Exercise 1: C++ Classes

Exercise 1.a

The following class (Fig. 1) declares a 2-D point data-type:

```
#ifndef EX1A_H
#define EX1A_H

// class declaration
class point {

    float x, y;                // point coordinates

public:
    point (float, float);       // Constructor
    void move (float, float);
    void display (PinName, PinName);
};


#endif
```

Fig. 1 Class declaration in file *ex1a.h*

As can be seen in Fig. 2, *point* class definition is not finished:

```
#include "mbed.h"
#include <iostream>
#include <cmath>
#include "exla.h"


// constructor
point::point (float abs, float ord){

}

// move method
void point::move (float dx, float dy){

}

// display method
void point::display(PinName pinX, PinName pinY){

    DigitalOut ledX(pinX), ledY(pinY);

    cout << "Point = (" << x << " , " << y << ")\r" << endl;
    ledX = 1;
    ledY = 1;
    if ( abs(x) < abs(y) ){
        wait( abs(x) );
        ledX = 0;
        wait( abs(y)- abs(x) );
        ledY = 0;
    }
    else{
        wait( abs(y) );
        ledY = 0;
        wait( abs(x)-abs(y) );
        ledX = 0;
    }
}
```

Fig. 2 Uncompleted class definition in file *ex1a.cpp*

Complete the definition of the class constructor and method *move*. Note that the former initializes a *point* object and the latter results in x and y displacements of a *point* object.

Additionally, the method *display* outputs information about a *point* object in two different ways. First, the abscissa and ordinate of a point is displayed in textual form through the serial terminal using the standard C++ library. Secondly, two LEDs are assigned and light up when applied to an object. The period of time each LED is illuminated equals the values of the abscissa and ordinate. Consequently, the x-y plane is divided in two ‘sand-clock’ shaped regions (Fig. 3). One - the horizontal ‘sand-clock’ - in which the LED allocated to the abscissa lights longer than the one allocated to the ordinate and a complementary one - the vertical ‘sand-clock’ - in which the LED allocated to the ordinate lights longer than that of the abscissa.

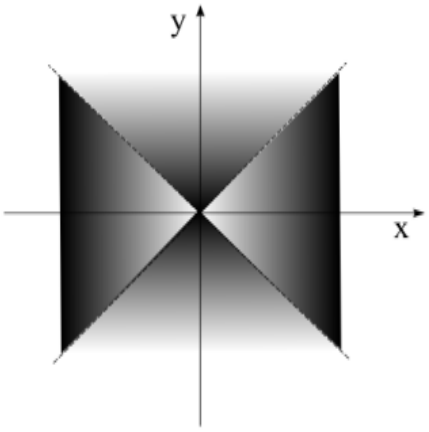

Fig. 3 x-y plane division into two sand-clock shaped regions

```
#include "mbed.h"
#include <iostream>
#include "exla.h"


// main function
int main(){
    cout << "static point created \r" << endl;
    point p (-1.0, 2.0);
    p.display(LED3, LED4);
    wait(1.0);

    cout << "moving the static point in x and y \r" << endl;
    p.move (3.0, -1.0);
    p.display(LED3, LED4);
    wait(1.0);

    cout << "dynamic point created \r" << endl;
    point *pt1;
    pt1 = new point(-3.0, -1.0);
    pt1->display(LED3, LED4);
    delete pt1;
    cout << "dynamic point removed \r" << endl;

    return 0;
}
```

Fig. 4 *Point* objects declarations and method applications in file *main.cpp*

Modify the statements declared in main.cpp (Fig. 4) to check new point initialization values. Answer the following questions.

Reference	Question/answer
Ex. 1-a	Should the class constructor return any value?

Section 1	
	Why <iostream> and <cmath>, the standard C++ libraries, are included in file ex1a.cpp
Ex. 1-a	What are the equivalent functions in C to **new** and **delete**?
Section 2	
Ex. 1-a	Why the macro **#ifndef** appears within the class declaration?

Section 3	

Exercise 1.b

In this exercise the previous class *point* is going to be modified. The goals are:

(1) to allow point object declarations of the type

```
point p1;
point p2(3.0);
```

(2) to include a method that returns the number of points created, and

(3) to create a class destructor.

