IOT 4차 중간 발표

7조

DIP SW 어댑터

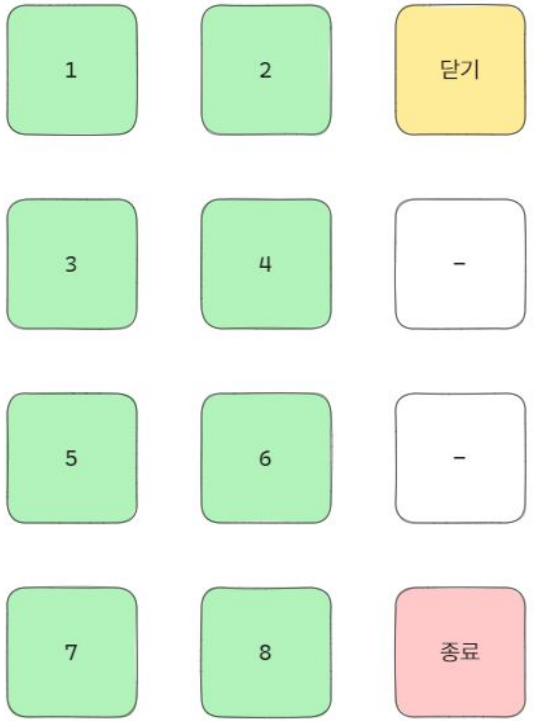
```
36  int getDIP() {
37      unsigned char b;
38      int dip = -1;
39      dip = open(DIP, O_RDONLY);
40      if (dip < 0) {
41          printf("dip device error\n");
42          return -1;
43      }
44      read(dip, &b, sizeof(b));
45      close(dip);
46      switch (b) {
47      case 1: //외부 1층 버튼
48          return 1;
49      case 2:
50          return 2;
51      case 4:
52          return 3;
53      case 8:
54          return 4;
55      case 16:
56          return 5;
57      case 32:
58          return 6;
59      case 64:
60          return 7;
61      case 128: //외부 8층 버튼
62          return 8;
63      default:
64          return -1;
65      }
66  }
```

- getDIP 함수
- DIP 스위치의 입력을 감지
  - 왼쪽부터 2^0 ~ 2^7 까지 입력받을 수 있게 하는 코드

TACT SW 어댑터

```c
int getTACT() {
    unsigned char b;
    int tactswFd = -1;
    tactswFd = open(TACT, O_RDONLY);
    if (tactswFd < 0) {
        printf("tact device error\n");
        return -1;
    }
    read(tactswFd, &b, sizeof(b));
    close(tactswFd);
    switch (b) {
    case 1: // 내부 1번 버튼
        return 1;
    case 2: // 내부 2번 버튼
        return 2;
    case 3: // 내부 닫기 버튼
        printf("close button pushed!\n");
        return 200;
    case 4: // 내부 3번 버튼
        return 3;
    case 5: // 내부 4번 버튼
        return 4;
    case 7: // 내부 5번 버튼
        return 5;
    case 8: // 내부 6번 버튼
        return 6;
    case 10: // 내부 7번 버튼
        return 7;
    case 11: // 내부 8번 버튼
        return 8;
    case 12: // 내부 종료 버튼
        return 400;
    default:
        return -1;
    }
}
```

- getTACT 함수
- TACT Switch 입력을 감지



printVector

```cpp
void printVector(const std::vector<int>& vec) {
    for (size_t i = 0; i < vec.size(); ++i) {
        printf("%d ", vec[i]);
    }
    printf("\n");
}
```

- printVector 함수
  - upArr, downArr
    (엘리베이터의 목적지를 저장해 놓은 배열)
  - 이를 디버깅 하기 위해 만든 편의성 메서드

printCLCD

```
// CLCD에 출력
void printCLCD(string S) {
    char* cstr = new char[S.length() + 1];
    strcpy(cstr, S.c_str());
    int clcd_fd = open(CLCD, O_WRONLY);
    write(clcd_fd, cstr, 32);
    close(clcd_fd);
    delete[] cstr;
    printf("CLCD Output: %s\n", S.c_str());
}
```

- printCLCD 함수
  - CLCD에 text를 출력해주기 위한 함수

CLCDHelper

```
// CLCD에 변수를 넣어 출력할 때 도우미
string CLCDHelper(int I, int space) {
    char s[10];
    sprintf(s, "%d", I);
    string STR(s);

    for (int i = STR.length(); i < space; i++) {
        STR = '0' + STR;
    }
    return STR;
}
```

- CLCDHelper 함수
  - CLCD에 엘리베이터의 층수를 보여주기 위한 편의성 함수

EV_moving_CLCD, EV_Idle_CLCD

```
void EV_moving_CLCD(int current_floor) {
    string s1 = "  EV is moving  ";
    string s2 = "Current : " + CLCDHelper(current_floor, 4) + "  ";

    printCLCD(s1 + s2);
}
```

```
void EV_Idle_CLCD(int current_floor) {
    string s1 = "  EV is stopped ";
    string s2 = "Current : " + CLCDHelper(current_floor, 4) + "  ";

    printCLCD(s1 + s2);
}
```

- EV_moving_CLCD, EV_Idle_CLCD
  - 엘리베이터가 움직이고 있는(멈춰있는) 현재 층을 CLCD에 출력해주는 함수 (moving, stopped)

EV 클래스 + 생성자

```
class EV {
public:
    int current_floor;
    vector<int> upArr;
    vector<int> downArr;
    bool isDirectionUp;
    bool isMoving;

    EV() {
        current_floor = 1;
        isDirectionUp = true;
        isMoving = false;
        printf("Elevator initialized at floor %d\n", current_floor);
        EV_Idle_CLCD(current_floor);
    }
```

