# Final-Practice Questions

Q1. Given the following code snippet, complete the tasks listed below:

1. #include <iostream>
2. #include <string>
3. #include <algorithm>
4. #include <numeric>
5. #include <vector>
6. using namespace std;
8. struct ComputerParts {
9. string partType;
10. string model;
11. double price;
12. double powerConsumption;
13. bool used;
14. };
16. int main() {
18. vector<ComputerParts> wishList;
20. }

**Task 1**. Add to the wishLish some number of desired Computer Parts   
Eg. GPU, RTX 3080, 1000, 300, false

**Task 2**. Sort the wishList by price in ascending order

**Task 3**. Print to the screen the part in the wishList the part that has the most and least powerConsumption

**Task 4.** Remove from the wishList all the parts that are 'used'

**Task 5.** Find the average cost of all the parts in the wishList

**Task 6.** Change all the items in the wishList to 'used' and reduce their price by 20% and save this result to a new vector

**Task 7.** Figure out how many of a given part type is in the wishList eg. how many CPUs and print to the screen

**Task 8.** Create new wishList of computer parts and add parts to it. Combine it with the original wishList

Q2. Given the following struct that describes a Gem,

1. struct Gem {
2. string name;
3. string colour;
4. int shine;
5. bool polished;
6. };

 Create a Box class has the following properties:

* An STL container of Gems as a data member
* A += operator that adds Gems to the box
* A -= operator that removes Gems from the Box based on a given string (name) to search for. If the string is found then remove any Gems that match. If there isn’t a match at all then throw an exception of string type with a relevant error message
* A function called process that doesn’t receive any parameters nor returns any value. This function will iterate through the Gems in the Box and polish them if they aren’t yet polished. Polishing a Gem will cause its shine value to increase by 20. For this functionality incorporate the use of threads to divide the work.
* A display function that displays the contents of the box

The following main will use the Box class:

1. int main() {
2. Box b;
4. cout << "\nAdding Gems" << endl;
5. b += Gem{"Emerald", "Green", 20, false};
6. b += Gem{"Ruby", "Red", 50, true};
7. b += Gem{"Sapphire", "Blue", 80, false};
8. b += Gem{"Onyx", "Black", 60, false};
9. b += Gem{"Opal", "White", 10, false};
10. b.display();
12. cout << "\nRemoving Gems" << endl;
13. try {
14. b -= "Ruby";
15. b -= "Emerald";
16. b -= "Diamond";
17. b.display();
18. }
19. catch (string c) {
20. cout << c << endl;
21. }
23. cout << "\nPolishing Gems" << endl;
24. b.process();
25. b.display();

28. cout << "\nVector of Boxes" << endl;
29. vector<Box\*> boxes;
30. boxes.push\_back(new Box());
31. boxes.push\_back(new Box());
32. boxes.push\_back(new Box());
34. for (auto x : boxes) {
35. x->operator+=(Gem{ "Emerald", "Green", 20, false });
36. x->operator+=(Gem{ "Ruby", "Red", 90, false });
37. x->operator+=(Gem{ "Sapphire", "Blue", 40, false });
38. }
40. for (auto x : boxes) {
41. delete x;
42. }
43. }

 As an **extra** **task**, modify the Box class to instead have a container of **Gem pointers** and apply the use of **smart pointers** to the solution.

Q3. There are 5 errors in the following code (Logical or Compile). Identify then in the following format:

Line Number

Type Of Error

Description of Error

Fix for Error

1. // sample.h
3. #include <iostream>
4. #include <string>
5. #include <vector>
7. struct Gem {
8. std::string name{};
9. unsigned weight{};
10. };
12. class TreasureBox {
13. std::vector<Gem> gems;
14. unsigned weight{};
15. public:
16. TreasureBox(const std::string\* names, unsigned\* weights, unsigned size);
17. TreasureBox& operator+=(const Gem& g);
18. void update(const std::string& name);
19. std::ostream& display() const;
20. };
22. // sample.cpp
23. #include <iostream>
24. #include <string>
25. #include <string\_view>
26. #include <vector>
27. #include <algorithm>
28. #include <numeric>
29. #include <thread>
30. #include <future>
31. #include "sample.h"
33. TreasureBox::TreasureBox(const std::string\* names, unsigned\* weights, unsigned size){
34. for (unsigned x = 0; x < size; ++x) {
35. gems.push\_back(Gem{ names[x], weights[x] });
36. weight += weights[x];
37. }
38. }
40. TreasureBox& TreasureBox::operator+=(const Gem& g) {
41. gems.push\_back(g);
42. weight += g.weight;
43. return \*this;
44. }
46. void TreasureBox::update(const std::string& name) {
47. std::vector<Gem>::iterator itr;
48. for (itr = gems.begin(); itr != gems.end(); ++itr) {
49. if (itr->name == name) {
50. unsigned sum = 0;
51. sum = std::accumulate(name.begin(), name.end(), 0);
52. itr->weight += sum;
53. }
54. }
55. }
57. std::ostream& TreasureBox::display() const {
58. std::cout << "TreasureBox Contents" << std::endl;
59. for (auto& g : gems) {
60. std::cout << g.name << " : " << g.weight << std::endl;
61. }
62. return std::cout;
63. }
65. void task(std::promise<unsigned>& p) {
66. Gem gems[]{ {"Red" , 10}, {"Green", 20}, {"Blue", 30} };
68. auto idx = std::find\_if(gems, gems + 2, [](const Gem& g) {
69. return g.name == "Green";
70. });
71. }
73. int main() {
75. std::string strs[]{ "Ruby", "Emerald", "Sapphire", "Opal" };
76. unsigned nums[]{ 10, 20, 30, 40 };
78. std::unique\_ptr<TreasureBox> tbr(new TreasureBox());
79. std::unique\_ptr<TreasureBox> ptr(new TreasureBox(strs, nums, 4));
81. std::string\_view sv = "Ruby";
82. std::string s = "Opal";
83. ptr->update(sv);
84. ptr->update(s);
85. ptr->display();
87. \*ptr += Gem{ "Diamond", 50 } += Gem{ "Amber", 60 };
88. (\*ptr += Gem{ "Pearl", 70 }).display();
90. std::promise<unsigned> pro;
91. std::future<unsigned> fut = pro.get\_future();
93. std::thread t1;
94. t1.join();
95. t1 = std::thread(task, std::ref(pro));
96. std::cout << "Promised Future: " << fut.get() << std::endl;
97. t1.join();
98. }