Complete the new declaration of the class *point* (Fig. 5) and the definition (Fig. 6) of the different class constructors, the class destructor and the methods *move* and *pointNumber*.

```
#ifndef EX1B_H
#define EX1B_H

// class declaration
class point {
  float x, y;                    // point coordinates
  static int pointCounter;       // to count the number of points
 public:
  point (void);                  // constructor 0 arguments
  point (float);                 // constructor 1 argument
  point (float, float);          // constructor 2 arguments
  ~point(void);                  // destructor
  void move (float, float);
  void display (PinName, PinName);

};


#endif
```

Fig. 5 Uncompleted class declaration in file *ex1b.h*

It is worth noting the *static* qualifier of the integer class attribute *pointCounter*. When some kind of information needs to be collectively updated and shared by the set of currently existing objects of a class, the corresponding class attribute needs to be declared 'statically'. Briefly speaking, the class attribute has the same value along the objects of the class.

```cpp
#include "mbed.h"
#include <iostream>
#include <cmath>
#include "ex1b.h"


// static variable initialization
int point::pointCounter = 0;

// constructor 0 arguments
point::point (void){

}
// constructor 1 argument
point::point (float value){

}
// constructor 2 arguments
point::point (float abs, float ord){

}
// destructor
point::~point(void){

}
// move method
void point::move (float dx, float dy){

}
// display method
void point::display(PinName pinX, PinName pinY){

    DigitalOut ledX(pinX), ledY(pinY);

    cout << "Point = (" << x << " , " << y << ")\r" << endl;
    ledX = 1;
    ledY = 1;
    if ( abs(x) < abs(y) ){
        wait( abs(x) );
        ledX = 0;
        wait( abs(y)- abs(x) );
        ledY = 0;
    }
    else{
        wait( abs(y) );
        ledY = 0;
        wait( abs(x)-abs(y) );
        ledX = 0;
    }
}
// method to count the number of points
int point::pointNumber(void){

}
```

Fig. 6 Uncompleted class definition in file *ex1b.cpp*

```
#include "mbed.h"
#include <iostream>
#include "ex1b.h"


// main function
int main(){
    cout << "static points A and B created \r" << endl;
    point A (1.5, 2.5);
    A.display(LED3, LED4);
    wait(1.0);

    point B;
    B.display(LED3, LED4);
    wait(1.0);
    cout << "number of points = " << A.pointNumber() << "\r" << endl;

    cout << "dynamic point created \r" << endl;
    point* pointPointer;
    pointPointer = new point(3.0);
    pointPointer->display(LED3, LED4);
    wait(1.0);
    cout << "number of points (before delete)=" <<  B.pointNumber() << "\r"
         << endl;

    cout << "dynamic point removed \r" << endl;
    delete pointPointer;
    cout << "number of points (after delete)=" <<  B.pointNumber() << "\r"
         << endl;

    return 0;
}
```

Fig. 7 *Point* object declarations and point counting in file *main.cpp*

Add new point declarations to the statements declared in main.cpp (Fig. 7) to show how the number of points counted is updated. Answer the following questions.

Reference	Question/answer
Ex. 1-b Section 1	Is the constructor an overloaded method in this class?

Ex. 1-b Section 2	What is the effect of the **static** modifier in the declaration of the integer variable **pointCounter**?
Ex. 1-b Section 3	Does the class need a destructor to compile correctly?

Exercise 1.c

In this exercise the class *point* is again modified to add a method to compare 2 points. A point A (x, y) is defined as greater than a point B (u, v) if x + y is greater than u + v. The method shall return 0, when the points

are equal; -1, when the calling point is greater than the point passed as parameter; and, 1, when the point passed as parameter is greater than the calling point.

Complete the class *point* declaration (Fig. 8) and the definition of the method *compare* (Fig. 9)

```
#ifndef EX1C_H
#define EX1C_H

// class declaration
class point {
  float x, y;                    // point coordinates
  static int pointCounter;       // to count the number of points
 public:
  point (void);                  // constructor 0 arguments
  point (float);                 // constructor 1 argument
  point (float, float);          // constructor 2 arguments
  ~point(void);                  // destructor
  void move (float, float);
  void display (PinName, PinName);
  int compare(const point&);     // comparison method
};


#endif
```

Fig. 8 Uncompleted class declaration in file *ex1c.h*

```
#include "mbed.h"
#include <iostream>
#include <cmath>
#include "ex1c.h"


// static variable initialization
int point::pointCounter = 0;

// constructor 0 arguments
point::point (void){

}
// constructor 1 argument
point::point (float value){

}
// constructor 2 arguments
point::point (float abs, float ord){

}
// destructor
point::~point(void){

}
// move method
void point::move (float dx, float dy){

}
// display method
void point::display(PinName pinX, PinName pinY){

    DigitalOut ledX(pinX), ledY(pinY);

    cout << "Point = (" << x << " , " << y << ")\r" << endl;
    ledX = 1;
    ledY = 1;
    if ( abs(x) < abs(y) ){
        wait( abs(x) );
        ledX = 0;
        wait( abs(y)- abs(x) );
        ledY = 0;
    }
    else{
        wait( abs(y) );
        ledY = 0;
        wait( abs(x)-abs(y) );
        ledX = 0;
    }
}
// method to count the number of points
int point::pointNumber(void){

}
// method to compare two points
int point::compare(const point& P){

}
```

