Exercise 2
Class and Object Manipulations

Duration : 90 minutes (including Part I and Part II, as well as the submission)

Objectives
This exercise aims at implementing the following concepts:
- Operator overloading
- Passing objects as parameters to functions
- Friend functions

Exercise Materials
- Program templates are provided for this exercise. Please download from the elearning and extract the ZIP file to your local drive.
- You have the choice to use Microsoft **VS Code**, **Dev C++** or any other IDEs to write the code for this exercise.
- If you choose VS Code, use debug **"Console program"** to run the program.
- If you use Dev C++, you do not have to work in a folder. Instead, you open directly the source file.

Deliverable Item
- Only the **source code** file is needed for the submission (i.e., **exercise2.cpp**).
- You must submit your source code at elearning.

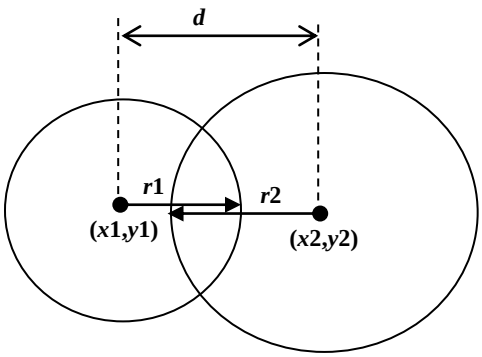
Plagiarism Policy
- Discussions among the students are still possible during the exercise session.
- However, all works must be done individually.
- Any kind of plagiarism (e.g., copy and paste code by any mean) would lead to disqualification of submissions for both parties (i.e., students that copy others' code and students that give their code to others).

Late Submission Policy
- **5% deduction for every 5 minutes late**.
- For example: if the duration of the exercise is 90 minutes, and your submission is received on the 91st minute, you are only eligible to earn 95% at maximum of the total marks.

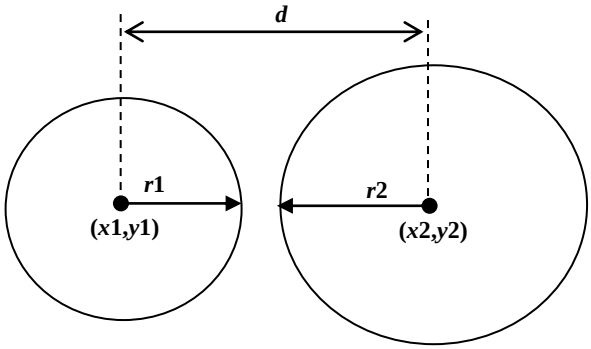
The Problem or Case Study

You will be writing a program which is able to determine whether an object overlaps to another object. For simplicity, you will be working on circle objects, as the calculation for the overlap between circles is much easier. Example calculation is given as follows:

Given two circles with their center coordinates *x* and *y*, as well as the radius, *r*, as shown in the following figure:



a): The circles overlap



b): The circles do not overlap

The distance between the two circles, ***d***, can be determined by the pythagoras' theorem:

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Then, the circles are identified overlap if the distance between them is less than or equal to the sum of their radius:

$$overlap\ if\ d \leqslant (r1 + r2)$$

Tasks

In this exercise, you will be writing a program for the above case study using the programming elements stated in the objectives section.

The exercise is divided into two parts:

- Part 1 consists of guided tasks, in which the questions or tasks are accompanied with the code segment for the solution. This part will not be assessed. However, this part is a pre-requisite for the next part.
- Part 2 consists of tasks that you need to do it yourself. However, Part 1 should give you an idea to accomplish the tasks in Part 2. **Only this part will be graded** for this exercise.

Part 1: Guided Tasks

You should spend about 40 to 50 minutes to complete this part. This part will not be graded. However, it is a pre-requisite for the next part.

1. Before getting started, let's take a look how the program is written in procedural programming (i.e., using regular functions). Open and analyse the program **overlap_procedural.cpp**.

