CLASSES AND PRIVACY

Workshop 3 (0.8)

In this workshop, you are to code a class with private data members and public and private member functions. The class will encapsulate a "Mark" for an assessment.

LEARNING OUTCOMES

Upon successful completion of this workshop, you will have demonstrated the abilities to

- define a class type;
- privatize data within the class type;
- instantiate an object of class type;
- access data within an object of class type through public member functions;
- use standard library facilities to format data inserted into the output stream (DIY);

SUBMISSION POLICY

The workshop is divided into 3 sections;
in-lab - 30% of the total mark
To be completed before the end of the lab period and submitted from the lab.
at-home - 35% of the total mark
To be completed within 2 days after the day of your lab.
DIY (Do It yourself) – 35% of the total mark
To be completed within 3 days after the at-home due date.
The *in-lab* section is to be completed after the workshop is published, and before the end of the lab session. The *in-lab* is to be submitted during the workshop period form the lab.
If you attend the lab period and cannot complete the *in-lab* portion of the workshop during that period, ask your instructor for permission to complete the

in-lab portion after the period. You must be present at the lab in order to get credit for the *in-lab* portion.
If you do not attend the workshop, you can submit the *in-lab* section along with your *at-home* section (see penalties below). The *at-home* portion of the lab is due on the day that is 2 days after your scheduled *in-lab* workshop (23:59) (even if that day is a holiday).
The DIY (Do It Yourself) section of the workshop is a task that utilizes the concepts you have done in the in-lab + at-home section. This section is completely open ended with no detailed instructions other than the required outcome. You must complete the DIY section up to 3 days after the at-home section.
All your work (all the files you create or modify) must contain your name, Seneca email and student number.
You are responsible to back up your work regularly.
Ask your professor if there are any additional requirements for your specific section.

Citation and Sources

When submitting the DIY part of the workshop, Project and assignment deliverables, a file called sources.txt must be present. This file will be submitted with your work automatically.
You are to write either of the following statements in the file "sources.txt":
I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.
Then add your name and your student number as signature
OR:
Write exactly which part of the code of the workshops or the assignment are given to you as help and who gave it to you or which source you received it from.
You need to mention the workshop name or assignment name and also the file name and the parts in which you received the code for help.
Finally add your name and student number as signature.
By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrong doing.

Late Submission Penalties:

-*In-lab* portion submitted late, with *at-home* portion:
0 for *in-lab*. Maximum of **DIY+at-home**/10 for the workshop.

-at-home or DIY submitted late:

1 to 2 days, -20%, 3 to 7 days -50% after that submission rejected.

-If any of *the at-home* or in-lab portions is missing, the mark for the whole workshop will be **0/10**

-If DIY portion is missing you will lose the mark for the DIY portion of the workshop.

WORKSHOP DUE DATES

You can see the exact due dates of all assignments by adding -due after the submission command:
Run the following script from your account (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

```
~profname.proflastname/submit 244/NXX/WS02/in_lab -due<ENTER>
~profname.proflastname/submit 244/NXX/WS02/at_home -due<ENTER>
~profname.proflastname/submit 244/NXX/WS02/DIY -due<ENTER>
```

COMPILING AND TESTING YOUR PROGRAM

All your code should be compiled using this command on matrix:

```
g++ -Wall -std=c++11 -o ws (followed by your .cpp files)
```

After compiling and testing your code, run your program as follows to check for possible memory leaks: (assuming your executable name is "ws")

```
valgrind ws <ENTER>
```

IN-LAB (30%)

MARK MODULE

The Mark class can hold a mark value and an "out of" value and then display the mark in following formats: As is, Raw and Percentage

For example if the mark is 12.5 out of 20, it can be shown as:
As is: *12.5/20*
Raw : *0.625* (12.5 divided by 20)
Percentage: *%63* (%62.5 rounded up)

To accomplish the above follow these guidelines:

Design and code a module named `Mark`.

Follow the usual rules for creating a module: (i.e. Compilation safeguards for the header file, namespaces and coding styles your professor asked you to follow).

