Write a C++ program that does the following:
Provided a palette of colors, adjust the colors when necessary to optimize the percentage of people who will be able to easily detect the color differences

Creating a program that optimizes color visibility for a given palette involves considering factors like color contrast, accessibility, and color perception. Below is a simple example in C++ that adjusts colors to optimize visibility. This program assumes RGB colors and uses a basic algorithm to improve color contrast. Note that this is a simplified example, and you may need more sophisticated algorithms for a real-world application.

cpp                                                                                Copy code

```cpp
#include <iostream> #include <cmath> #include <vector> struct RGBColor { int
red; int green; int blue; }; double calculateContrast(const RGBColor& color1,
const RGBColor& color2) { // Use a simple formula to calculate color contrast
ratio const double luminance1 = 0.2126 * color1.red + 0.7152 * color1.green +
0.0722 * color1.blue; const double luminance2 = 0.2126 * color2.red + 0.7152 *
color2.green + 0.0722 * color2.blue; const double brighter =
std::max(luminance1, luminance2); const double darker = std::min(luminance1,
luminance2); return (brighter + 0.05) / (darker + 0.05); } void
optimizeColors(std::vector<RGBColor>& palette) { const double targetContrast =
4.5; // Adjust this value based on your requirements for (size_t i = 0; i <
palette.size(); ++i) { for (size_t j = i + 1; j < palette.size(); ++j) { double
currentContrast = calculateContrast(palette[i], palette[j]); // If the current
contrast is below the target, adjust the colors while (currentContrast <
targetContrast) { // You may need a more sophisticated algorithm for adjusting
colors palette[i].red = std::min(255, palette[i].red + 10); palette[i].green =
std::min(255, palette[i].green + 10); palette[i].blue = std::min(255,
palette[i].blue + 10); currentContrast = calculateContrast(palette[i],
palette[j]); } } } } int main() { std::vector<RGBColor> palette = { {255, 0, 0},
// Red {0, 255, 0}, // Green {0, 0, 255}, // Blue // Add more colors to your
palette }; optimizeColors(palette); // Display the optimized palette for (const
auto& color : palette) { std::cout << "RGB(" << color.red << ", " << color.green
<< ", " << color.blue << ")\n"; } return 0; }
```

This example defines a simple RGBColor struct, a function to calculate color contrast, and a function to optimize colors based on a target contrast ratio. Adjust the **`targetContrast`** value based on your requirements. Keep in mind that this is a basic example, and you might need more advanced color adjustment techniques for a production environment.