DALHOUSIE UNIVERSITY
DEPARTMENT OF ENGINEERING MATHEMATICS
ENGM3282: DATA STRUCTURES AND NUMERICAL METHODS

ASSIGNMENT # 4, Due date: Tuesday, October 9, 2018, 1:00 PM

1. Complete the implementation of the classes `circle` and `cylinder` as decleared in the program `cylinder.cpp`. Your output should look like:

```
p = (1,2)
cir = (centre = (1,2), radius = 3)
cyl = (base = (centre = (1,2), radius = 3), height = 7)
```

```
/*  File: cylinder.cpp
    classes which represents points, circles and cylinders in the plane */

#include <iostream>
#include <fstream>

using namespace std;

class point {
private:
    float x;
    float y;

public:
    point(float h, float v);            // x = h and y = v

    void write(ostream &out) const;     // write a point in the form (x,y)
};

class circle {
private:
    point centre;
    float radius;

public:
    circle(const point &p, float r);

    void write(ostream &out) const;
};

class cylinder {
private:
    circle base;
    float height;

public:
    cylinder(const circle &c, float ht);

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

```
int main(void)
{
    point p(1.0, 2.0);

    cout << "p = ";
    p.write(cout);
    cout << endl;

    circle cir(p, 3.0);

    cout << "cir = ";
    cir.write(cout);
    cout << endl;

    cylinder cyl(cir, 7.0);
    cout << "cyl = ";
    cyl.write(cout);
    cout << endl;

    return 0;
}

////////////////////////////// implementation of point ///////////////////////

point::point(float h, float v) // x = h and y = v
{
    x = h;
    y = v;
}

void point::write(ostream &out) const // write a point in the form (x,y)
{
    out << "(" << x << "," << y << ")";
}


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









////////////////////////////// implementation of cylinder ///////////////////////
```

2. In the following program an employee of a company is represented by an object of type `employee` consisting of a name, employee id, employee salary and the workday starting time. The starting time is a `class time24` object.

   Implement the `class employee`.

   For marking purposes your output should look like:

```
x =  {Jones, 123, 45000, 08:00}
y =  {Smith, 124, 50000, 08:30}
```

```
/*  File: employeetime.cpp
    Illustrates composition of classes
*/

#include <iostream>
#include <string>

using namespace std;

class time24 {
private:
    int hour_;                          // hour between 0 and 23
    int minute_;                        // minute between 0 and 59

public:
    time24(int h, int m);               // constructor

    void write(ostream &out) const;     // prints hour:minute to out
};

class employee {
private:
    string name;  // last name
    int id;       // employee id number
    float salary; // employee salary
    time24 start; // workday start time

public:
    employee(string n, int i, float s, const time24 &t);
    employee(string n, int i, float s, int h, int m);
    void write(ostream &out) const;
};
int main(void)
{
    time24 u(8, 0);
    employee x("Jones", 123, 45000.0, u);
    employee y("Smith", 124, 50000.0, 8, 30);

    cout << "x =  "; x.write(cout); cout << endl;

    cout << "y =  "; y.write(cout); cout << endl;

    return 0;
}
```

```
//////////////////// Implementation of time24 ////////////////////////////

time24::time24(int h, int m)
{
   hour_ = h;
   minute_ = m;
}

void time24::write(ostream &out) const
{
   out << hour_ /10 << hour_ % 10 << ":" << minute_ / 10 << minute_ %10;
}

//////////////////////// implementation of employee ////////////////////
```

3. In the following program an employee of a company is represented by an object of type `employee` consisting of a name, employee id and employee salary. The `class employee` has methods read() and write().

   Information about a company's workforce is stored in an object of type `workforce` consisting of the workforce size and a dynamically allocated array of employees. The methods read() and write() are defined for workforce.

   Do not modify the main program. Complete the program by writing the constructor, destructor, copy constrctor, operator=, read and write for the class workforce. The constructor must allocate the array of employees and the destructor must delete it.

   For marking purposes, use the input file `workforcein.txt` and run your program twice, once choosing to copy and the other choosing to assign.

```
/*  File: workforce.cpp

    An employee's information is stored as an object of type employee
    consisting of name, employee id and employee salary.  The methods
    read() and write() are defined for employee.

    Information about a company's workforce is stored in an object of type
    workforce consisting of the workforce size and a dynamically allocated
    array of employees.  The methods read() and write() are defined for
    workforce.

    The read() and write() methods for workforce uses the read() and write()
    methods for employee

    Programmer: your name             Date:
*/

#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 = "nobody", int i = 0, float s = 0);

    void read(istream &in);
    void write(ostream &out) const;
};

class workforce {
private:
   int size;           // size of workforce
   employee* list; // array of employees

public:
    workforce(int n);   // set size = n and allocate an array of employees
    // big 3 prototypes here



    void read(istream &in);
    void write(ostream &out) const;
};

int main(void)
{
    workforce w(5);
    ifstream fin("workforcein.txt"); // file containing the list of employees
    ofstream fout("workforceout.txt");
    char ch;

    w.read(fin);
    fout << "\nw = \n";
    w.write(fout);

    cout << "C to copy otherwise assign: ";
    cin >> ch;

    if(ch == 'C') {
        workforce v(w);     // v's scope is the body of the if
        fout << "\nv = \n";
        v.write(fout);
                            // v is destroyed here
    } else {
        workforce u(10);    // u's scope is the body of the else
        u = w;
        fout << "\nu = \n";
        u.write(fout);
                            // u is destroyed here
    }
```