2. Now, let's get started writing the code using OOP approach. Modify the program **exercise2.cpp.** You start off by creating a class named `Circle`. The code for the class attributes, constructor, accessor and mutator methods are given (except the method `set()`). Add another mutator method named `set` to the class, which sets all the attributes. Copy only the highlighted lines from the code segment below.

```
//-----------------------------------------------
// Class declaration
//-----------------------------------------------
class Circle
{
  private:
    double x, y, r;
  public:
    Circle(int _x = 0, int _y = 0, int _r = 0);
    double getX() const;
          …….
          …….
    void setR(double value);

    void set(double _x, double _y, double _r);    // copy only this line
};
```

```
//-----------------------------------------------
// Class definition
//-----------------------------------------------

Circle::Circle(int _x, int _y, int _r) : x(_x), y(_y), r(_r)
{}

double Circle::getX() const { return x; }
    …….
    …….
void Circle::setR(double value) { r = value; }

void Circle::set(double _x, double _y, double _r)
{
    x =_x;
    y =_y;
    r =_r;
}
```

3. Next, is to implement **operator overloading**. Here you will be **redefining** the minus operator such that when a circle substracts another circle, the result will be the distance between these circles. Copy the highlighted lines below.

```
//-------------------------------------------------
// Class declaration
//-------------------------------------------------
class Circle
{ private:
    double x, y, r;

  public:
    Circle(int _x = 0, int _y = 0, int _r = 0);
    …
    …

    double operator- (Circle right);   // add this declaration
};
```

```
//------------------------------
// Class definition
//------------------------------
// add the following definition code for the overloaded minus operator

double Circle::operator- (Circle right)
{
    double dx = x - right.x;
    double dy = y - right.y;
    double dist = sqrt(dx*dx + dy*dy);
    return dist;
}
```

4. Next, is to implement **passing objects as parameters to functions**. Here you will be defining a function named `overlap` which accepts two `Circle` objects as parameters and return a boolean value indicating whether the circles overlap. Copy the highlighted lines below.

```
//-------------------------------------------
// Function declaration
//-------------------------------------------
// add the following function declaration on top (before the main function)

bool overlap(Circle c1, Circle c2);
```

```
//-----------------------------------------------------------
// Function definition
//-----------------------------------------------------------
// add the function definition at the bottom of the program

bool overlap(Circle c1, Circle c2)
{
    double d = c1 - c2; // The overloaded operator minus is invoked here,
                        //  to get the distance between the circles.
    return d <=(c1.getR() + c2.getR());
}
```

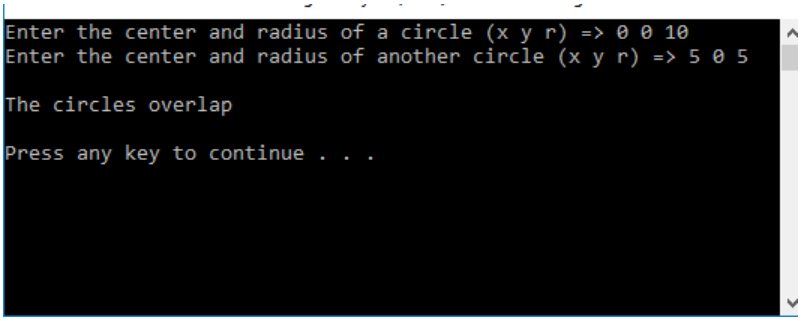
5. Next, is to test the class in the **main function**. Complete the code for the main function such that it creates two `Circle` objects and sets the attributes of these circles with inputs read from the keyboard. Then, it prints the output onto the screen showing whether the two circles overlap. Copy only the highlighted lines from the following code segment.

