float amount);
    float getbalance(void);
    float getwithdrawfee(void);
};

class savingsaccount :public account {
private:
    float interestrate;

public:
    savingsaccount(string name, float fee, float rate);
    void write(ostream &out);
    void addinterest(void);
};

class chequingaccount : public account {
private:
    int remainingfreewithdrawals;

public:
    chequingaccount(string name, float fee, int numberfree);
    void write(ostream &out);
    bool withdraw(float amount);
};


int main(void)
{
    savingsaccount s("joe", 0.50, 0.02);
    chequingaccount c("bob", 0.50, 2);

    s.deposit(100.);
    c.deposit(100.);
```

```
    cout << "savings account: \n";
    s.write(cout);
    cout << "\nchequing account: \n";
    c.write(cout);

    s.withdraw(10.);
    s.withdraw(5.);
    s.withdraw(20.0);
    s.addinterest();

    cout << "\nsavings account: \n";
    s.write(cout);

    c.withdraw(10.);
    c.withdraw(5.);
    c.withdraw(20.0);

    cout << "\nchequing account: \n";
    c.write(cout);

   return 0;
}

//////////////////// implementation of account ///////////////////////
account::account(string name, float fee)
{
    owner = name;
    balance = 0;
    withdrawfee = fee;
}

void account::write(ostream &out)
{
    out << "owner: " << owner << "\n";
    out << "balance: " << balance << "\n";
    out << "withdrawfee " << withdrawfee << "\n";
}

void account::deposit(float amount)
{
    balance = balance + amount;
}

bool account::withdraw(float amount)
{
    bool result;
    if (amount > balance-withdrawfee) {
        cout << "Insufficient funds";
        result = false;
    } else {
        balance = balance - amount - withdrawfee;
        result = true;
    }
```

```
    return result;
}

float account::getbalance(void)
{
    return balance;
}

float account::getwithdrawfee(void)
{
    return withdrawfee;
}

//////////////////// implementation of savingsaccount ////////////////////////
```

```
//////////////////// implementation of chequingaccount ////////////////////////
```

3. In this question we modify the bank account classes of the previous question to derive `class savingsaccountplus` from `class savingsaccount`. This new class is like a savings account except that the withdrawal fee is not charged when the balance is greater than a minimum balance.

   Complete the implemenation of the derived classes. To achieve code reuse, methods of the derived classes should utilize methods of the base classes.

   For marking purposes, run the program and capture the screen output.

   Your output should look like:

```
savings account:
owner: joe
balance: 1200
withdrawfee 0.5
interest rate: 0.02

chequing account:
owner: bob
balance: 1200
withdrawfee 0.5
remaing free withdrawals: 2

savings account plus:
owner: jill
balance: 1200
withdrawfee 0.5
```

```
interest rate: 0.02
minimum balance: 1000

savings account:
owner: joe
balance: 814.47
withdrawfee 0.5
interest rate: 0.02

chequing account:
owner: bob
balance: 799.5
withdrawfee 0.5
remaing free withdrawals: 0

savings account plus:
owner: jill
balance: 815.49
withdrawfee 0.5
interest rate: 0.02
minimum balance: 1000
```

```
/* File: accountsavingsplus.cpp
   Shows how to derive classes */

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

class account {
private:
    string owner;
    float balance;
    float withdrawfee;

public:
    account(string name, float fee);
    void write(ostream &out);
    void deposit(float amount);
    bool withdraw(float amount);
    float getbalance(void);
    float getwithdrawfee(void);
};

class savingsaccount :public account {
private:
    float interestrate;

public:
    savingsaccount(string name, float fee, float rate);
```

```
    void write(ostream &out);
    void addinterest(void);
};

class chequingaccount : public account {
private:
    int remainingfreewithdrawals;

public:
    chequingaccount(string name, float fee, int numberfree);
    void write(ostream &out);
    bool withdraw(float amount);
};


class savingsaccountplus :public savingsaccount {
private:
    float minimumbalance;

public:
    savingsaccountplus(string name, float fee, float rate, float minbal);
    bool withdraw(float amount);
    void write(ostream &out);
};


