AJOU UNIV. 1973

C 프로젝트

-교착 상태 예방-

전자공학 프로그래밍

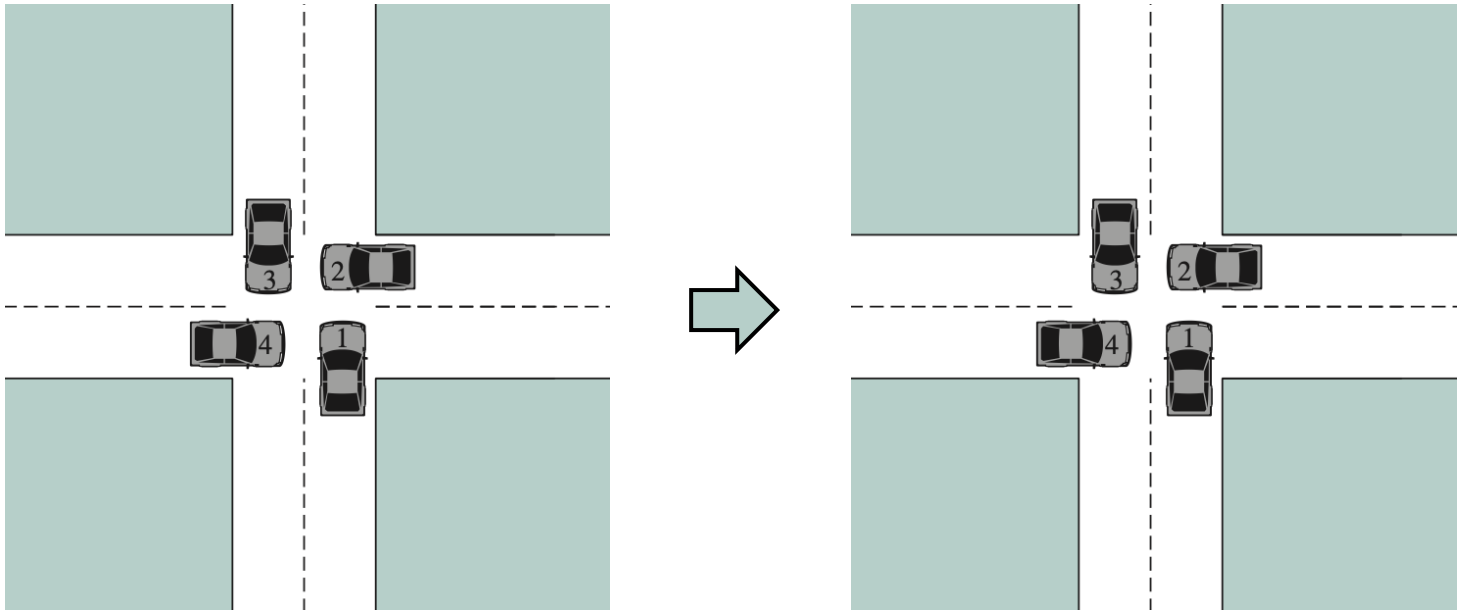
Embedded SoftwareLab

C 프로젝트. 교착 상태 예방

- 프로젝트 설명
- 구현 결과 예시

C 프로젝트. 교착 상태 예방

- 교착 상태 (Deadlock)
  - 정의 : 두 개 이상의 작업이 상대방의 작업이 끝나기만을 기다리고 있는 상태
    - 교차로에서 각 자동차가 다른 자동차가 움직이기만을 기다리며 무한 대기



C 프로젝트. 교착 상태 예방

- 교착 상태 예방 (Deadlock Prevention)
  - 교착 상태가 일어나지 않도록 사전에 방지하는 방법
  - 교착 상태의 발생 조건 4가지 중 하나를 부정함으로써 교착 상태를 예방

- 교착 상태의 발생 조건 4가지
  - 상호 배제 (Mutual Exclusion)
    - 한 번에 한 개의 프로세스만이 공유자원을 사용할 수 있음
  - 점유 대기 (Hold and Wait)
    - 프로세스가 할당된 자원을 가진 상태에서 다른 자원을 기다림
  - 비선점 (No Preemption)
    - 프로세스는 작업을 마친 후 자원을 반환
    - 이미 다른 프로세스에게 할당된 자원을 강제적으로 빼앗을 수 없음
  - 순환 대기 (Circular Wait)
    - 프로세스의 자원 점유 및 점유된 자원의 요구 관계가 원형을 이루면서 대기
    - 각 프로세스가 순환적으로 다음 프로세스가 요구하는 자원을 갖고 있음

