DALHOUSIE UNIVERSITY
DEPARTMENT OF ENGINEERING MATHEMATICS
ENGM3282

ASSIGNMENT # 9, Due date: Tuesday November 20, 2018, 1:00 PM

1. In the program `figures.cpp` we have a base class `location` and various derived classes: `circle, triangle, rectangle`. Complete the implementation of the derived classes.

   For marking purposes, run the program entering: circle(1,2,3), triangle(3,4,1,2,1) and rectangle(5,6,3,4).

```
// File: figures.cpp

#include <cmath>
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

class location {
private:
    float x;  // position of the figure
    float y;

public:
    void read(istream& in) ;
    void write(ostream& out);
    float area(void); // returns 0
};

class circle : public location {
private:
    float radius;

public:
    void read(istream& in);
    void write(ostream& out);
    float area(void);      // area of the figure;
};

class rectangle : public location {
private:
    float side1, side2;

public:
    void read(istream& in);
    void write(ostream& out);
    float area(void);      // area of the figure;
};


class triangle : public rectangle {
private:
    float angle;

public:
    void read(istream& in);
    void write(ostream& out);
    float area(void);      // area of the figure;
};

int main()
{
    string type;                       // figure type
    ofstream fout ("figuresout.txt");

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

        if(type == "circle") {
            circle* p = new circle;
            p->read(cin);
            fout << "\nobject is a circle\n";
            p->write(fout);
            delete p;
        } else if (type == "triangle")  {
            triangle* p = new triangle;
            p->read(cin);
            fout << "\nobject is a triangle\n";
            p->write(fout);
            delete p;
        } else if (type == "rectangle") {
            rectangle* p = new rectangle;
            p->read(cin);
            fout << "\nobject is a rectangle\n";
            p->write(fout);
        } else break; // we are done entering data

    }

    return 0;
}

/////////////////// implementation of location /////// //////////

void location::read(istream& in)
{
    if(in == cin) cout <<"x coordinate: ";
    in >> x;

    if(in == cin) cout <<"y coordinate: ";
    in >> y;
}

float location::area(void)
{
    return 0.0;
}

void location::write(ostream& out)
{
    out << "x coordinate: " << x << "\n";
    out << "y coordinate: " << y << "\n";
    out << "area = " << area() << endl;

}

/////////////////// implementation of circle /////// //////////
```

```
////////////////// implementation of rectangle /////// ///////////
```

```
////////////////// implementation of triangle /////// ///////////
```

2. Try out the program `figurespointer.cpp` which uses the same classes and almost the same main program as the previous question.

   Run the program and enter the same figures as in the previous question. Explain why this program does not work properly.

   Change the classes (not main) so that the program runs properly and run the program with the same data as before.

```
// File: figurespointer.cpp

#include <cmath>
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

class location {
private:
    float x;  // position of the figure
    float y;

public:
    void read(istream& in) ;
    void write(ostream& out);
    float area(void); // returns 0
};

class circle : public location {
private:
    float radius;

public:
    void read(istream& in);
    void write(ostream& out);
    float area(void);      // area of the figure;
};

class rectangle : public location {
private:
    float side1, side2;

public:
    void read(istream& in);
    void write(ostream& out);
    float area(void);      // area of the figure;
};

class triangle : public rectangle {
private:
    float angle;

public:
    void read(istream& in);
    void write(ostream& out);
    float area(void);      // area of the figure;
};

int main()
{
    string type;                      // figure type
    ofstream fout ("figurespointerout.txt");
    location* p;

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

        if(type == "circle") {
            p = new circle;
            p->read(cin);
            fout << "\nobject is a circle\n";
            p->write(fout);
            delete p;
        } else if (type == "triangle")  {
            p = new triangle;
            p->read(cin);
            fout << "\nobject is a triangle\n";
            p->write(fout);
            delete p;
        } else if (type == "rectangle") {
            p = new rectangle;
            p->read(cin);
            fout << "\nobject is a rectangle\n";
            p->write(fout);
        } else break; // we are done entering data

    }
    fout.close();
    return 0;
}

///////////////// implementation of location /////// //////////

void location::read(istream& in)
{
    if(in == cin) cout <<"x coordinate: ";
    in >> x;

    if(in == cin) cout <<"y coordinate: ";
    in >> y;
}

float location::area(void)
{
    return 0.0;
}
```

```
void location::write(ostream& out)
{
    out << "x coordinate: " << x << "\n";
    out << "y coordinate: " << y << "\n";
    out << "area = " << area() << endl;

}

////////////////// implementation of circle //////////////////


////////////////// implementation of triangle //////////////////


////////////////// implementation of rectangle //////////////////
```

3. Try out the main program `figstack.cpp` which uses the STL `stack` class to implement a stack of `figures`. Explain why this program does not work properly.

