Rubric General Overview (specific percentage allocation may differ slightly or each assignment**)**:

Correctness of your program: Compressed file extraction errors and compilation errors lose 50%. Your program will earn significant (10% each, up to 80%) points losses for any/all of the following: • Run-time errors and/or incorrect answers • Not following assignment instructions, including output/format instructions • Not following algorithm specifications / requirements provided for the assignment	50 to 80%
Documentation: including assumptions, explanations, illustrations, results, and output consideration (appropriate type, quantity and quality of comments).	10% each, up to 30%
Design, modularity, readability, style/maintainability & organization of program	10% each, up to 30%

1. Correctness
   - Compilation errors and/or warnings (see g++ -Wall -g flags) (-5/warning up to 50%; -50% for compile error)
   - Runtime errors each (-10 to 50 % each) the cost will be lower if you turn off broken parts.
   - Memory leaks (-10 for first assignment 20% thereafter)

2. Meets required specifications, compliance with constraints, material submitted, implementation and performance
   - You submitted all required material, as specified, by the deadline: (-5 to 10% each)
   - Your code meets all explicit and implicit requirements in instructions: (-5 to 10% each)
   - Performance your algorithm/code does not waste time/space resources (-5 to 10% each)
   - You must turn off all debugging code (unless final command line argument == "DEBUG") (-10%)
   - Your program must not produce un-needed or excessive output when run without arguments (- 10%)

3. Program Organization/style/comments
   - Proper Doxygen comments for each file, class and module: (-10%)
   - In line, you must write non-redundant comments to help informed readers understand why you wrote your code the way you did, whenever it might be unclear or confusing. Ideally, your code will be so clear it needs very few such comments. (-5 to 10% each unexplained, confusing code line or false comment)
   - Meets all Coding style guidelines below (variable/module names, modular, cohesive, loosely coupled, non-redundant readable code with proper indentations, blank lines, etc. -5 to 10% each)

Use the checklist below to ensure the best score

Mark each lettered requirement after you verify that all your code meets that requirement

Quality/Style checklist: any work you submit must meet each of these guidelines

A. Unless specified otherwise, at the top of your console output, display author(s), title & summary.

B. Do **not** submit code that uses **break (except in case statements) or continue:** make code easier to understand. Do not use STL, C++11 library, prohibited features or features not yet covered when program is assigned.

C. You must avoid excessive **chaining**. The main method must clearly show the overall program structure. Beyond main, you must have **three or more non-trivial methods** (like IPO) in your program – see next guideline.

D. Decompose problems into sub-methods that you can name easily with verbs or noun-verb phrases.

E. Code must be **easy to read**. Use a standard indentation model, { }. Limit lines to 60-80 characters.

F. **Factor or eliminate redundant code**: put any set of three or more lines duplicated in multiple places into a method. Review your code to find duplication that you could eliminate if you re-factored, re-structured or reused methods. You must define other methods as needed for structure and/or to eliminate redundancy.

G. Limit variable **scope**. Do **NOT use global variables** (except debug), unless you get prior instructor approval.

H. Code that does actual input, processing/calculation, or actual output must be in separate methods.

I. You usually do **not** do actual input or output directly in main. Main must call methods to do those tasks. Use separate method(s) to do console output and separate method(s) to do any other output.

J. The code that asks the user for input must not be in the same method as code to read data from a file.

K. You must include **Doxygen** style **Class/file comments** with proper description/**purpose**, **author**, file, etc.

L. You must have a blank line & complete **Doxygen** style **method comments** for each method header/prototype.

M. You must not use "**magic numbers**" or literals (e.g. 3.141 or "enter") in methods. Instead, use CONSTANTS that clarify your code. Using -1, 0, 1 and 2 are ok as loop increments, but not if they have other meaning.

N. You must use **Good names/identifiers** and designs to reduce the need for explanatory comments. You must use easy to understand, meaningful identifiers for method and variable names.

O. Structure your solution using static methods that accept parameters and return values where needed.

P. When a **printed listing** is required, it must be **easy to read and include page and line numbers**.

Q. Submit **typescripts** of your program test & application g++ compilation & full execution with valgrind.

R. You must remove **redundant comments** that say what the code says, for example: x++; //increment x.
If code might be confusing, first try to simplify. If you cannot simplify it, your comments must say why you did it this way, for example: *// I ran out of time and this works; so I accept responsibility and point loss.*

S. All file names must meet instruction requirements. Do not use spaces in filenames for use on Linux.

T. Follow conventional case standards on ClassNames, methodNames, variableNames, CONSTANT_NAMES.

U. Solo programs are for you to demonstrate what you have mastered. Unless authorized, you may not use code, request aid from anyone else, or communicate with anyone else about solo programs except the instructor.

V. **Ask all questions about projects in class so that everyone benefits from the answers**. I am often out of town and rarely get or reply to email homework questions between our class meetings / discussions and the due date

W. You may use any mix of standard design tools: pseudo-code, hierarchy diagrams, flow charts, DFDs, etc. You **must use them correctly**. You may change your design after submitting required drafts. (For pseudo-code guidelines, see http://users.csc.calpoly.edu/~jdalbey/SWE/pdl_std.html .)

X. Deductions for each internal style / quality error can increase/double each assignment (2^5 = 32).

Y. **You are responsible for ensuring that I receive your work for grading by the deadline**.

Z. Do not submit executable code. Server may reject attachments that contains executable code (.exe) even zipped.