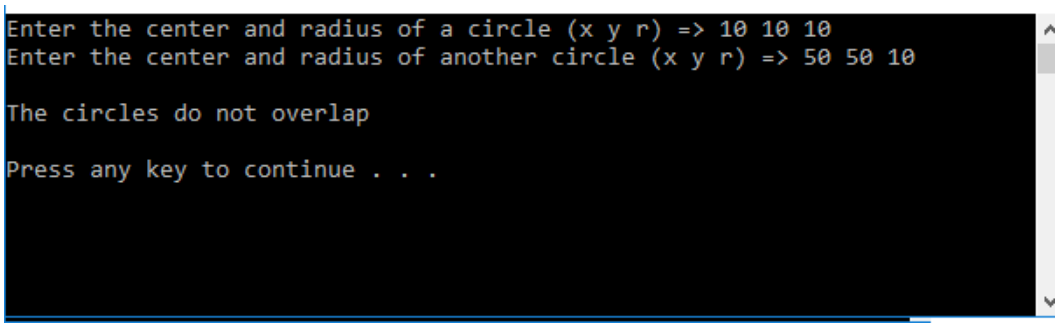
```
int main()
{
   ….
   ….
    cout << "Enter the center and radius of a circle (x y r) => ";
    cin >> x >> y >> r;
    firstCircle.set(x,y,r);

    cout << "Enter the center and radius of another circle (x y r) => ";
    cin >> x >> y >> r;
    secondCircle.set(x,y,r);
    cout << endl;

    if ( overlap(firstCircle, secondCircle) )
        cout << "The circles overlap" << endl;
    else
        cout << "The circles do not overlap" << endl;
   ….
   ….
    return 0;
}
```

Run the program. You should get the following output:





6. Next, is to implement **friend functions**. Rearrange the declaration of the function `overlap` so that it can access the private members of the circles. Also, rewrite the definition of the function such that the objects `c1` and `c2` now access directly to the attributes, rather than using the accessor methods. Modify only the highlighted lines accordingly.

```
//---------------------------------------------------------
// Class declaration
//---------------------------------------------------------
class Circle
{
  private:
    double x, y, r;

  public:
    Circle(int _x = 0, int _y = 0, int _r = 0);
    …
    …
   // move the declaration line of function 'overlap' to the inside of
   //  the class declaration and declare the function as 'friend'

    friend bool overlap(Circle c1, Circle c2);

    // 'overlap' is not a member to the class but a regular function. However
    //  it has direct accesses to the private members as it is delcared
    //  as a friend to the class.
};
```

```
//-----------------------------------------------------------
// Function definition
//-----------------------------------------------------------
bool overlap(Circle c1, Circle c2)
{
    double d = c1 - c2;       // The overloaded operator minus is invoked here
                              //  To get the distance between the circles.

    return d <=(c1.r + c2.r);  // this line has been rewritten such that c1 and c2 access
                               //   directly to r, rather than using the accessor getR()
    // c1 and c2 are able to access directly to the private member r,
    // as the function 'overlap' has been declared as a friend to class 'Circle'
}
```

7. Before moving to Part 2, revise all the steps you have done in Part 1. Try to understand the the followings:
    a. How to declare and define an overloaded operator. Also, how to call to the overloaded operator to execute it.
    b. How to pass objects as parameters to a function.
    c. What a friend function is, and how to declare a function to be a friend of a class.

Part 2: Graded Tasks

You should spend about 30 to 40 minutes to complete this part. Only this part will be graded for this exercise.

8. Add another **overloaded operator**, for the times `operator *`. Define this operator such that when a circle is multiplied with another circle, the result would be a boolean value indicating whether the circles overlap. This operator actually behaves like the function `overlap`. ***Notes:*** you may want to refer to the `operator -` in Part 1 in order to accomplish this task.

   Then, in the main function, modify the highlighted line below such that it uses the overloaded `operator *` instead of the function `overlap`.

```
    if ( overlap(firstCircle, secondCircle) )
        cout << "The circles overlap" << endl;
    else
        cout << "The circles do not overlap" << endl;
```

9. Define another **friend function** named `inputCircle` that reads user inputs for the attributes, `x`, `y` and `r`. This function should accept a circle object as parameter sent by reference. The function should look like below:

```
void inputCircle(Circle &c)
{
    // code for this function goes here
}
```

   Then, in the main function, modify the highlighted lines below such that it uses the function `inputCircle` to read user inputs and set the circles' attributes.

```
    cout << "Enter the center and radius of a circle (x y r) => ";
    cin >> x >> y >> r;
    firstCircle.set(x,y,r);

    cout << "Enter the center and radius of another circle (x y r) => ";
    cin >> x >> y >> r;
    secondCircle.set(x,y,r);
```