In your `Mark` header file, predefine the following constants as integers (in sdds namespace).

`DSP_RAW` with a value of 1.
`DSP_PERCENT` with a value of 2.
`DSP_ASIS` with a value of 3.
`DSP_UNDEFINED` with a value of -1.

These values are going to be used to display a Mark in different formats (i.e Raw, As is and Percentage)

Create a class called Mark.

PRIVATE MEMBERS:

This class holds a mark for an assessment using three member variables:

- `m_displayMode`
  This is an integer that holds how a mark is to be displayed. The value of this member variable can be one of the four constant values listed above.
- `m_mark`
  This is a double value that holds the actual mark given for an assessment.
- `m_outOf`
  This is an integer value that is the maximum value for a mark.

Create one private member function that is used for fool-proof data entry.

```
void flushKeyboard()const:
```

Read one character at the time using cin.get() until you read the newline character ('\n')

PUBLIC MEMBERS:

`void set(int diaplayMode);`

This function sets `m_displayMode` member variable to value of the incoming argument (`displayMode`)

`void set(double mark, int outOf);`

This function sets "`m_mark`" and "`m_outOf`" member variables to the corresponding arguments. When this function is called, if the `outOf` argument is not provided, it will default its value to 1.

`void setEmpty();`

Sets `m_displayMode` to `DSP_UNDEFINED`, `m_mark` to -1 and `m_outf` to 100.

`bool isEmpty()const;`

returns true if the `Mark` object is Empty.

`int percent()const;`

divides `m_mark` by `m_outOf` them multiplies it by 100 and ads 0.5 to it. Then it will cast the outcome to an integer and returns it.

`double rawValue()const;`

returns the result of `m_mark` divided by `m_outOf`.

`bool read(const char* prompt);`

This function reads the mark in following format:
mark/out of, for example: 30/50 or 20/20 and returns true if successful and If it cannot read the any part of the mark it will fail and return false.

To do this **read** first prints the `prompt` argument. Then it will read a double into `m_mark`, ignore one character and then read an integer into `m_outOf`. Afterwards it will check if the "`cin`" object has failed or not. If "`cin`" has failed, it will clear `cin`, set the `Mark` object to empty and return false.

In any case before returning anything read will flush the keyboard.

`ostream& display() const;`

If the object is empty, it will print:
`"Empty Mark object!"`
if it is not empty, depending on the value of `m_displayMode` it will print the following:
if `m_displayMode` is `DSP_RAW` it will print the rawValue().
if `m_displayMode` is `DSP_PERCENT` it will print `"%"` and then the percent() value;

if `m_displayMode` is `DSP_ASIS` it will print `m_mark` , `"/"` , and then `m_outOf`.
If `m_displayMode` is `DSP_UNDEFINED` it will print `"Display mode not set!"`
If `m_displayMode` is none of the above it will print `"Invalid Mark Display setting!"`
At the end read function will return the `cout` object.

IN-LAB MAIN MODULE

```
/*************************************************************************
// OOP244 Workshop 3: Member functions and privacy
// File markTester.cpp
// Version 1.0
// Date     2019/09/19
// Author   Fardad Soleimanloo
// Description
// tests Mark data entry
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
// Fardad
/////////////////////////////////////////////////////////////////
*************************************************************************/
#include <iostream>
#include "Mark.h"
using namespace std;
using namespace sdds;
int main() {
   Mark m1, m2;
   m1.setEmpty();
   cout << "m1: Empty mark: [";
   m1.display() << "]" << endl;
   m1.set(12.5, 20);
   cout << "m1: Display not set: [";
   m1.display() << "]" << endl;
   cout << "m1: Display set to percentage:" << endl;
   m1.set(DSP_PERCENT);
   cout << "m1: 12.5 out ouf 20 is ";
   m1.display() << endl;
   while (!m1.read("Enter Mark for m1 (mark/out_of): ") ){
      cout << "Invalid Mark, Retry: ";
   }
   cout << "m1: The mark you entered is: ";
```

```
    m1.set(DSP_ASIS);
    m1.display() << endl;
    cout << "m1: With the raw value of: ";
    m1.set(DSP_RAW);
    m1.display() << endl;
    cout << "m1: And percentage value of: ";
    m1.set(DSP_PERCENT);
    m1.display() << endl;
    m2.setEmpty();
    cout << "Setting m2 to the raw value of m1..." << endl;
    m2.set(m1.rawValue());
    cout << "done!" << endl;
    cout << "m2: The mark is: ";
    m2.set(DSP_ASIS);
    m2.display() << endl;
    cout << "m2: With the raw value of: ";
    m2.set(DSP_RAW);
    m2.display() << endl;
    cout << "m2: And percentage value of: ";
    m2.set(DSP_PERCENT);
    m2.display() << endl;
    return 0;
}
```