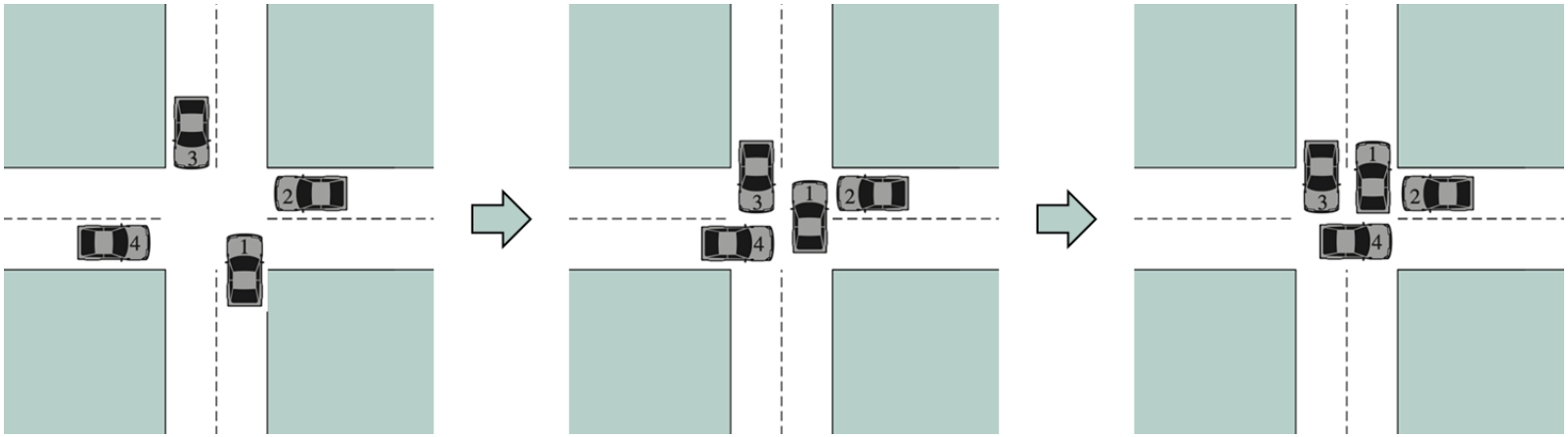
C 프로젝트. 교착 상태 예방

- 프로젝트 설명
  - 목표
    - 교착 상태가 발생하지 않도록 자동차를 시작점에서 목적지까지 이동
    - 주어진 조건을 만족하는 교착 상태 예방 방법을 통해 이동
  - 교착 상태 예방 방법
    - 상호 배제 (Mutual Exclusion) 제거
      - **본 프로젝트에서는 사용하지 않음**
    - 점유대기(Hold and Wait) 제거
      - 점유대기 : 자동차가 현재 위치를 점유한 상태에서 다음 위치를 점유하기 위해 대기한다.
    - 비선점(No Preemption) 제거
      - 비선점 : 이동하려는 위치에 자동차가 있다면, 그 위치로 이동할 수 없으며 대기한다.
    - 순환대기(Circular Wait) 제거
      - 순환대기 : 순환적으로 각 자동차가 요구하는 위치는 다음 번 이동할 자동차가 점유하고 있다

