

EE3093: C++ Assignment

UNIVERSITY OF ABERDEEN

SESSION 2020-21

EE3093: Cpp Assignment

Degree Examination in EE3093 C/C++ PROGRAMMING

First Half Session: 2020-21

Release Date: 1 / 12 / 2020

Submission Date: 7 / 12 / 2020

This is an open book online assessment which is expected to take up to 3 hours to complete but submission will be accepted, without incurring any penalties, until the above submission deadline; students who have not submitted after this time will be required to present for reassessment in the resit diet.

Please follow the instructions and answer the questions accordingly. Where requested, please upload files through MyAberdeen as instructed. Information on how to use your smartphone or tablet to scan your work and upload on MyAberdeen is available at https://abdn.blackboard.com/bbcswebdav/xid-19374611_1 and also at https://abdn.blackboard.com/bbcswebdav/xid-19374612_1.

You **must not** attempt to communicate with any candidate regarding this exam by any means including but not limited to, electronically, orally or passing/showing written material to another candidate and you **must not** attempt to view another candidate's work. All submissions will be checked for plagiarism.

The weighting of this assessment is 30%

PLEASE READ AND FAMILIARISE YOURSELF WITH THE REGULATIONS DETAILED ON NEXT PAGE BEFORE ATTEMPTING THIS EXAMINATION

Assessment weighting

The weighting of the alternative assessment for each course is as stated in the assessment specification or exam paper.

For details of the performance descriptors associated with each CGS grade, please refer to the Common Grading Scale site <https://www.abdn.ac.uk/staffnet/teaching/common-grading-scale-2840.php>.

Submission of work for assessment

Each student must submit their own solution or report on the assessment *via* MyAberdeen by the deadline stated on the exam paper or on the assignment specification.

Late penalties as described in the School of Engineering Student Handbook will apply for any submissions submitted beyond the submission deadline.

The submission deadlines are stated in the specification for each alternative assessment. Students who submit their solution or report on the assessment after the stated deadline will be required to present for assessment in the resit diet.

If illness, injury, IT/internet related problem, or any other condition prevents you from completing an assessment, coursework, or examination, or if you consider that your performance may have been detrimentally affected having taken the exam or done the assignment, then you must complete the Absence form on MyAberdeen.

The Absence form is available on MyAberdeen, and must be completed within **three** working days of the submission deadline of the alternative assessment.

Plagiarism, Collusion and Contract Cheating

Your solution to the alternative assessment **must** be entirely your own personal work or, where the intellectual work of others is used, it must be clearly identified and referenced. This applies to everything you submit for assessment. Copying someone else's work, colluding with someone else in doing the set exercise or exam, and plagiarism will be regarded as academic misconduct.

The rules approved by the University for the conduct of examinations are set out in a separate document entitled "Rules for the Conduct of Written Examinations for Degrees and Diplomas" which can be found at:

<https://www.abdn.ac.uk/staffnet/teaching/assessment-policies-and-guidance-6099.php>

Where there is reason to believe that cheating in a prescribed degree assessment (including the alternative assessment to end of course written exam) has occurred this will be dealt with in accordance with the Code of Practice on Student Discipline, which can be found at:

[https://www.abdn.ac.uk/staffnet/documents/academic-quality-handbook/Code of Practice in Student Discipline \(Academic\).pdf](https://www.abdn.ac.uk/staffnet/documents/academic-quality-handbook/Code of Practice in Student Discipline (Academic).pdf)

The University's definition of Plagiarism is *the use, without adequate acknowledgement, of the intellectual work of another person in work submitted for assessment*. This definition includes the unattributed use of course materials. A student cannot be found to have committed plagiarism where it can be shown that the student has taken all reasonable care to avoid representing the work of others as his or her own.

Student details

(1) **Fill in the table below:** Replace text within the # # symbols with your data (do not delete the # # symbols).

