1 Syllabus—Important bits

1. Grading:

(a) All homework code throughout the course will be compiled with Visual Studio 2017. You must make sure your code works on Visual Studio 2017 before submitting. If your code works on XCode but not VS2017, then your code is relying on quirky implementation-defined or undefined behaviors and can be improved.

(b) 10% of your grade is the max of your midterm and final exam scores. This policy is meant to help you: if you have a bad day on the day of the midterm or if you must miss the midterm for any reason, your final exam score can overwrite your midterm score.

(c) Your lowest 2 homework scores will be dropped. It is recommended you use this policy as one would use an insurance policy.

(d) View example overall grade distributions from past PIC 10B courses. Even though exam scores average around 50%, overall grades for the course tend to be quite good.

(e) View the RemarksOnGrading.pdf document for more information about course overall grades.

(f) Class Participation: Full marks are rewarded for earning 72% of all points. Simply responding nets 80% of available. Points. Thus, one can answer every question wrong throughout the quarter and still get full marks on class participation.

(g) Homework: view the HW_Codes.pdf document.

2. Formal Polices:

(a) With a valid excuse for inability to take the final exam, an oral final exam will be administered if the student has more than 70% of course points.

(b) Students may collaborate, but this must be clearly marked on homework. Do not copy code!

(c) Students with disabilities need to notify the OSD office sooner rather than later.

(d) Exam regrade requests must be submitted during the discussion students received their exams. Requests must be outlined on a separate sheet of paper; the exam itself cannot be modified.

(e) Your entire exam will be regraded; your score may go up but may also go down.

(f) Homework regrades must be made via a handwritten request within 5 business days of homework grade release date. Scores may go up or may go down.

(g) Questions should be asked on the discussion forums. Questions specific to a student can be asked during office hours.
2 Doxygen Commenting

Doxygen is a tool for automatically creating documentation from comments in C++ source files. Commenting in the Doxygen style, thus, not only increasing readability of one’s code, but also allows one to easily create documentation for one’s code.

Generally, comments should help individuals understand how your code works, and it should help individuals understand how to use your code. To the first point, comments should explain semantically what different lines of code are doing. To the second point, comments should explain the purpose of blocks of code in your program, and what information a user must supply in order to use the code.

Do not overload your code with comments, however. Variable names can be clear enough that commenting can be kept minimal; certain lines of code can be simple enough that commenting is unnecessary.
Comments before classes should outline the purpose of the class.

3 Review

3.1 size_t

size_t is an implementation defined type designed to be used for container indexing and sizing. Depending on the architecture of the system, size_t will be an unsigned integer type of enough bytes to store the size, and consequently the indices, of the largest possible container on the system. It is important to use size_t so you do not have to worry about integer overflow errors. Further, functions like std::vector<T>::size() return data of type size_t, so it is important to use variables of type size_t to store the return of these functions to avoid downcasting, which can result in errors.

However, one must be weary in treating unsigned data like normal integer data. For example,

```c++
#include <iostream>

int main() {
    unsigned long sz = 1000;
    int y = -25;
    int x = sz/y;
    std::cout << x << std::endl;
}
```

will print out 0.

3.2 Const correctness

Const correctness with regard to data means that one should mark data that does not change as const. Const correctness with regard to classes means that functions that do not change the state of the class should be marked as const.

```c++
/** 
 * A useless Example to explain const correctness 
 */

class Example {
    public:

    /** 
     * Constructor to set the const data of the class 
     * @param the_number is the magic member data 
     */
    Example(int the_number) : the_number(the_number) {}

    /** 
     * function to modify the parameter to some nonsense 
     * @param x a reference to the data to modify 
     */
    void theFunction(int& x) const {
        x = the_data*the_number;
    }
}
```
In this example, we mark Example::theFunction(int& x) as const but not Example::theSetter(int x). While theFunction does modify data via the reference x, it does not modify the state of the class, so it should be made const. theSetter on the other hand modifies member data, so cannot be made const.

Importantly, const objects only have access to const functions, and non-const objects have access to all functions, but will use non-const versions if there are any available. This last point is called “overloading on const,” and will be discussed later.