프로젝트 설명

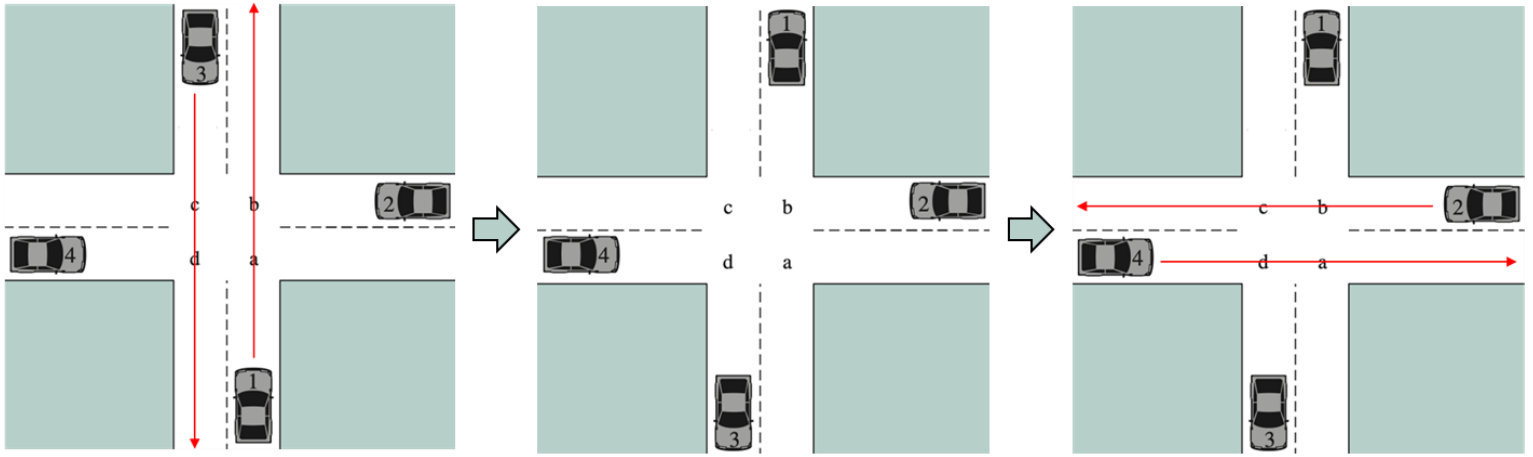
- **자동차의 기본 이동 조건**
  - 1. 모든 자동차는 1초마다 1칸씩 이동할 수 있음
  - 2. 같은 시간 내 자동차 이동 순서는 자동차의 번호에 따라 순서대로 이동
  - 3. 자동차의 이동 위치에 다른 자동차가 있을 경우 해당 자동차는 이동 불가

  - 이동 예시
    - 시간 t에 모든 자동차가 1칸씩 앞으로 이동하려고 시도
      - 자동차의 이동 순서는 낮은 번호부터 순서대로 움직임(1->2->3->4)
      - 1번의 1칸 이동으로 2번은 대기
      - 3, 4번 1칸 이동