Student ID:	[# _____ #]
Digit:	[_1 st _ _2 nd _ _3 rd _ _4 th _ _5 th _ _6 th _ _7 th _ _8 th _]

(2) **Update all fields in the document header:** Double click on the page **header**; Press "CTR + A"; Press **F9**.

Instructions to users

- Ensure you have followed the instructions in the “Student Details” section so that **your student ID** is (automatically) **reported on each page of this document**.
- Create a **new directory** and **create** there a **new C++ project (console application)**, as shown in labs/tutorials.
- The source code for each section should be reported at the end of the document.
- In this document, specific requirements for your implementation are set depending on your student ID. All details provided in each section. Make sure you identify correctly the requirements set for you. In any section of the assignment, an **implementation** that conforms to the **wrong requirements** will **not be marked**. If your implementation satisfies the set requirements (for your student ID) and additional non-contradictory requirements, the latter do not incur any penalty, but do not receive extra marks either.

Submission info

The assessment in this document is developed across sections that test increasing levels of knowledge and skills; make sure you have understood, implemented and tested one section before moving on to the next.

Your **submission** via myAberdeen should include:

- This document, with relevant sections filled as instructed (see **text in red** throughout this document). Prior to submission, **convert** this word file **into a PDF file**.
From MS Word, select File menu and then “Save as PDF”. If you do not know how to convert a word document into a PDF as explained during the course.
- All relevant **source code**, in the dedicated sections in the appendix.

Recommended completion, backup and submission strategy:

Start working on the day the assignment is released; if you can setup, compile and run an early version of your cpp project (even if it only implements one of the required functions), you can ensure your HW and SW are working properly. If a technical problem arises, you can then contact course staff readily.

As you progress with your answers, save all material (cpp/h files and PDF), on an external repository. For example: at the end of each day of work, you can zip all material (source files and PDF; no executables) finalized up to that day and email to yourself. If your PC fails on a given day, you will not lose multiple days of work.

You will submit your assignment via MyAberdeen. During the submission period, MyAberdeen accepts multiple submissions. When you are satisfied with your answer, submit. If you then improve your answers before the deadline, you can submit again; this later submission will be marked. Do not wait till the deadline to make your only submission, as unexpected technical problems may lead you to miss the submission deadline entirely.

Part 1:

Implement a class ***shape*WithCoordinate**, that contains the geometrical features of the shape (side/radius, perimeter, area) and the coordinates of its centre in a 2D reference system as in Figure 1. The actual geometrical **shape** depends on your student ID as in Table 1.

Table 1: Type of shape depending on your student ID.

shape	
8th digit of Student ID is either of these: 0, 2, 4, 6, 8	8th digit of Student ID is either of these: 1, 3, 5, 7, 9
shape = circle	*shape* = square

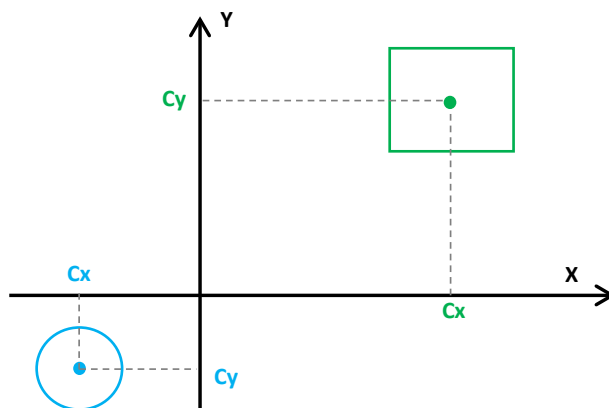


Figure 1: Pictorial example of **circleWithCoordinates** and **squareWithCoordinates**.

Details of the required implementation

Your class should support the **public member functions** in Table 2, along with any required (private/public) member functions/variables. Some entries in Table 2 are specific for certain student ID; you are expected to implement the option relative to your student ID; you are free to implement also the other option, but will receive no marks for it. These functions allow the user to **set** (and then **get**) the side dimension (square side or circle radius) and centre coordinates (Cx, Cy). After side dimension is set (initialized), a successive call to set should fail and an error message displayed; after the reset function is called, the side can be set again. The same applies to centre coordinates.