int main(void)
{
    savingsaccount s("joe", 0.50, 0.02);
    chequingaccount c("bob", 0.50, 2);
    savingsaccountplus t("jill", 0.50, 0.02, 1000.00);
    s.deposit(1200.00);
    c.deposit(1200.00);
    t.deposit(1200.00);

    cout << "savings account: \n";
    s.write(cout);
    cout << "\nchequing account: \n";
    c.write(cout);
    cout << "\nsavings account plus: \n";
    t.write(cout);

    s.withdraw(100.00);
    s.withdraw(200.00);
    s.withdraw(100.00);
    s.addinterest();

    cout << "\nsavings account: \n";
    s.write(cout);

    c.withdraw(100.00);
    c.withdraw(200.00);
    c.withdraw(100.00);
```

```
    cout << "\nchequing account: \n";
    c.write(cout);

    t.withdraw(100.00);
    t.withdraw(200.00);
    t.withdraw(100.00);
    t.addinterest();

    cout << "\nsavings account plus: \n";
    t.write(cout);

   return 0;
}

//////////////////// implementation of account ////////////////////////
account::account(string name, float fee)
{
    owner = name;
    balance = 0;
    withdrawfee = fee;
}

void account::write(ostream &out)
{
    out << "owner: " << owner << "\n";
    out << "balance: " << balance << "\n";
    out << "withdrawfee " << withdrawfee << "\n";
}

void account::deposit(float amount)
{
    balance = balance + amount;
}

bool account::withdraw(float amount)
{
    bool result;
    if (amount > balance-withdrawfee) {
        cout << "Insufficient funds";
        result = false;
    } else {
        balance = balance - amount - withdrawfee;
        result = true;
    }

    return result;
}

float account::getbalance(void)
{
    return balance;
}

float account::getwithdrawfee(void)
```

```
{
    return withdrawfee;
}

//////////////////// implementation of savingsaccount ///////////////////////




//////////////////// implementation of chequingaccount ///////////////////////




///////////////// implementation of savingsaccountplus ///////////////////////
```

4. In this question we illustrate the behaviour of the big 3 methods (copy constructor, operator= and destructor) in a derived class.

   Complete the implementation of the `class point3`.

   For marking purposes, run the program and capture the screen output.

   Your output should look like:

```
creating point2: 1, 2
creating point3: 1, 2, 3
creating point2 copy: 1, 2
creating point3 copy: 1, 2, 3
creating point2: 0, 0
creating point3: 0, 0, 0
assigning point2: 1, 2
assigning point3: 1, 2, 3
desroying point3: 1, 2, 3
desroying point2: 1, 2
desroying point3: 1, 2, 3
desroying point2: 1, 2
desroying point3: 1, 2, 3
desroying point2: 1, 2
```

```
/*  File: points2and3.cpp
    investigate the big3 with inheritance*/

#include <iostream>
#include <fstream>
```

```
using namespace std;

class point2 {
private:
    float x;
    float y;
public:
    point2(float xval = 0.0, float yval = 0.0);
    point2(const point2 &other);
    ~point2(void);
    point2& operator=(const point2 &rhs);

    void write(ostream &out) const;
};

class point3 : public point2 {
private:
    float z;
public:
    point3(float xval = 0.0, float yval = 0.0, float zval = 0.0);
    point3(const point3 &other);
    ~point3(void);
    point3 &operator=(const point3 &rhs);

    void write(ostream &out) const;
};

int main(void)
{
    point3 a(1.0, 2.0, 3.0), b(a), c;

    c = a;

    return 0;
}

////////////////////// implementation of point2 ////////////////////////

point2::point2(float xval, float yval)
{
    x = xval;
    y = yval;
    cout << "creating point2: ";
    write(cout);
    cout << endl;
}
point2::point2(const point2 &other)
{
    x = other.x;
    y = other.y;
    cout << "creating point2 copy: ";
    write(cout);
    cout << endl;
}
```

```
point2::~point2(void)
{
    cout << "desroying point2: ";
    write(cout);
    cout << endl;
}
point2 &point2::operator=(const point2 &rhs)
{
    if(this != &rhs) {
        x = rhs.x;
        y = rhs.y;
        cout << "assigning point2: ";
        write(cout);
        cout  << endl;
    }
    return *this;
}
void point2::write(ostream &out) const
{
    out << x << ", " << y;
}

////////////////////// implementation of point3 ////////////////////////
```