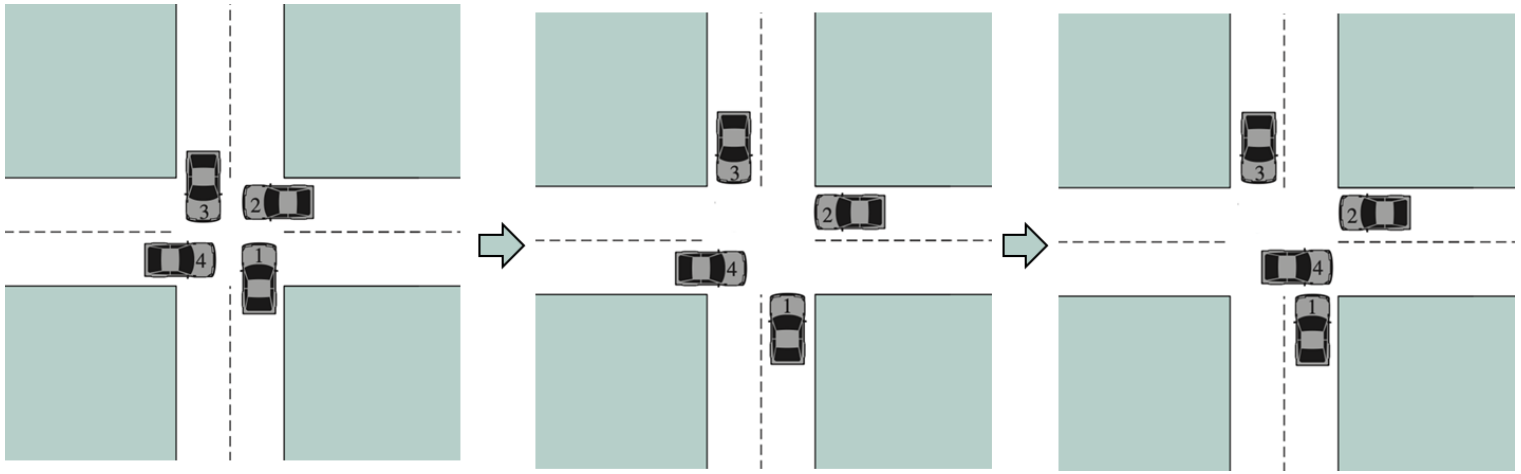
프로젝트 설명

- **점유대기(Hold and Wait) 제거**
  - 점유대기 : 자동차가 현재 위치를 점유한 상태에서 다음 위치를 점유하기 위해 대기
  - 예방 방법 : 시작점 to 도착점의 도로(모든 칸)를 점유한 상태로 이동
  - 이동 예시
    - 시간 t에 모든 자동차가 1칸씩 앞으로 이동하려고 시도
      - 1번 : 도로(직진 방향의 모든 위치) 점유 후 1칸 이동
      - 2번 : 이동하려고 시도하지만, b 위치가 이미 점유되어 있어 도로 점유 실패 후 대기
      - 3번 : 도로 점유 후 1칸 이동
      - 4번 : 이동하려고 시도하지만, a 및 d 위치가 이미 점유되어 있어 도로 점유 실패 후 대기
    - 1번 및 3번 자동차가 목적지에 도달하여 도로 점유 해제 후 2번, 4번 이동 가능



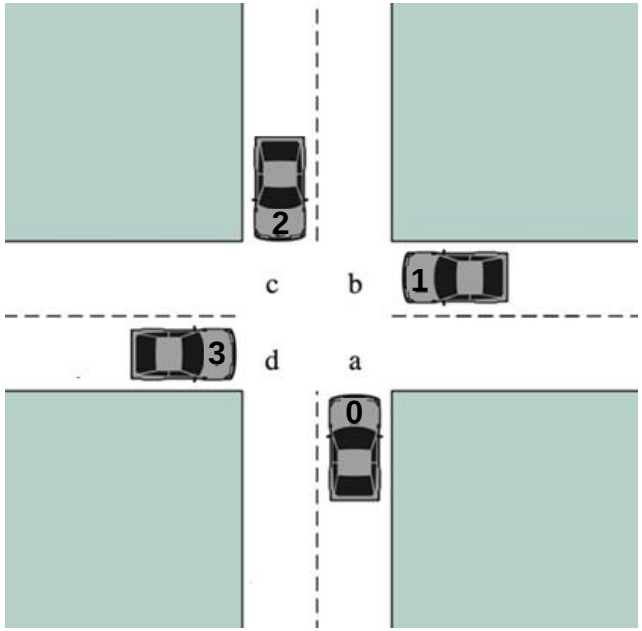
프로젝트 설명

- **비선점(No Preemption) 제거**
  - 비선점 : 움직이려는 위치에 자동차가 있으면, 그 위치로 이동 불가
  - 예방 방법 : 움직이려는 다음 위치가 점유중일 경우, 1칸 후진
  - 이동 예시
    - 시간 t에 모든 자동차가 1칸씩 앞으로 이동하려고 시도
      - 1번 :이동하려는 위치를 2번이 점유하고 있으므로, 1칸 후진
      - 2번 : 이동하려는 위치를 3번이 점유하고 있으므로, 1칸 후진
      - 3번 : 이동하려는 위치를 4번이 점유하고 있으므로, 1칸 후진
      - 4번 : 1칸 이동