EXECUTION EXAMPLE RED VALUES ARE USER ENTRY

```
m1: Empty mark: [Empty Mark object!]
m1: Display not set: [Display mode not set!]
m1: Display set to percentage:
m1: 12.5 out ouf 20 is %63
Enter Mark for m1 (mark/out_of): abc
Invalid Mark
Enter Mark for m1 (mark/out_of): 12.5/abc
Invalid Mark
Enter Mark for m1 (mark/out_of): 12.5/20
m1: The mark you entered is: 12.5/20
m1: With the raw value of: 0.625
m1: And percentage value of: %63
Setting m2 to the raw value of m1...
done!
m2: The mark is: 0.625/1
m2: With the raw value of: 0.625
m2: And percentage value of: %63
```

IN-LAB SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload the Mark module and the markTester.cpp program to your matrix account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account during the lab(use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS03/in_lab<ENTER>

and follow the instructions generated by the command and your program.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

AT_HOME (35%)

MARK MODULE (COMPLETED)

Complete the Mark Module by adding the following:

In your Mark header file, add the following predefine constants as integers (in sdds namespace).

DSP_GPA with a value of 4.
DSP_LETTER with a value of 5.

ADDITIONAL PRIVATE MEMBER VARIABLE AND FUCTION:

- m_name
  A private array of 51 characters to keep a C style string for the name or the title of the mark.

```
void prnLetter()const;
```

A private function that prints the letter grade equivalent of the mark. If the mark is not empty, this function will check the percent() value of the mark and:

print	value between
F	0 and 49
D	50 and 54
D+	55 and 59
C	60 and 64
C+	65 and 69
B	70 and 74
B+	75 and 79
A	80 and 89
A+	90 and 100
?	values more than 100

Note that this function will not print new line after the letter grade and will not print anything if Mark is empty

ADDITIONAL PUBLIC MEMBER FUNCTIONS:

```
void set(const char* name);
```

Sets the m_name member char array to the content pointed by name to the maximum of 50 characters

```
void set(const char* name, double rawMark, int outof = 1);
```

First calls the set(const char*) function to set the name, then calls the set(double, int) function to set the m_mark and m_outOf member variables.

```
void setEmpty();
```
(modify the in_lab implementation)

Add a line of code to set m_name to an empty string.

```
bool readName(const char* prompt , int maxLen = 50);
```

This function should first print the prompt they try to read up to ”maxLen” characters or 50 (whichever comes first) from the console and place it in m_name. *(i.e. if the value passed for maxLen is greater that 50, then it should be reset to 50)*

If the number of characters entered are more than the limit above (i.e. cin fails) then cin should be cleared, keyboard flushed, and the m_name should be set to an empty string. The function in this case will return false, otherwise it will return true.

```
void changeOutOf(int value);
```

This function should change the value of m_outOf member variable to the incoming argument value and then adjust the value of m_mark member variable relatively to keep the total mark value the same.

If the value is less than 1, it should set the mark to a safe empty state.