Table 2: Required public member functions for your shape (some based on your student ID). **Marks in Table 5.**

All students	
//If side not set, print error message (return 0) double getArea()	
//If side not set, print error message (return 0) double getPerimeter()	
// get radius (circle) or side (square); if not set, print error message (return 0) double getSide()	
//If coordinates are initialized, return true and set: out_Cx = X coordinate; out_Cy = Y coordinate. // Return false otherwise. bool getCoord(double & out_Cx, double & out_Cy)	
// Print shape, side and centre coordinates to screen void printShapeAndCoordInfo()	
//set radius/side (circle/square) for input>0 and side not already set; return true. False otherwise bool inputSide(double in_side)	
//set Center coordinates (Cx, Cy); return true. If already set, print error message, return false. bool inputCoord(double in_Cx, double in_Cy)	
void resetShape()	
void resetCoord()	
7th digit of Student ID is either of these: 0, 1, 2, 3, 4	7th digit of Student ID is either of these: 5, 6, 7, 8, 9
//Like inputSide: ask for user input bool inputSideFromKeyboard()	//Like inputSide: generate rand input (0,max_val] bool inputRandomSide(double max_val = 10)
//Like inputCoord: rand coords [-max_val,max_val] bool inputRandomCoord(double max_val = 25)	//Like inputCoord: ask for user input bool inputCoordFromKeyboard()

Report your selection for this section

Table 3: Fill in the table below to report your options for Part 1 and the names of the files implementing and testing part 1

Your selection	ID digit leading to that selection
Shape: _____	Based on 8 th ID digit being: _____
Input option (left/right in Table 2): _____	Based on 7 th ID digit being: _____

Main comments to your implementation

You can report here the main features of your implementation. Full source code is reported in the dedicated section at the end of this document

Required Testing

Test your implementation **with the test routine given below** to ensure it is correct. The routine should require minimal modification to be adapted to your implementation. Call from **main()**; include all necessary header files.

Table 4: Test routine for part 1.

```
#include "shape.h"
void testPart1()
{
    const int tot_test = 10;
    circle testobj_array[tot_test];
    for (int i = 0; i < tot_test; i++)
    {
        cout<< "Test item i=" << i << endl;
        cout << "Before initialization" << endl;
        testobj_array[i].printShapeAndCoordInfo();

        cout << "Init shape : " << endl;
        // uncomment the one relative to your selection
        //testobj_array[i].inputRandomSide();
        //testobj_array[i].inputSideFromKeyboard();
        testobj_array[i].printInfo();

        cout << "Init coordinates:" << endl;
        // uncomment the one relative to your selection
        //testobj_array[i].inputRandomCoord();
        //testobj_array[i].inputCoordFromKeyboard();
        testobj_array[i].printInfo();

        // insert here additional tests for all other required functions
        //

        cout << "Done" << endl<< endl;
    }
}
```

Check your test results: are these as expected? If not, which functions do not work properly?

Paste here the output screenshots of your test and report any comments.

Marking (leave blank for staff):

Note: no mark awarded for implementing functionalities not required (based on your ID).

Table 5: Marks allocation for Part 1 (out of 22 CGS marks for the entire assignment):

Part 1	CGS Marks (up to)
Properly formatted and commented header/Cpp files	
Implementation of class *shape* : tidiness and comments in the code.	_____ / 1
getArea() : implementation and test (via test routine)	_____ / 0.5
getPerimeter() : implementation and test (via test routine)	_____ / 0.5
getSide() : implementation and test (via test routine)	_____ / 0.5
getCoord(...) : implementation and test (via test routine)	_____ / 1
printShapeAndCoordInfo() : implementation and test (via test routine)	_____ / 1
inputSide() : implementation and test (via test routine)	_____ / 0.5
inputCoord() : implementation and test (via test routine)	_____ / 0.5
resetShape() : implementation and test (via test routine)	_____ / 0.5
resetCoord() : implementation and test (via test routine)	_____ / 0.5
inputSideFromKeyboard() / inputRandomSide() : implem. and test (via test routine)	_____ / 1.5
inputRandomCoord() / inputCoordFromKeyboard() : implem. and test (test routine)	_____ / 2
Additional code the test routine to test all functions	_____ / 1
Tot Part 1 = 11 CGS Marks	