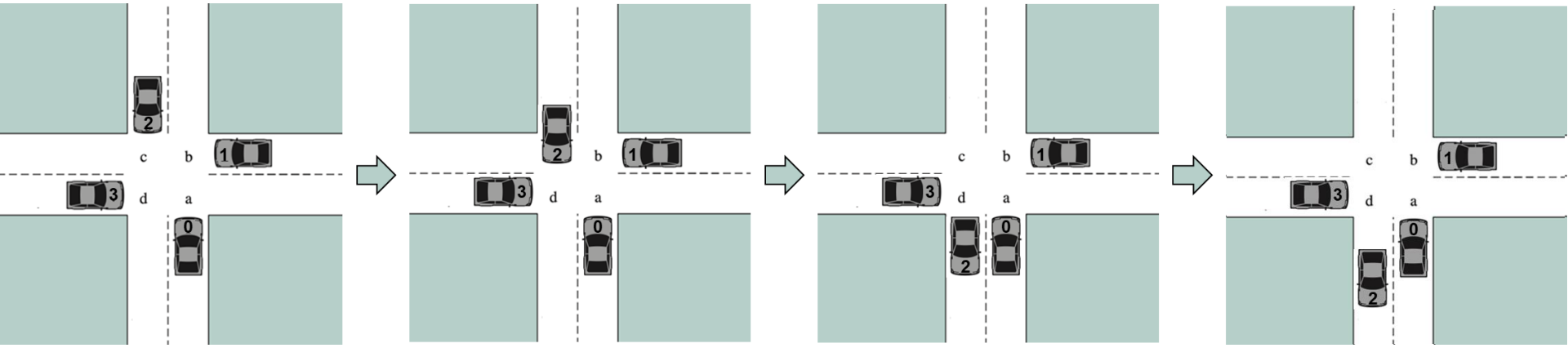
프로젝트 설명

- **순환대기(Circular Wait) 제거**
  - 순환대기 : 각 자동차가 요구하는 위치는 다음 번 이동할 자동차가 점유
  - 예방 방법 : 교차로(4칸)를 공유자원으로 설정 후 교차로 진입 순서 배정
    - 교차로 (a,b,c,d)에 접근, 이동 가능한 경우
      - 1. 이미 교차로를 지나고 있는 자동차가 교차로를 벗어날때 까지 점유 유지
      - 2. 교차로를 점유한 자동차가 없으며, 교차로 진입 가능 순서에 해당 되는 자동차
        - 진입 가능 순서 : 매초마다 ((‘학번’ + ‘0123’) % 4)의 자리수 숫자에 해당하는 자동차 진입 가능
        - E.g. 학번:202124425 -> 2021244250123 -> 2021200210123



프로젝트 설명

- **순환대기(Circular Wait) 제거**
  - 이동 예시
    - 학번 : 202124425일 경우 202120021’0123’ 순서로 교차로 진입 가능
    - 시간 1초에 모든 자동차가 1칸씩 앞으로 이동하려고 시도
      - 0, 1, 3번 : 교차로 진입 불가 : 대기
      - 2번 : 교차로 진입가능 : 1칸 이동
    - 시간 2초 ~ 4초까지 2번 자동차가 교차로 통과중이므로 2번만 계속 이동
    - 시간 5초 : 2번 자동차 교차로 진입 가능
      - 0,1,3번 : 교차로 진입 불가
      - 2번 : 1칸 이동
    - 시간 6초 : 0번 자동차 교차로 진입 가능



프로젝트 설명

- 코드 설명
  - car_info 구조체
    - 자동차에 대한 정보
    - car : 자동차 이름. 전체 도로 출력시 자동차를 표시하기 위한 변수
    - direction : 진행 방향. 입력 받은 순서대로 0, 1, 2, 3(상하좌우)의 값을 가짐
    - using_load : 자동차가 사용하는 도로
    - current_position : 자동차의 현재 위치
    - desination : 자동차의 목적지
    - hold : 자동차의 도로 점유 유무

```
typedef struct Car_info {
        char car;
        int direction;
        int using_load;
        int current_point[2];
        int destination[2];
        int hold;
} Car_info;
```

  - void init_load(char** load), void print_load(char** load)
    - 실습1, 과제1 참고

프로젝트 설명

- 구현 내용 설명
  - Car_info* createCar(char** load, char car, int direction, int using_load)
    - 자동차 이름, 및 사용 도로, 방향 정보를 Car_info 구조체 형태로 리턴
    - Main문에서 입력받은 4대의 자동차는 순서대로 0,1,2,3(상하좌우)의 이동방향을 가짐
    - 상하좌우 방향에 따라 이동할 도로의 시작 위치와 도착점 결정
      - E.g. 첫 번째로 Car : 'A', using_load : 3 일 경우, 마지막 행의 3번 째 열이 시작위치
    - 구조체의 hold 변수는 -1로 초기화
  - void update_ load(char** load, Car_info* car)
    - 시간에 따라 이동한 자동차의 위치를 바탕으로 load 변수 수정
    - 이중포인터로 나타낸 2차원 배열 load의 값을 car_info 구조체 배열을 통해 업데이트