To accomplish this before changing the m_outOf member variable, multiply the value of m_mark to the division of the argument value by m_outOf. Make sure you cast the argument value to double before division to have an accurate calculation.

```
double GPA()const;
```

Returns the exact GPA value of the mark; which is rawValue() multipled by 4.

```
double letterBasedGPA()const;
```

Returns the GPA value of the mark based of its letter grade: *(Not the GPA() function)*

Returns	for percent() between	
0.0	0 and 49	(F)
1.0	50 and 54	(D)
1.5	55 and 59	(D+)
2.0	60 and 64	(C)
2.5	65 and 69	(C+)
3.0	70 and 74	(B)
3.5	75 and 79	(B+)
4.0	80 and 89	(A)
4.0	90 and 100	(A+)

```
ostream& display(int width = 55)const;
```

(this is not a new function, modify the in_lab implementation and add the following features)

Before printing the mark, if the m_name is not an empty string, print the m_name left justified in ”width” spaces filling the empty spaces with dots (`'.'`) .

If m_name is an empty string, don't print the anything for m_name and continue with printing the mark.

Add the following formatting adjustments:

For DSP_RAW; Print the number with two digits after the decimal point

For DSP_ASIS ; Print the m_mark value with one digit after the decimal point.

Add the following printing modes:

DSP_LETTER.

If the printing mode is DSP_LETTER, make a call to prnLetter() to print the letter grade.

DSP_GPA

If the printing mode is DSP_GPA, Print the return value of the GPA function with two digits after the decimal point.

AT HOME MAIN MODULE

```
/*****************************************************************
// OOP244 Workshop 3: Member functions and privacy AT HOME
// File markAtHomeTester.cpp
// Version 1.0
// Date      2019/09/23
// Author    Fardad Soleimanloo
// Description
// tests Mark data entry
//
// Revision History
// -----------------------------------------------------------
// Name            Date            Reason
// Fardad
/////////////////////////////////////////////////////////////////
*****************************************************************/
#include <iostream>
#include "Mark.h"
using namespace std;
using namespace sdds;
ostream& line(int len, char ch) {
   for (int i = 0; i < len; i++, cout << ch);
   return cout;
}
ostream& number(int num) {
   cout << num;
```

```
   for (int i = 0; i < 9; i++) {
      cout << " - " << num;
   }
   return cout;
}
void listMarks(const Mark* marks, int num) {
   Mark result;
   result.setEmpty();
   double gpa = 0;
   int aver = 0;
   int i;
   for (i = 0; i < num; i++) {
      marks[i].display(60) << endl;;
      aver += marks[i].percent();
      gpa += marks[i].letterBasedGPA();
   }
   line(64, '-') << endl;
   result.set(gpa / num, 4);
   result.set(DSP_GPA);
   result.set("GPA");
   line(30, ' ') ;
   result.display(30) << endl;
   result.set(aver / num, 100);
   result.set(DSP_PERCENT);
   result.set("Average");
   line(30, ' ');
   result.display(30) << endl;
}
int main() {
   Mark m[3];
   Mark test;
   int i;
   cout << "Enter 3 marks: " << endl;
   for (i = 0; i < 3; i++) {
      m[i].setEmpty();
      m[i].set(DSP_LETTER);
      number(i + 1) << ":" <<endl;
      while (!m[i].readName("Subject Name: ", 6) || !m[i].read("Mark: ")) {
         cout << "Invalid Entry, retry!" << endl;
         m[i].set(DSP_LETTER);
      }
   }
   listMarks(m, 3);
   test.setEmpty();
   for (i = -1; i <= 21; i++) {
      cout << i << ": -------------------" << endl;
      for (int j = 1; j <= 5; j++) {
         test.set(j);
         test.set(double(i), 20);
         test.display() << endl;
      }
      if (i < 0) i += 8;
   }
   return 0;
}
```

