

Queens College, CUNY, Department of Computer Science
Object-Oriented Programming in C++
CSCI 211/611
Summer 2018
Instructor: Dr. Sateesh Mane

© Sateesh R. Mane 2018

due date Friday, July 20, 2018, 11.59 pm

Homework: Classes: functions and methods

- Experience with other classes has demonstrated that in many cases the source of difficulty is not the mathematics or the programming.
- The source of difficulty is the English (understanding the text).
- If you do not understand the words in the lectures or homework, **THEN ASK.**
- If you do not understand the concepts in the lectures or homework, **THEN ASK.**
- Send me an email, explain what you do not understand.
- Do not just keep quiet and then produce nonsense in exams.
- **Consult your lab instructor for assistance.**
- You may also contact me directly, but I cannot promise a prompt response.
- Please submit your inquiry via email, as a file attachment, to `Sateesh.Mane@qc.cuny.edu`.
- Please submit one zip archive with all your files in it.
 1. The zip archive should have either of the names (CS211 or CS611):
`StudentId_first_last_CS211_hw_classes1.zip`
`StudentId_first_last_CS611_hw_classes1.zip`
 2. The archive should contain one “text file” named “hw_classes1.[txt/docx/pdf]” and one cpp file per question named “Q1.cpp” and “Q2.cpp” etc.
 3. Note that not all questions may require a cpp file.

General information

- You should include the following header files, to run the programs below.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <cmath>
```

- If you require additional header files to do your work, feel free to include them.
- **Include the list of all header files you use, in your solution for each question.**
- The questions below do not require complicated mathematical calculations.
- If for any reason you require help with mathematical calculations, **ask the lab instructor or the lecturer.**

Q1 Classes: functions and methods

- **Write a class Student with private data and public methods.**
- The class has two private data members of type `string` and `vector<double>`.

```
class Student {  
private:  
    string name;  
    vector<double> grades;  
  
public:  
    // to do  
};
```

- We shall write additional class methods, to be described below.
- Some of the methods will be tagged `const`.

Q2 Accessors and mutators

- **Write public accessor and mutator methods.**

```
string getName() {...} //should be "const"
void setName(const string &s) {...}
string& nameRef() {...}
```

- The method `nameRef()` is safe, because the reference is to the data member `name`, and `name` does not go out of scope at the function exit.
- Because the return value is a reference to `name`, **the method `nameRef()` is both an accessor and a mutator.**

```
st.nameRef() = "something"; // method is mutator
cout << st.nameRef() << endl; // method is accessor
```

- Because `nameRef()` can be employed as a mutator, it is not `const`.
- **Write a mutator to add a grade to the vector `grades`.**

```
void addGrade(double x) {...}
```

1. If $x \geq 0$ and $x \leq 100$, populate the vector `grades` with the value x .
2. Else return and do nothing.

Q3 `const` methods

- **Write a method to calculate and return the average grade.**
- If the size of `grades` is zero, then return 0.

```
double getAvg() // etc
```

- **Write a method to return the highest grade.**
- If the size of `grades` is zero, then **return -1**.

```
double highestGrade() // etc
```

- **Write a method to `print()` the name and grades.**

```
void print() // etc
```

1. First print `name`.
2. Next print the grades in a loop, one grade on each line.
3. Print a message “no grades posted yet” if the size of `grades` is zero.

- **Explain why all of the methods in this section are `const`.**

Q4 non-const methods

- Write a method to return a pointer to the address of an element of grades.

```
double* gradePtr(int n) {...}
```

- Return the address of `grades[n]` if the value of n is valid.
- Else return `NULL`.
- Explain why this method is not const.

Q5 Class declaration

- Your overall class declaration should look like the following.

```
class Student {
private:
    string name;
    vector<double> grades;

public:
    string getName();                // apply keyword "const" correctly
    void setName(const string &s);
    string& nameRef() { return name; }

    void addGrade(double x);
        if ((x >= 0) && (x <= 100))
            grades.push_back(x);
    }

    // non-const methods
    double * gradePtr(int n);

    // const methods
    double getAvg();                // apply keyword "const" correctly
    double highestGrade();          // apply keyword "const" correctly
    void print();                   // apply keyword "const" correctly
};
```

Q6 Functions

- **Write two functions as follows to use your code.**

```
void highlow_avg_grade(const Student *a, const Student *b, int n);  
void highlow_top_grade(const Student *a, const Student *b, int n);
```

- In both functions, a is a pointer to a single object and b is a pointer to an array of length n .
- First function:

1. Find the name and average grade of the student with the highest average grade.
2. Find the name and average grade of the student with the lowest average grade.
3. **Print output to screen.**

```
cout << "high avg = " << name_high << " " << high << endl;  
cout << "low avg  = " << name_low << " " << low << endl;
```

- Second function:

1. Same as the first function but replace `getAvg()` by `highestGrade()`.
2. **Print output to screen.**

```
cout << "high top grade = " << name_high << " " << high << endl;  
cout << "low top grade  = " << name_low << " " << low << endl;
```


Q7 Example main program

- Your code should work correctly when tested with the following main program.

```
// include headers, class declaration, functions

int main()
{
    Student *Alice = new Student;
    Student *BobTwins = new Student[2];

    // use nameRef() to set name of Alice to "Alice";
    // use setName(...) to set names of BobTwins to "Bob A" and "Bob B"

    // call print() for Alice and BobTwins

    for (int i = 65; i <= 110; i += 10) {
        // addGrade(i+0.1)    add grades for Alice
    }

    for (int i = 57; i <= 110; i += 10) {
        // addGrade(i+0.2)    add grades for BobTwins[0]
    }

    int igrade=0;
    while (true) {
        // double *d = ... gradePtr(igrade)    pointer to double for BobTwins[0]
        // if d == NULL then break out of loop
        // addGrade(*d - 0.5)    add grades for BobTwins[1]

        ++igrade;                // increment counter
    }

    // call print() for Alice and BobTwins

    highlow_avg_grade(..., ..., 2);        // call functions
    highlow_top_grade(..., ..., 2);

    // release memory as appropriate

    return 0;
}
```