1. 총 층의 정보: 1층 ~ 5층
2. EV 정보
   1. 최대 하중 1500kg
   2. 한 사람의 무게는 60kg으로 가정
   3. EV ~~opening/closing 시간은 각각 2초씩,~~ while\_open 시간은 10초
   4. EV 한 층 이동 시간은 3초
   5. Schedule 종류
      1. Independent\_Schedule
         1. Input\_Info
      2. Dependent\_Schedule
         1. Move
         2. Open\_While
3. Main 함수의 개략적인 구조

Int tick;

void main(int argc, char \*\*argv){

tick = 0;

linkedlist() 파일정보 입력

while(1){

tick++;

…

}

}

1. EV 구조체

struct EV;

----------------

int cur\_level;

int direction;

int weight;

1. 외부에서 입력되는 데이터 단위
   1. 탑승 이벤트에 대한 정보: struct Input\_Info;

struct Input\_Info

------------------------

Int tick;

Int load\_level;

Int direction;

Int passenger;

Int unload\_level[ ];

1. Scheduler에 Input\_Info 연결  
   - 외부에서 탑승 이벤트가 입력되면. 그 이벤트는 tick 값의 오름차순 순서로 Scheduler에 배정
   1. 입력이 없는 초기 상태

head

tail

* 1. 외부에서 입력된 탑승 이벤트가 Scheduler에 배정된 상태

# Scheduler.tick = Input\_Info.tick;

Scheduler Linked List

---------------------------

int tick;

Scheduler \* next;

Input\_Info \*Info;

int Door\_SW;

Scheduler Linked List

---------------------------

int tick;

Scheduler \* next;

Input\_Info \*Info;

int Door\_SW;

head

tail

struct Input\_Info

----------------------

int tick;

int load\_level;

int direction;

int passenger;

int unload\_level[ ];

input\_Info \*next;

struct Input\_Info

----------------------

int tick;

int load\_level;

int direction;

int passenger;

int unload\_level[ ];

input\_Info \*next;

1. Scheduler.Info->Input\_Info를 Track에 In\_Event로 전달
   1. main함수의 tick과 Scheduler 의 tick 이 일치 할 때,   
      Scheduler.Info->Input\_Info로부터 정보를 받아 In\_Event 생성하여 Track구조체 배열의 해당 level 뒤에 연결

struct In\_Event;

---------------------

int direction;

int passenger;

int unload\_level[ ];

In\_Event \*next;

Track Linked List

----------------------

In\_Event \*In\_ptr;

int Out\_passenger;

1. Move 이벤트의 생성 & 배정
   1. Move 이벤트는 Track[EV.cur.level].level == EV.cur\_level을 만족 할 때 까지   
      연속적으로 ++tick에 추가적인 Move 이벤트 생성한 뒤  
      ~~남아 있는 모든 Dependent\_Schedule\*의 int tick 값에 +3 해준다.~~

struct Input\_Info

----------------------

int tick;

int load\_level;

int direction;

int passenger;

int unload\_level[ ];

Struct Move

-----------------

int tick;

void Move( );

Scheduler Linked List

---------------------------

int tick;

Scheduler \* next;

Input\_Info \*Info;

Event \*Move;

Event \*Open;

head

tail

Scheduler Linked List

---------------------------

int tick;

Scheduler \* next;

Input\_Info \*Info;

Event \*Move;

Event \*Open;

1. Move 뒤따라 Out\_Event 배정
   1. Track[EV.cur.level].level == EV.cur\_level && Track.In\_ptr != NULL조건을 만족할 때,   
      Track에 연결되어 있는 In\_Event.unload\_level[ ] 데이터를 바탕으로  
      Out\_Event를 해당 층에 각각 배정

struct In\_Event;

---------------------

int direction;

int passenger;

int unload\_level[ ];

In\_Event \*next;

Track Linked List

----------------------

int level;

In\_Event \*In\_ptr;

int Out\_passenger;

Track Linked List

----------------------

int level;

In\_Event \*In\_ptr;

int Out\_passenger;

Track Linked List

----------------------

int level;

In\_Event \*In\_ptr;

int Out\_passenger;

struct In\_Event;

---------------------

int direction;

int passenger;

int unload\_level[ ];

In\_Event \*next;

1. While\_Open 이벤트 생성 & 배정
   1. Track[EV.cur.level].level == EV.cur\_level && Track.In\_ptr != NULL조건을 만족할 때,  
      (7 에서 이미 만족한 조건)
   2. While\_Open 이벤트를 생성하여 ++tick 에 배정 한 뒤,   
      (그 전 까지는 ++tick에 Move 생성하여 배정을 반복하는 과정이었음)
   3. Move 이벤트 생성하여 tick+2에 배정
   4. ++tick 뒤의 남아있는 모든 Dependent\_Schedule의 int tick 값에 +10 해준다.   
      ~~(단 10 tick 동안 같은 층에서 Independent\_Schedule 있다면 +10 tick 재조정)~~

Struct Move

-----------------

int tick;

void Move( );

Struct While\_Open

---------------------

int tick;

void While\_Open( );

Scheduler Linked List

---------------------------

int tick;

Scheduler \* next;

Input\_Info \*Info;

Event \*Move;

Event \*Open;

Scheduler Linked List

---------------------------

int tick+3;

Scheduler \* next;

Input\_Info \*Info;

Event \*Move;

Event \*Open;

Scheduler Linked List

---------------------------

int tick+10;

Scheduler \* next;

Input\_Info \*Info;

Event \*Move;

Event \*Open;

1. Move 뒤따라 EV 정지
   1. Track[EV.cur.level].level == EV.cur\_level && Track.Out\_ptr != NULL 조건 만족할 때,
   2. While\_Open 이벤트를 생성하여 ++tick 에 배정 한 뒤,   
      (그 전 까지는 ++tick에 Move 생성하여 배정을 반복하는 과정이었음)
   3. 뒤따라 오는 Dependent\_Schedule 이 없으므로,   
      ~~(뒤따라 오는 Dependent\_Schedule의 tick에 +10 조정 해줄 필요 없음)~~
   4. 다음 Independent\_Schedule의 tick이 올 때 까지 정지
   5. 다음 Independent\_Schedule이 오면 5번부터 반복
2. EV가 2대일 때

struct EV\_1;

----------------

int cur\_level;

int direction;

int weight;

struct EV\_2;

----------------

int cur\_level;

int direction;

int weight;