EXECUTION EXAMPLE RED VALUES ARE USER ENTRY

```
Enter 3 marks:
1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1:
Subject Name: WEB2222
Invalid Entry, retry!
Subject Name: WEB222
Mark: abc
Invalid Entry, retry!
Subject Name: WEB222
Mark: 12/13
2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2 - 2:
Subject Name: OOP244
Mark: 15/20
3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3 - 3:
Subject Name: DBS211
Mark: 18/20
WEB222.....................................................A+
OOP244.....................................................B+
DBS211.....................................................A+
-----------------------------------------------------------------
                            GPA...........................3.8
                            Average.......................%85
-1: -------------------
Empty Mark object!
Empty Mark object!
Empty Mark object!
Empty Mark object!
Empty Mark object!
8: -------------------
0.40
%40
8.0/20
1.6
F
9: -------------------
0.45
%45
9.0/20
1.8
F
10: -------------------
0.50
%50
10.0/20
2.0
D
11: -------------------
0.55
%55
11.0/20
2.2
D+
12: -------------------
0.60
%60
12.0/20
```

2.4
C
13: ------------------
0.65
%65
13.0/20
2.6
C+
14: ------------------
0.70
%70
14.0/20
2.8
B
15: ------------------
0.75
%75
15.0/20
3.0
B+
16: ------------------
0.80
%80
16.0/20
3.2
A
17: ------------------
0.85
%85
17.0/20
3.4
A
18: ------------------
0.90
%90
18.0/20
3.6
A+
19: ------------------
0.95
%95
19.0/20
3.8
A+
20: ------------------
1.00
%100
20.0/20
4.0
A+
21: ------------------
1.05
%105
21.0/20
4.2
?

AT-HOME SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload `Mark module` and the `markAtHomeTester.cpp` program to your `matrix` account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account (use your professor's Seneca userid to replace `profname.proflastname`, and your section ID to replace `NXX`, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS03/at_home<ENTER>

and follow the instructions generated by the command and your program.

> **IMPORTANT**: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

DIY (35%)

Use the Marks module to develop the DIY part.

MARKSREPORT MODULE

The MarksReprot holds an array of Marks with an unknown length (decided at run time) and a Title that represents what this report is on. To Make the Name of the report printable, make sure it is not more than 70 Characters.

The MarkReport module should work as follows.
1- It should get initialized to have the essentials of the module ready for data entry.

2- Ask the user for the name of the report and the number of marks to be entered. Then it should start a fool-proof mark entry and receive all the marks.

3- Then it should create and print the report. (see execution sample)

4- Finally it should be terminated to free all the dynamically allocate resources.

The Main for this module is as follows:

```
#include "MarksReport.h"
int main() {
   sdds::MarksReport mr;
   mr.initialize();
   mr.getMarks();
   mr.print();
   mr.terminate();
   return 0;
}
```

For your convenience an extra optional module is added to this project named "ReportUtils"

Use of this module is completely optional since all the tasks can be developed in the MarksReport Module. However, if you have any additional code that you want to use (for example codes that you may have from the previous workshops and you would like to reuse) you can place your code in this module.

This Module will be compiled along with your MarksReport Module. If you wish not to use it, just leave it as is (an empty module) and its compilation will not affect your code.

Your execution for this module should match the following:

EXECUTION EXAMPLE RED VALUES ARE USER ENTRY

```
Enter the report Name: Jane Doe, Semester 2
Enter the number of marks: 3
Please enter 3 marks:
1: Subject Name: DBS211
Mark (mark/outof): 75/100
2: Subject Name: DCF255
Mark (mark/outof): 20/25
3: Subject Name: OOP244
Mark (mark/outof): 70/85
Entry Completed!

------------------------------------------------------------------------
```

```
Jane Doe, Semester 2
------------------------------------------------------------------------
Marks entered:
DBS211.....................................................B+
DCF255.....................................................A
OOP244.....................................................A
------------------------------------------------------------------------
                              Lowest Mark.....................%75
                              Highest mark....................%82
                              GPA.............................3.8
```

DIY SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload Mark, MarksReport and ReportsUtils modules and the report.cpp program to your matrix account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account (use your professor's Seneca userid to replace profname.proflastname, and your section ID to replace NXX, i.e., NAA, NBB, etc.):

~profname.proflastname/submit 244/NXX/WS03/DIY<ENTER>

and follow the instructions generated by the command and your program.

IMPORTANT: Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.