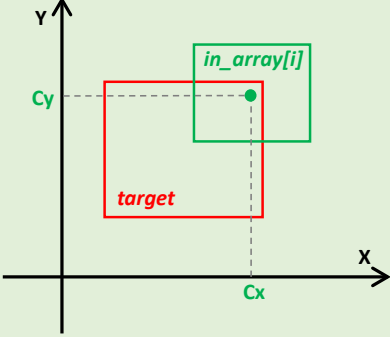
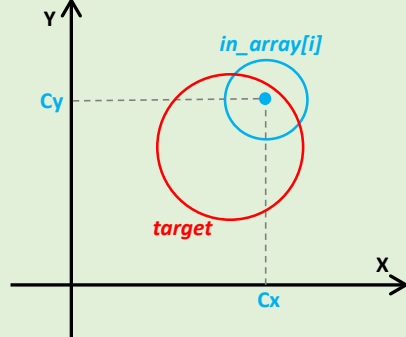
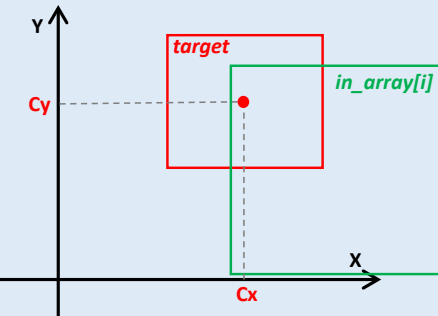
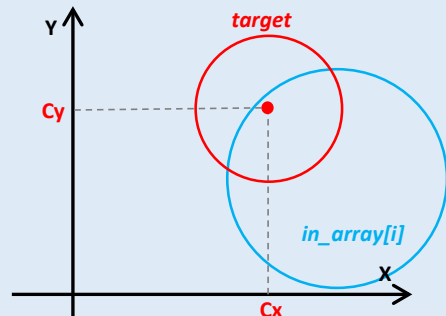
Part 2:

Instances of the class implemented in Part 1 are now used as variables to implement the functions in Table 6 (global functions, not members of your class). You can also implement any helper function you find useful. Implementation details depend on your student ID, as in Table 7. Delete from Table 7 the options that do not apply to you.

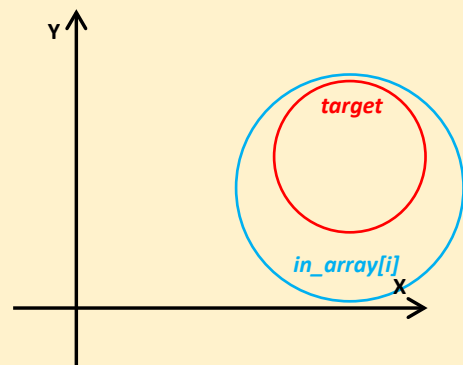
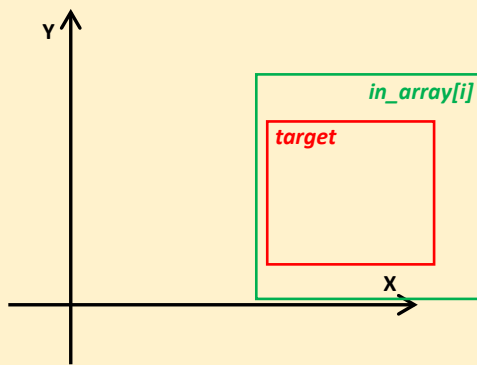
Table 6: Functions using circle/square objects as input. Marks in **Table 10**