- EV 클래스
  - 엘레베이터 구현을 위한 클래스 선언부분
- EV() 생성자
  - 현재 층 수: 1층 초기화
  - 방향: 상승으로 초기화
  - 움직임 : 정지로 초기화
  - CLCD : idle 상태 표시

addTarget

```
void addTarget(int floor) {
    if (floor > current_floor) {
        if (find(upArr.begin(), upArr.end(), floor) == upArr.end()) {
            upArr.push_back(floor);
            sort(upArr.begin(), upArr.end());
            printf("Target floor added(upArr): %d\n", floor);
            printVector(upArr);
        }
    }
    else if(floor < current_floor) {
        if (find(downArr.begin(), downArr.end(), floor) == downArr.end()) {
            downArr.push_back(floor);
            sort(downArr.begin(), downArr.end(), std::greater<int>()); // error warning !!!!! : cpp grammar
            printf("Target floor added(downArr): %d\n", floor);
            printVector(downArr);
        }
    }
}
```

- 타겟 배열(upArr, downArr)에 층 추가하기
- 호출한 층과 현재 층(current_floor)과 비교
- 타겟 배열에 층 추가 후, 배열 특성에 맞게 정렬

move() 함수

```
void move() {
    if (upArr.size() + downArr.size() > 0) {
        printf("Starting to move.\n");
        printf("upArr : ");
        printVector(upArr);
        printf("downArr : ");
        printVector(downArr);
        printf("current floor : %d\n",current_floor);
        if (isMoving) {
            if (isDirectionUp) {
                // 위로 상승
                if (current_floor < upArr.front()) { ... }
                // 지금 층 == 목적지
                else if (current_floor == upArr.front()) { ... }
                // 아래로 하강(더 이상 위로 못감)
                else { ... }
            }
            else {
                // 아래로 하강
                if (current_floor > downArr.front()) { ... }
                // 지금 층 == 목적지
                else if (current_floor == downArr.front()) { ... }
                // 위로 상승(더 이상 아래로 못감)
                else { ... }
            }
        }
    }
```

- 엘리베이터가 움직이는 알고리즘
- 주요조건 : 타겟 배열, 움직임 상태, 방향
- 뱡향에 따라 현재 층과 목표 층을 비교하여 엘레베이터의 움직임을 결정함.

move() 함수 - 상승 중

```
if (isDirectionUp) {
    // 위로 상승
    if (current_floor < upArr.front()) {
        current_floor++;
        EV_moving_CLCD(current_floor);
        usleep(TIME_QUANTUM);
        if (current_floor == upArr.front()) {
            upArr.erase(upArr.begin());
            isMoving = false;
            if (upArr.empty()) {
                isDirectionUp = false;
                printf("direction Changed(1) (up -> down)\n");
            }
            EV_Idle_CLCD(current_floor);
        }
    }
```

- 추가할 층이 현재 층보다 높은 경우
- 1층 올라가기
- 1초 대기
- 타겟 도착 여부 확인
  - 배열에서 해당 층 삭제
  - 움직임 : 정지
  - 배열이 비어있는 여부
    - 방향전환

move() 함수 - 도착

```
// 지금 층 == 목적지
else if (current_floor == upArr.front()) {
    upArr.erase(upArr.begin());
    isMoving = false;
    if (upArr.empty()) {
        isDirectionUp = false;
        printf("direction Changed(2) (up -> down)\n");
    }
    EV_Idle_CLCD(current_floor);
    printf("Arrived at floor: %d\n", current_floor);
}
```

- 엘리베이터의 현재 층 == 목적지
  - 배열에서 해당 층 삭제
  - 움직임 : 정지
  - 배열이 비어있는 여부
  - 방향전환

move() 함수 - 상승 중

```
// 아래로 하강(더 이상 위로 못감)
else {
    isDirectionUp = false;
    printf("direction Changed(3) (up -> down)\n");
    current_floor--;
    EV_moving_CLCD(current_floor);
    usleep(TIME_QUANTUM);
    if (current_floor == downArr.front()) {
        downArr.erase(downArr.begin());
        isMoving = false;
        if (downArr.empty()) {
            isDirectionUp = true;
            printf("direction Changed(4) (down -> up)\n");
        }
        EV_Idle_CLCD(current_floor);
        printf("Arrived at floor: %d\n", current_floor);
    }
}
```

- 추가할 층이 현재 층보다 낮은 경우
- 방향 전환
- 1층 내려가기
- 1초 대기
- 타겟 도착 여부 확인
  - 배열에서 해당 층 삭제
  - 움직임 : 정지
  - 배열이 비어있는 여부
    - 방향전환
-

Main

```
int main() {
    EV elevator;
    while (true) {
        int dip_input = getDIP();
        int tact_input = getTACT();

        if (dip_input != -1) {
            elevator.addTarget(dip_input);
        }

        if (tact_input == 400) {
            printf("Program exit command received.\n");
            break;
        }
        else if (tact_input != -1) {
            if (tact_input == 200) {
                // 문 닫기 버튼
                elevator.isMoving = true;
            }
            else {
                elevator.addTarget(tact_input);
            }
        }
        if(elevator.isMoving){
            elevator.move();
        }


    }
    return 3;
}
```

- 엘레베이터 객체 선언
- 작동시작 : 무한루프
  - DIP, TACT 입력받기
  - DIP 층 입력 존재 : addTarget 호출
  - Tact 입력 존재
    - 종료 버튼(400) : 무한루프 중단
    - 닫기 버튼(200) : EV 움직임 상태 true로 변환
    - 층 입력 존재 : addTarget 호출
  - 움직임 상태 == true : EV 움직임 시작