1. Problem Description

This project concerns using multi-thread logic to schedule events of all elevators in one building. The system should make whole process efficient and make sure the carts schedule fair enough for every passenger.

2. Structures Definition

- This system should have a vector which is used to store all request from each floor. And the vector should implement as FIFO which means that passenger who press early will be assigned cart earlier than other.
- For each floor except the lowest floor and the highest floor only have two buttons which is up and down button, and those two bottoms will notify all carts when passenger press button.
- For each cart have m buttons which on behalf of each floor.

3. Events Definition

- **Processing floors' requests**: Determine with cart should be response this request.
- **Showing cart moving direction**: The result will be used by processing floors' requests and schedule cart's work.
- **Show carts' position**: This method should show each cart's position for waiting passengers in each floor.

4. Classes Definition

- **Cart Class**:
  Parameters:
  (1) Capacity(int): Means the max headcount of this cart.
  (2) Current headcount(int): Means current number of people in this cart.
  (3) Direction(bool): Means current moving direction (up or down).
  (4) State(bool): Means cart moving or stop.
  (5) Buttons(vector): Means all buttons which represent each floor in the cart. And the index of vector means floor number minus one.
  (6) Requests for up (max heap): When the state is up using a max heap store all requests and pop request in sequence.
  (7) Requests for down (min heap): When the state is down using a min heap store all requests for pop request in sequence.
  (8) Range(vector<int>): Indicate the range of levels that the cart could receive.
  (9) Symbols(vector<char>): Due to all carts are design max carry 20 people, so we use 20 symbols represent different people in each cart for display purpose.

Methods:

(1) Could Join: Check new request whether could be joined or not. And printing a warning message and when a new request will cause cart overload.
(2) Update state: Switching state between move and stop.
(3) Update direction: Switching direction between up and down.
(4) Check state: For test using, Checking request queue valid or not. For example, when cart direction is upping, the down request queue should be empty, vice versa.

- **Controller Class**:
  Parameters:
  (1) Container of Carts(vector): Include all carts object which in current building.
  (2) Number of Carts(int)

  Methods:

  (1) Add Cart: Push Cart instance to container of Carts and update Carts' number.
  (2) Cart selector: Base on the passenger's requirement and Carts' situations choose the cart that will pick up current passenger.
  (3) Add task: Add task to Cart which is selected, if no Cart could take this task then send this task to task buffer.

- **Passenger Request**:
  Parameters:
  (1) Weigh(int): The weight of current passenger.
  (2) From(int): Current passenger's starting level.
  (3) To(int): Current passenger's destination level.
  (4) Direction(int): Current running direction.
  (5) Symbol(char): When this request put into one cart, It will be assigned one symbol for display purpose.

5. Functions

- Requests Generator (return deque<passengrerReq>): Use level number and request number generate a set of requests for test purpose.
- Requests allocator: Simulate real passengers enter the building in different period and tell the controller their requests.
- Request processor: Gather all current info from public requests pool and private requests pool in controller. Then call the function cartSelector() in Controller class schedule missions for different carts.
- Run one cart: This function gets the pointer of one cart, then based on the requests in this cart and the direction run this cart.
- Run all carts: It will create carts' number's thread to run each cart in the building.
- Print building framework: Get the levels number and carts number plot this building's frame in console.
- Print carts: Get all carts' info about current level and people symbols in each cart, then print it in console.

6. About main construction

- Gather user input about carts number, levels and requests number
- Print building framework
- Calculate all carts' initial position's coordinate
- Create four threads:
  1) Allocate requests which will simulate real passengers enter and press bottoms at different time.
  2) Controller process requests in this building and send them to cart which could accept those requests.
  3) Run carts which will run all carts in this building
  4) Print carts which will monitor all carts' state and update it in console.

7. Algorithm Analysis

- Fairness analysis: The base rule of requests schedule is first pressing key first processing. Each time controller process requests in requests pool will depend on the sequence that requests enter the pool. Obviously, all requests which could be accept by cart will leave requests pool. For those requests which can not be scheduled at that time it will come back to requests pool and keep origin order. Also, they have higher priority than any new incoming requests.
- Efficiency analysis:
  1) Pick up efficiency: For making sure that using the closest cart which have the same direction as this request, controller will check all carts in same direction and it range include this request's start level, and select the closest cart pick up this passenger.
  2) Running efficiency: All carts will not keep upping and downing for get the closest and ignore other requests which have longer distance to their destination. Once cart accept first request the direction that direction will keep until all requests in it run out.

8. How to run

- Initialize carts: Input carts number and levels for generate correct carts.
- Initialize total requests: Input total requests number. When all requests have been processed the program will end. (Both reqs and buffer).