   For marking purposes push the figures: circle(1,2,3), triangle(3,4,1,2,1) and rectangle(5,6,3,4) then pop them.

   Fix the main program and the class so that it does work. For marking purposes use the same data.

```
// File: figstack.cpp

#include <cmath>
#include <iostream>
#include <fstream>
#include <string>
#include <stack>

using namespace std;

class location {
private:
    float x;  // position of the figure
    float y;

public:
    void read(istream& in) ;
    void write(ostream& out);
    float area(void); // returns 0
};

class circle : public location {
private:
    float radius;

public:
    void read(istream& in);
    void write(ostream& out);
    float area(void);      // area of the figure;
};

class rectangle : public location {
private:
    float side1, side2;
```

```
public:
    void read(istream& in);
    void write(ostream& out);
    float area(void);      // area of the figure;
};

class triangle : public rectangle {
private:
    float angle;

public:
    void read(istream& in);
    void write(ostream& out);
    float area(void);      // area of the figure;
};

int main()
{
    ofstream fout("figstack.out");
    stack<location> mystack;
    char ch;       // response to prompt
    string type;   // type of figure
    location* ptr; // pointer to object pushed or popped

    while(1) {
        // print a little menu
        cout << "\n\np = push \n";
        cout << "o = pop\n";
        cout << "q = quit\n\n";

        cin >> ch;

        if (ch == 'p') {
            cout <<"\nEnter type of data to push : ";
            cin >> type;
            ptr = NULL;

            if(type == "circle") ptr = new circle;
            else if(type == "triangle") ptr = new triangle;
            else if (type == "rectangle") ptr = new rectangle;

            if(ptr != NULL) { // got a valid type
                ptr->read(cin);
                mystack.push(*ptr);
            }
        }

        if(ch == 'o') {
            if(mystack.empty()) cout << "stack is empty\n";
            else {
                ptr = &mystack.top();
                mystack.pop();
                cout << "popped:\n"; ptr->write(cout); cout << "\n";
                fout << "popped:\n"; ptr->write(fout); fout << "\n";
            }
        }

        if(ch == 'q') break;
    }
    fout.close();
    return 0;
}

////////////////// implementation of location /////// ///////////

void location::read(istream& in)
{
    if(in == cin) cout <<"x coordinate: ";
    in >> x;

    if(in == cin) cout <<"y coordinate: ";
    in >> y;
}

float location::area(void)
{
    return 0.0;
}

void location::write(ostream& out)
{
    out << "x coordinate: " << x << "\n";
    out << "y coordinate: " << y << "\n";
    out << "area = " << area() << endl;

}

////////////////// implementation of circle /////// ///////////



////////////////// implementation of rectangle /////// ///////////



////////////////// implementation of triangle /////// ///////////
```

4. In the following program the bank account classes from assignment 8 are used.

   (a) Run the program using the input file `transactionsin.txt`, choose the chequeing account when prompted. Does the program work properly?

   (b) Make the necessary methods in the `class account` virtual. Does the program work properly?

   (c) Make the necessary change to the `processtransactions` function prototype so that the program works properly.

```
// File: accountstransactions.cpp

#include <iostream>
#include <fstream>
#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);
};


void processtransactions(account x);

int main(void)
{
    char type;
    savingsaccount s("joe", 0.50, 0.03);
    chequingaccount c("bob", 0.50, 2);

    cout << "Which account to process? (c or s) :";
    cin >> type;

    if(type == 's') processtransactions(s);
    if(type == 'c') processtransactions(c);

    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 //////////////////////
// as per assignment 8



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

// as per assignment 8

void processtransactions(account x)
{
   ifstream fin("transactionsin.txt");
   char type;
   float amount;

   while(fin >> type >> amount) {
      if(type == 'w'){
          x.withdraw(amount);
          cout << "withdraw " << amount << "\n";
      }

      if(type == 'd') {
         x.deposit(amount);
         cout << "deposit " << amount << "\n";
      }

      x.write(cout);
```

```
        cout << "\n";
    }
}
```