Function	Description
If you implemented Circle in Part 1	
<code>void fillCircleArray(circle in_array[], int tot_items)</code>	Set radius and coordinates for each element in the array. You can use any input functions developed in Part 1 (including from keyboard or random).
<code>void printCircleArray(circle in_array[], int tot_items)</code>	Print shape and coordinate info for each element in the array.
<code>// input and output arrays have size tot_items; // return the number of matches found int findMatchingCircles(circle target, circle in_array[], circle out_array[], int tot_items)</code>	For the input target : find each circle in_array[i] , <i>i</i> -th element of the array, that satisfies the rule in Table 7 matching target and in_array[i] . The matching items in in_array[] are used to initialize out_array[] (which can accommodate up to <i>tot_items</i> circles). If the number of matches is N, out_array[0] has the geometrical features and coordinates as the first match; out_array[1] as the second match; so on till out_array[N-1] is set as the last match.
If you implemented Square in Part 1	
<code>void fillSquareArray(square in_array[], int tot_items)</code>	As for the circle (side instead of radius).
<code>void printSquareArray(square in_array[], int tot_items)</code>	Same behaviour as for the circle.
<code>int findMatchingSquare(square target, square in_array[], square out_array[], int tot_items)</code>	Same behaviour as for the circle. See matching rule in Table 7 .

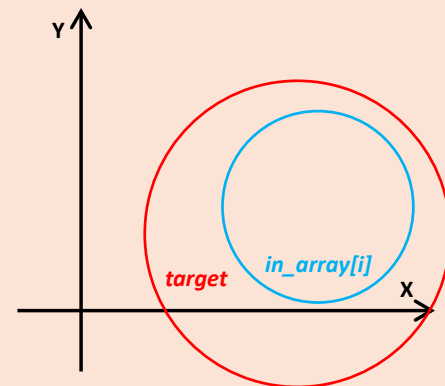
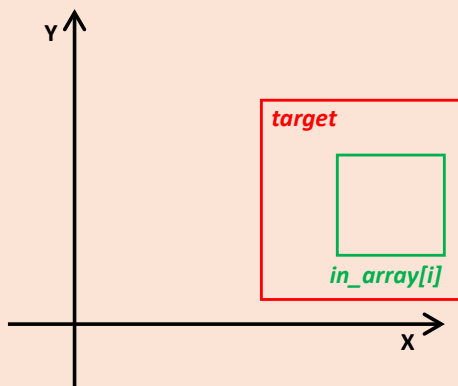
Table 7: Matching rule, based on the 6th digit of your student ID; a pictorial example shown for each rule (both square & circle).

If you implemented Square (with coordinates) in Part 1	If you implemented Circle (with coordinates) in Part 1
6th digit of student ID is 0 or 1 → Rule_0: Match on <i>in_array[i]</i> if centre of <i>in_array[i]</i> is inside <i>target</i>	
	
6th digit of student ID is 2 or 3 → Rule_1: Match on <i>in_array[i]</i> if centre of <i>target</i> is inside <i>in_array[i]</i>	
	

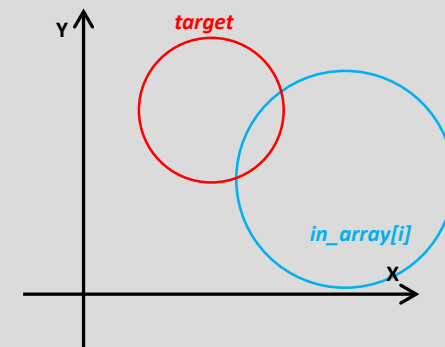
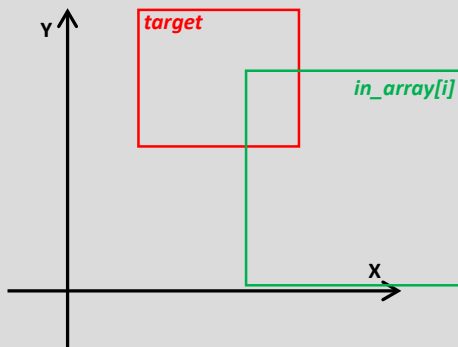
6th digit of student ID is 4 or 5 → Rule_2: Match on $in_array[i]$ if **target** is entirely inside $in_array[i]$