```
    fout << "\nw = \n";     // check whether w is still intact
    w.write(fout);

    fin.close();
    fout.close();

    return 0;
}

////////////////////// implementation of employee ////////////////////////
employee::employee(string n, int i, float s)
{
    name = n;
    id = i;
    salary = s;
}

void employee::read(istream &in)
{
    in >> name >> id >> salary;
}

void employee::write(ostream &out) const
{
    out << name << " " << id << " " << salary << endl;
}

////////////////////// implementation of workforce //////////////////////
```

4. Matrices can be represented as a class in C++. Instead of using a 2 dimensional array to store the matrix entries it is much more efficient to use a 1 dimensional array where the columns of the matrix are stored in the array one after another (see below for a picture).

   In the class shown below some of the methods of `class matrix` have been written.

   Do not modify the main program. **Complete the program by writing the big 3 and the matrix operators.**

   Note that the C header file `cstdlib` contains the `exit()` function which should be used to halt a program if the matrix sizes for matrix operations are not compatible.

   Note that the C++ header file `iomanip` should be used to make the columns of a matrix line up nicely when printed. For example, if we want a numeric variable `x` to appear in the stream `out` with a width of 6 then we send an iomanip object to the stream just before sending `x` as follows:

   ```
   out << setw(6) <<  x;
   ```

   For marking purposes, run your program using the input file `matricesin.txt`.

   Your output should look like:

```
The copy of a is
     1.5     2.7
     3.1     4.4

The sum of
     1.5     2.7
     3.1     4.4
and
     3.2     4.5
     1.1     2.9
is
     4.7     7.2
     4.2     7.3

The product of
     1.5     2.7
     3.1     4.4
and
     1.9     1.2       0     2.7
     2.9     3.6     2.1     1.1
is
   10.68   11.52    5.67    7.02
   18.65   19.56    9.24   13.21

/* File: matrices.cpp
   represent a matrix as a class with operator overloading*/

#include <iostream>
#include <fstream>
#include <cstdlib>
#include <iomanip>

using namespace std;

class matrix {
private:
    int rows_;       // number of rows
    int cols_;       // number of columns
    float* elements; // pointer to a 1 dimensional array which stores the
                     // columns of the matrix one after another
/*
     the matrix                      the array elements

+-               -+         [a00,a10,a20,...,a01,a11,a21,...,a02,a12,a22,...]
| a00 a01 a02 ... |              ^               ^               ^
| a10 a11 a12 ... |              |               |               |
| a20 a21 a22 ... |              |               |               |
|  .   .   .  ... |              |               |               |
|  .   .   .  ... |           column 0        column 1        column 3
+-               -+

*/

public:
```

```
    matrix(int m=1, int n=1, float s = 0.0);  // constructor
    ~matrix(void);                            // destructor
    matrix(const matrix &other);              // copy constructor
    matrix &operator=(const matrix &rhs);     // operator=

    int rows(void) const;                     // returns rows_
    int cols(void) const;                     // returnc cols_
    float at(int i, int j) const;             // returns the (i,j) entry
    float &at (int i, int j);                 // returns the (i,j) entry
};

// prototypes for matrix operators




int main(void)
{
    ifstream fin("matricesin.txt");
    ofstream fout("matricesout.txt");

    matrix a(2,2), b(2,2), c(2,4);

    fin >> a >> b >> c;

    // try the copy constructor
    matrix d(a);

    fout << "The copy of a is\n";
    fout << d << "\n";

    d = a + b; // check out operator= and operator+
    fout << "The sum of \n" << a << "and\n" << b << "is\n" << d;

    d = a * c; // check out operator= and operator*
    fout << "\nThe product of \n" << a << "and\n" << c << "is\n" << d;

    fin.close();
    fout.close();

    return 0;
}

////////////////////////// implementation of matrix ////////////////////////

// constructor, fill all values with s, default 1x1, s = 0.0
matrix::matrix(int m, int n, float s)
{
    int i;

    rows_ = m;
    cols_ = n;
    elements = new float[m*n];
```

```
    for(i = 0; i < m*n; i++) {
        elements[i] = s;
    }
}

// implementation of the big 3




// returns the (i,j) entry of the matrix by reference
float &matrix::at(int i, int j)
{
    if( (i >= rows_) || (j >= cols_) ){
        cerr << "subscript out of bounds\n";
        exit(1);
    }

    return elements[ i +j*rows_ ];
}

// returns the (i,j) entry of the matrix
float matrix::at(int i, int j) const
{
    if( (i >= rows_) || (j >= cols_) ){
       cerr << "subscript out of bounds\n";
       exit(1);
    }

    return elements[ i +j*rows_ ];
}

int matrix::rows(void) const
{
    return rows_;
}

int matrix::cols(void) const
{
    return cols_;
}

////////////////////////////// matrix operators /////////////////////////////
```