Fig. 9 Uncompleted class definition in file *ex1c.cpp*

Add new point declarations to exercise the comparison between points in file *main.cpp* (Fig. 10).

```cpp
#include "mbed.h"
#include <iostream>
#include "exlc.h"


// main function
int main(){
    cout << "static points A and B created \r" << endl;
    point A (1.5, 2.5);
    A.display(LED3, LED4);
    wait(1.0);

    point B (2.0, 1.5) ;
    B.display(LED3, LED4);
    wait(1.0);
    cout << "number of points = " << A.pointNumber() << "\r" << endl;

    cout  << "comparing point B to point A = " << B.compare(A)
          << "\r" << endl;

    return 0;
}
```

Fig. 10 Point declaration and comparison in *main.cpp*

Answer the following questions.

Reference	Question/answer
Ex. 1-c Section 1	Could you explain why the **&** is used in the parameter declaration of the **compare** method?
	And, the key-word **const**?

Exercise 1.d

Modify the class point to add a method that compares 2 points and returns the greatest one.

Complete the class *point* declaration (Fig. 11) and the definition of the method *maximum* (Fig. 13)

```
#ifndef EX1D_H
#define EX1D_H

// class declaration
class point {
  float x, y;                    // point coordinates
  static int pointCounter;       // to count the number of points
 public:
  point (void);                  // constructor 0 arguments
  point (float);                 // constructor 1 argument
  point (float, float);          // constructor 2 arguments
  ~point(void);                  // destructor
  void move (float, float);
  void display (PinName, PinName);

  int compare(const point&);     // comparison method
  point maximum (const point&);  // greatest of two points
};

#endif
```

Fig. 11 Uncompleted class declaration in file *ex1d.h*

Note how the pointer *this* is employed in *maximum* to call the method *compare.* Add new point declarations to exercise the comparison between points in file *main.cpp* (Fig. 12).

```
#include "mbed.h"
#include <iostream>
#include "exld.h"


// main function
int main(){
    cout << "static points A and B created \r" << endl;
    point A(1.5, 2.5);
    A.display(LED3, LED4);
    wait(1.0);

    point B(2.0, 1.5);
    B.display(LED3, LED4);
    wait(1.0);

    cout << "static point C is created (default constructor) \r" << endl;
    point C;
    C.display(LED2, LED4);
    wait(1.0);

    cout  << "point C is reassigned to the greatest of points A and B"
          << "\r" << endl;

    C = B.maximum(A);
    C.display(LED3, LED4);

    return 0;
}
```

Fig. 12 Point declaration and maximum calculation in *main.cpp*

```
#include "mbed.h"
#include <iostream>
#include <cmath>
#include "ex1d.h"


// static variable initialization
int point::pointCounter = 0;

// constructor 0 arguments
point::point (void){

}
// constructor 1 argument
point::point (float value){

}
// constructor 2 arguments
point::point (float abs, float ord){

}
// destructor
point::~point(void){

}
// move method
void point::move (float dx, float dy){

}
// display method
void point::display(PinName pinX, PinName pinY){

    DigitalOut ledX(pinX), ledY(pinY);

    cout << "Point = (" << x << " , " << y << ")\r" << endl;
    ledX = 1;
    ledY = 1;
    if ( abs(x) < abs(y) ){
        wait( abs(x) );
        ledX = 0;
        wait( abs(y)- abs(x) );
        ledY = 0;
    }
    else{
        wait( abs(y) );
        ledY = 0;
        wait( abs(x)-abs(y) );
        ledX = 0;
    }
}
// method to count the number of points
int point::pointNumber(void){

}
// method to compare two points
int point::compare(const point& P){

}
// method to compute the maximum of two points
point point::maximum(const point & P){
  int result;

  result=this->compare(P);
}
```

Fig. 13 Uncompleted class definition in file *ex1d.h*

Answer the following questions.

Reference	Question/answer
Ex. 1-d Section 1	The pointer *this*, where does it point to?

	In the **main** function, the operator = is applied to a pair of points, where is it defined?