6th digit of student ID is 6 or 7 → Rule_3: Match on $in_array[i]$ if $in_array[i]$ is entirely inside the **target**



6th digit of student ID is 8 or 9 → Rule_4: Match on $in_array[i]$ if $in_array[i]$ overlaps with **target** (partly or entirely)



Required Testing

Test your implementation **with the test routine given below** to ensure it is correct. The routine is written for a generic **shape**: this can be **square** or **circle**; apply the required (minimal) modification/addition to adapt it to your implementation. **Call from main()**; include all necessary header files.

```
#include "cpp_Shape.h"
void testPart2 ()
{
    static const int TOT_ITEMS = 20;
    shape in_array[TOT_ITEMS], out_array[TOT_ITEMS];
    int tot_matching_items;
    shape target;
    // set the target // use different values as appropriate
```



```

target.inputCoord(0.0, 0.0);
target.inputSide(10.0);
// fill the input array
fillShapeArray(in_array, TOT_ITEMS)
// search in_array[], find all items that match the target (tot_matching_items);
// Set elements of out_array so that:
// out_array[0] is the first element in in_array[] that matches the target
// out_array[1] is the second element in in_array[] that matches the target ...
// out_array[tot_matching_items-1] is the last element in in_array[] that matches the target
tot_matching_items = findMatchingShapes(target, in_array, out_array, TOT_ITEMS);
// print the result:
cout << "Target:" << endl;
target.printShapeAndCoordInfo();
cout << endl << "All " << tot_matching_items << " matching items:" << endl;
printShapeArray(out_array, tot_matching_items);
}

```

Check your test results and comment: are these as expected? If not, which cases did not yield the intended result?

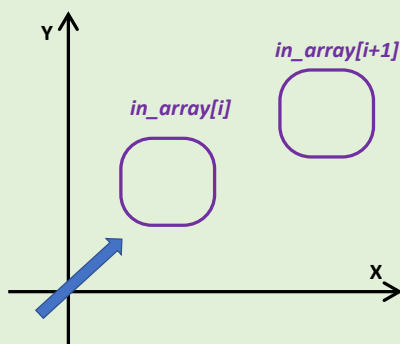
To further test your match-finding function, implement a variant of the fillShapeArray function:

fillShapeArray_PatternX(in_array, TOT_ITEMS).

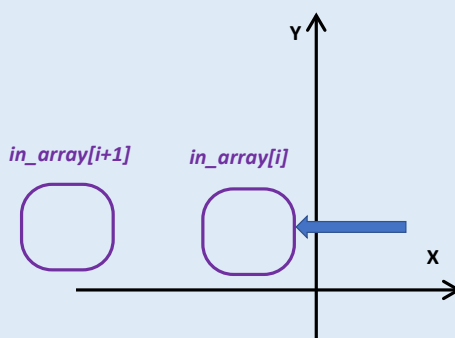
This function sets **either** centre **or** size of each item in the array following the pattern_X specified in Table 8. Using this function in your tests ensures a different selection of items in the input array compared to fillShapeArray().

Table 8: Pattern for additional tests based on the 8th digit of your student ID; a pictorial example shown for each.

