# Sample Problems

The following code problems are based on real operations in use in parts of Melco’s application software. For each one read the problem statement and implement the desired function in the language of your choice. C++ is preferable as that is the primary language in use in our applications, but other languages are acceptable so long as your solution is solid. Correctly functioning code is most important, but good structure, variable naming, and comments explaining your solution and thought process are also part of the consideration. If you are only able to arrive at a partial solution to one or both problems, then the best thing to do is explain as much of it as you were able to complete and outline where you ran into problems and why

## 1: Matching thread colors

Each embroidery design file specifies the thread colors that should be used by the design. The machine’s GUI software also specifies which thread colors are presently physically loaded on the machine. When loading a design to the machine, the user has the option of manually designating which of the machine’s colors will be used to represent each of the design colors, but we also implement a function to automatically make the best selection for them. The user can choose whether to use an ‘exact match’ or ‘closest match’.

The ’thread’ will be specified in a data structure like this (in C++, but if you choose to use a different language specify a similar structure for yourself)

struct Thread {  
 String manufacturer; // The manufacturer name for this thread type  
 String name; // The user-friendly name for this thread type, i.e. ‘Fire Engine Red’  
 int code; // The manufacturer-specific numeric ID for this thread  
  
 double color; // A floating point value representing the ‘color’ of the thread.  
 double difference(Thread& other); // Returns a floating point value representing the color difference between this Thread and another.  
};

The ’difference’ function returns a floating-point value representing how ‘close’ one color is to another.

For the purpose of this exercise, ‘color’ is a single floating point number and ‘difference’ should simply return the absolute value difference of the ‘color’ values of the two threads. In the real application, colors would be specified using RGB values, and the difference function uses a complex algorithm for computing ‘distance within a 3D visual color space’

If the user desires an ‘exact’ match between two threads, then they must have the same manufacturer, name, and code. The ‘difference’ is unimportant.

If the user desires a ‘closest’ match between two threads, then finding the two that return the smallest ‘difference’ value is desired, and the manufacturer/name/code are unimportant.

### Problem:

Given a single Thread value as input, a Boolean value representing whether the user desires an ‘exact’ or ‘closest’ match, and a list of Thread values available on the machine, return the index of the ‘matching’ thread among the machine threads, or -1 if no match can be found

i.e. in C++:

struct Thread {  
 String manufacturer;  
 String name;  
 int code;  
  
 double color;  
 double difference(Thread& other) {  
 // IMPLEMENT ME  
 }  
};

std::vector<Thread> machineThreads;

int matchingThread(Thread designThread, bool useExactMatch) {  
 // IMPLEMENT ME  
}

You can assume that the machineThreads vector will never be empty and all Thread values have valid contents.

### Example test cases:

Given a ‘machineThreads’ vector with the following contents:

[  
{“ThreadCo”, “Fire Engine”, 1234, 0.5},   
{“ThreadCo”, “Midnight”, 5678, 2.7},  
{“EmbroideryInc”, “Sunlight”, 900, 17.5}  
]

These would be expected return values for the given inputs, with ‘exactMatch’ parameter true/false. Note test case 2: the code/color values changing between different versions of the same thread from the same manufacturer is a real problem we encounter on occasion, as is code values overlapping between different manufacturers because they are manufacturer-specific

{“ThreadCo”, “Fire Engine”, 1234, 0.5} – exact: 0, closest: 0  
{“ThreadCo”, “Midnight”, 5679, 2.6} – exact: -1, closest: 1  
{“Threadhouse”, “Brick”, 100, 0.7} – exact: -1, closest: 0  
{“Threadhouse”, “Sunlight”, 5678, 14.5} – exact: -1, closest: 2

## 2: Rasterizing a vector

Embroidery designs are initially created by specifying a wireframe for the design elements. These wireframes are then filled in algorithmically to create all the stitches to create the actual design element. A simplified version of this process can be represented by rasterizing (converting to a list of multiple points) a single vector between two points.

### Example:

Picture an X/Y coordinate space with 10x10 dimensions. You can visualize the intended result by drawing a line between the start and end points and then ‘filling in’ every point that line crosses

Vector 1: (2,4) 🡪 (5,8) – output shown in red below  
Vector 2: (3,1) 🡪 (8,2) – output shown in yellow below

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0,9 |  |  |  |  |  |  |  |  | 9,9 |
|  |  |  |  |  | 5,8 |  |  |  |  |
|  |  |  |  | 4,7 | 5,7 |  |  |  |  |
|  |  |  | 3,6 | 4,6 |  |  |  |  |  |
|  |  | 2,5 | 3,5 |  |  |  |  |  |  |
|  |  | 2,4 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | 6,2 | 7,2 | 8,2 |  |
|  |  |  | 3,1 | 4,1 | 5,1 |  |  |  |  |
| 0,0 |  |  |  |  |  |  |  |  | 9,0 |

### Problem:

Write a function that takes two integer coordinate points as input and returns a list of all integer coordinate points that lie along the vector between them

i.e. in C++:

struct Point { int x; int y; }  
  
std::vector<Point> rasterize(Point start, Point end){  
 // IMPLEMENT ME  
}

Or write the equivalent in a different language if you prefer

Hint: Look at the visualization and notice that because we are ‘filling’ all the points, it starts and ends at the ‘outer’ corner of each of the end points. Does that alter what values you might use in your calculations?