5. The program `predprey.cpp` implements a predator–prey model. The program is in `solutions9.txt`.

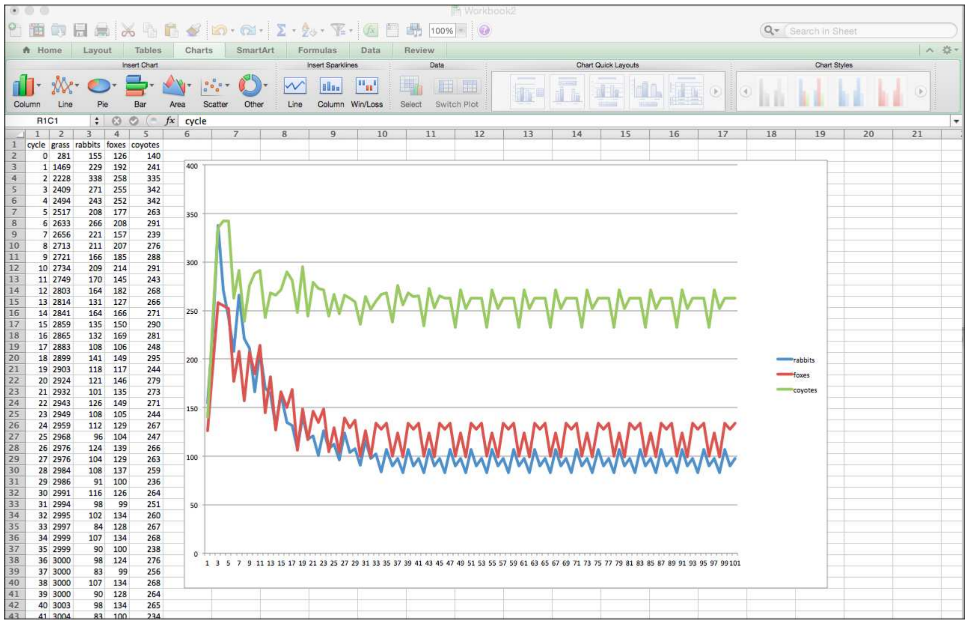
   Add a class `coyote` which interacts with both the `fox` and the `rabbit` classes. The following changes are needed:

   - Add the `class coyote` derived from `class creature`. Make the `who` method return the character `'c'`.
   - Add the static member `int coyote::lifespan = 4`.
   - Note that `int creature::count` can keep track of the number of coyotes by passing `type = 'c'`.
   - Modify `empty::next` to allow for `coyote` to appear at the top of the order where `fox` is currently.
   - Modify `fox::next` and `rabbit::next` to interact with `coyote`.
     - For a `fox`, in addition to the existing ways of dying, if the `coyote` count is greater than the `fox` count plus 1 then the `fox` dies.
     - For a `rabbit`, in addition to the existing ways of dying, if the `coyote` count plus the `fox` count is greater or equal to the `rabbit` count then the `rabbit` dies.
   - Give an implementation of `coyote::next` which is similar to the original `fox::next`.
   - Modify the constructor of `class world` to give 80% `empty`, 8% `grass`, 4% `rabbit`, 4% `fox` and 4% `coyote`.
   - Modify `world::display` to keep track of `coyotes`.

   For marking purposes, run your program for 5 cycles with a world size of 10 and submit the modified `predprey.cpp` program along with `ecosys.out`.

6. Run your previous program for 100 cycles with a world size of 60 and use Excel or some other plotting program to plot the rabbit, fox and coyote populations against cycle. Estimate the long run (average) populations of rabbit, fox and coyote.

   Here is the result I got using Excel to plot populations in `summaryout.txt` (yours may look quite different):



cycle	grass	rabbits	foxes	coyotes
0	281	155	126	140
1	1469	229	192	241
2	2228	338	258	335
3	2409	271	255	342
4	2494	243	252	342
5	2517	208	177	263
6	2633	266	208	291
7	2656	221	157	239
8	2713	211	207	276
9	2721	166	185	288
10	2734	209	214	291
11	2749	170	145	243
12	2803	164	182	268
13	2814	131	127	266
14	2841	164	166	271
15	2859	135	150	290
16	2865	132	169	281
17	2883	108	106	248
18	2899	141	149	295
19	2903	118	117	244
20	2924	121	146	279
21	2932	101	135	273
22	2943	126	149	271
23	2949	108	105	244
24	2959	112	129	267
25	2968	96	104	247
26	2976	124	139	266
27	2976	104	129	263
28	2984	108	137	259
29	2986	91	100	236
30	2991	116	126	264
31	2994	98	99	251
32	2995	102	134	260
33	2997	84	128	267
34	2999	107	134	268
35	2999	90	100	238
36	3000	98	124	276
37	3000	83	99	256
38	3000	107	134	268
39	3000	90	128	264
40	3003	98	134	265
41	3004	83	100	234