1.
Napisz funkcję *compressData,* która przyjmuje dwuwymiarową tablicę ***Array2D***, a następnie zwróci ***CompressedData***, zawierający skompresowane dane.

- Jeżeli obok siebie występuje ta sama liczba więcej niż 1 raz, funkcja dodaje do wektora tę wartość (value) i ilość jej wystąpień (count).
- jeżeli występują różne elementy po sobie, funkcja dodaje te wartości do wektora i ilość jej wystąpień (czyli 1).
- Jeżeli kompresja (struktura na wyjściu) miałaby większy rozmiar niż tablica wejściowa, funkcja zwraca pusty Optional.

INPUT: Dwuwymiarowa tablica danych *[Array2D]* do skompresowania.

{{0 0 0 1 1 2 3 },

{0 0 4 4 4 2 2 },

{2 2 2 2 2 1 2 }

OUTPUT: Vector CompressedData zawierająca skompresowane dane.

{{0,3},{1,2},{2,1},{3,1},

{0,2},{4,3},{2,2},

{2,5},{1,1},{2,1}}

```C/C++
template<typename T>
struct Data {
  T value;

  uint8_t count;
};

constexpr size_t sizeX{32};
constexpr size_t sizeY{16};

template<typename T>
using Array2D = std::array<std::array<T, sizeX>, sizeY>;
template<typename T>
using CompressedData =
std::optional<std::vector<Data<T>>>;

template<typename T>
CompressedData <T> compressData(const Array2D<T>& input) {
  // Implementation of the function
}
```

2.

Jesteś odpowiedzialny za zaprojektowanie i implementację automatu biletowego dla Krakowskiego transportu publicznego. Automat wydaje resztę tylko przy pomocy monet o nominałach 1, 2, 5.

Automat ma następujące stany:

Przyjmowanie pieniędzy od użytkownika: Automat przyjmuje pieniądze od użytkownika w pełnych złotych. Aby ułatwić parsowanie wejścia automat od użytkownika przyjmuje dowolne nominały monet.

Oczekiwanie na wybór biletu: Automat oczekuje na wybór biletu przez użytkownika po wrzuceniu monet. Dostępne ceny biletów to: 3, 4, 8, 10, 15.

Drukowanie biletu: Automat drukuje bilet po wyborze biletu przez użytkownika. Proszę to zasygnalizować w dowolny sposób.

Wydawanie reszty: Automat wydaje resztę używając najmniej monet.

Bilet wydrukowany: Automat kończy proces wydawania biletu, życzy użytkownikowi miłego dnia i wraca do stanu początkowego.

Premiowane będzie użycie poprawnych wzorców projektowych, kreatywność, odpowiednia walidacja, obsługa błędów, testy, wygląd, interfejs automatu oraz ogólność rozwiązania i intuicyjne poruszanie się po automacie. Dodatkowo użytkownik może się cofać do stanu początkowego i poprzedniego.

Przykładowe działanie programu:

A: “Witaj, z tej strony automat biletowy miasta Kraków, oczekuję wrzucenia do mnie pieniędzy”

U: 5, 2, 7, 10, 2, 1

A: “Twoje saldo wynosi: 27, wybierz bilet który chcesz kupić 3, 4, 8, 10, 15”

U: “Stan poprzedni”

A: “Oddaję Twoje monety”

A:“Witaj, z tej strony automat biletowy miasta Kraków, oczekuję wrzucenia do mnie pieniędzy”

U: 5, 2, 7, 10, 2, 1

A: “Twoje saldo wynosi: 27, wybierz bilet który chcesz kupić 3, 4, 8, 10, 15”

U: 10

A: “Trwa proces drukowania biletów *BRRR* *BRRR* *BRRR*”

A: “Bilet został wydrukowany oto Twoja reszta: 3*5 zł, 1*2 zł, 0*1 zł

A: “Życzę miłego dnia Twój bilecik 10 oczekuje na odbiór”

A: “Witaj, z tej strony automat biletowy miasta Kraków, oczekuję wrzucenia do mnie pieniędzy”

3.
Zaimplementuj funkcję `isColliding` zwracającą `true` jeśli dwa trójkąty w przestrzeni 2D kolidują ze sobą. Następnie napisz wizualizator tej kolizji. Trójkąty powinny być widoczne na ekranie i co najmniej jeden z nich możliwy do sterowania przez użytkownika. Kolizja powinna być w jakiś sposób zasygnalizowana użytkownikowi.

```C/C++
struct vec2
{
    float x, y;
}
struct triangle
{
    std::array<vec2, 3> points;
}
bool isColliding(const & triangle1, const & triangle2 );
```

Porządnie zrobiony cmake oraz testy do każdego zadania są dodatkowo punktowane!