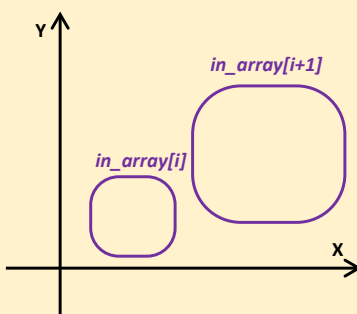
8th digit is 0 or 2 → Pattern_0: Centre of $in_array[i+1]$ is further away than $in_array[i]$ (direction of the arrow)



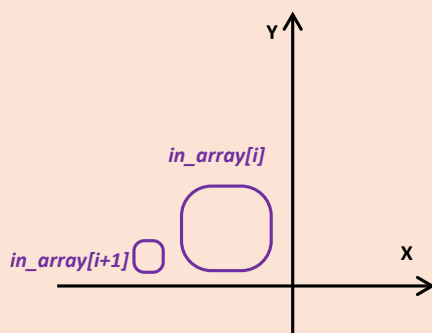
8th digit is 4 or 6 → Pattern_1: Centre of $in_array[i+1]$ is further away than $in_array[i]$ (direction of the arrow)



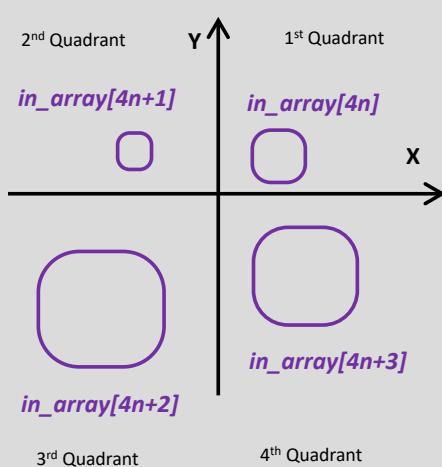
8th digit is 8 or 1 → Pattern_2: Size of $in_array[i+1]$ is larger than $in_array[i]$ (centre coordinates up to you)



8th digit is 3 or 5 → **Pattern_3**: Size of $in_array[i+1]$ is smaller than $in_array[i]$ (centre coordinates up to you)



8th digit is 7 or 9 → **Pattern_4**: Centre of $in_array[0]$ is in the first quadrant; $in_array[1]$ in the second; $in_array[2]$ in the third; $in_array[3]$ in the fourth; and so on. Size of each shape is up to you.



Check your NEW test results and comment: are these as expected? Any case not yielding the intended result?

Report your selection for this class

Table 9: Fill in the table below to **report your options** for **Part 2** and the **names of the files** implementing and testing part 2

Your selection	ID digit leading to that selection
Shape: _____	Based on 8 th ID digit being: _____
Rule (0 to 4 in Table 7): _____	Based on 6 th ID digit being: _____
Pattern (0 to 4 in Table 8): _____	Based on 8 th ID digit being: _____

Marking (leave blank for staff):

Note: no mark awarded for implementing functionalities not required (based on your ID).

Table 10: Marks allocation for **Part2** (out of **22 CGS marks** for the **entire assignment**):

Part 1	CGS Marks (up to)
fillShapeArray (...) : implementation and test (via test routine)	____ / 1
printShapeArray(...) : implementation and test (via test routine)	____ / 1
findMatchingShapes(...) : implementation and test (via test routine)	____ / 5
fillShapeArray_PatternX() : implementation and use in the test routine	____ / 3
Comment on test results:	____ / 1
Tot Part 2= 11 CGS Marks	

Total Score (for staff)

Marking: leave blank for staff

Task:	Marks (up to)
Part 1	_____ / 11
Part 2	_____ / 11
TOT	_____ / 22

Appendix source code:

Report here all source code that will be marked for a given section. This include the header files and any cpp file (including the one where the main is). **When moving to Part 2**, you **do not need to report again** a (say header) **file** if this has **not been modified** from Part 1. Simply state that the file (already reported in Part 1) is used. Make sure your code is readable when pasted in the document.

Part 1:

Content of header file __filename_____.h :

Content of header file _____.h :

Content of cpp file _____.h :

Part 2:

Content of header file _____.h :

Content of header file _____.h :

Content of cpp file _____.h :