프로젝트 설명

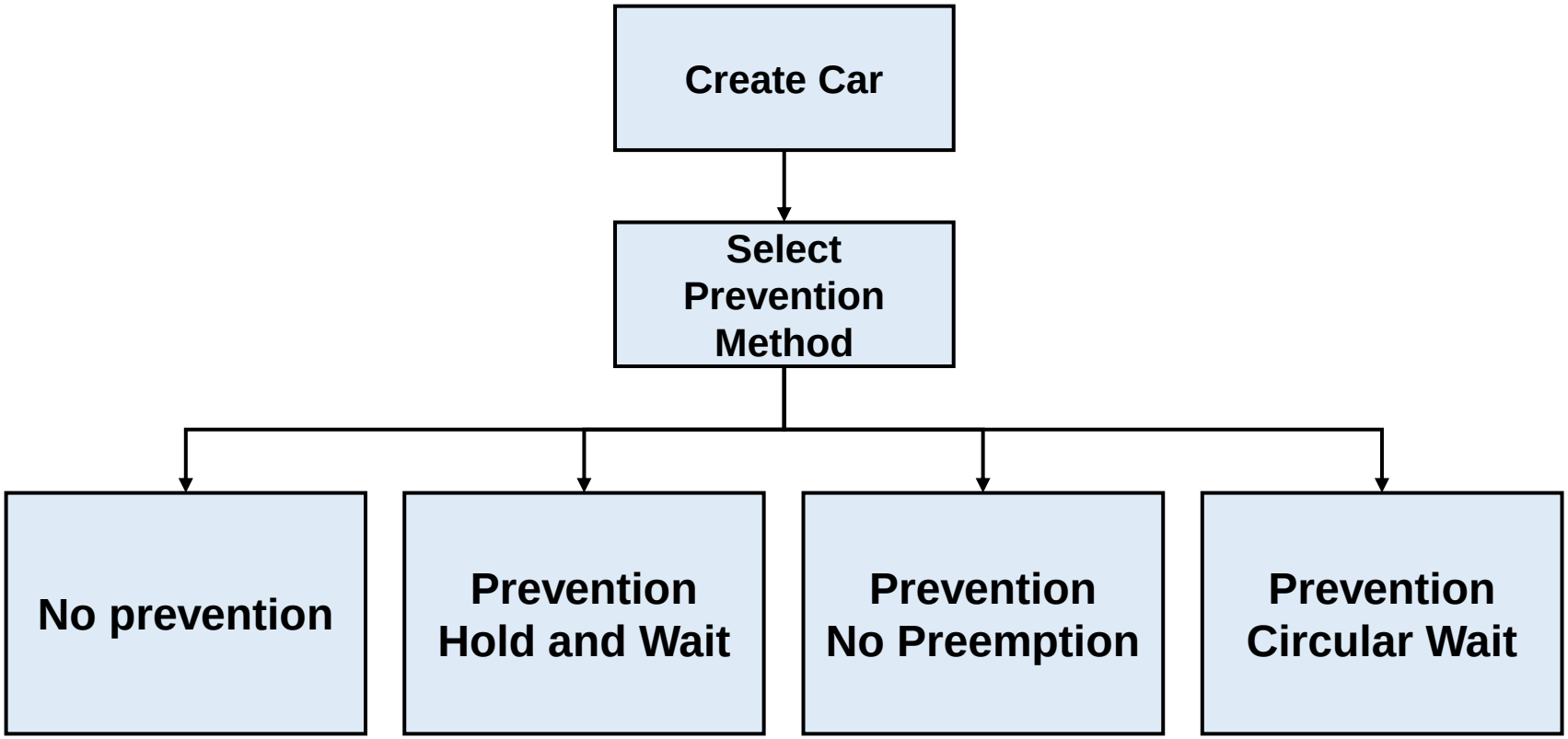
- 구현 내용 설명
  - void No_prevention(char** load, Car_info* car[])
    - 전체 도로와 자동차 정보를 매개변수로 받음
    - 교착 상태 예방 방법을 사용하지 않고 기본적인 자동차 이동 조건만 이용하여 자동차를 목적지까지 이동시킴
    - 이중포인터의 load와 자동차의 다음 위치를 확인하여, 이동 여부 결정
    - 모든 자동차가 각각의 목적지에 도착하면 프로그램 종료
    - 교착 상태가 발생할 경우, 시간만 증가하며 프로그램이 종료되지 않는 것이 정상
  - Void Prevention_Hold_and_Wait(char** load, Car_info* car[])
    - 전체 도로와 자동차 정보를 매개변수로 받음
    - 점유대기 제거 방법을 이용하여 모든 자동차를 목적지까지 이동시킴
    - No_prevention 함수의 내용을 바탕으로 점유 대기 예방 조건을 추가하여 자동차 이동
    - 이동하는 자동차는 도로를 점유한 후 이동
    - 모든 자동차가 각각의 목적지에 도착하면 프로그램 종료

프로젝트 설명

- 구현 내용 설명
  - void Prevention_No_Preemption(char** load, Car_info* car[])
    - 전체 도로와 자동차 정보를 매개변수로 받음
    - 비선점 제거 방법을 이용하여 모든 자동차를 목적지까지 이동시킴
    - No_prevention 함수의 내용을 바탕으로 비선점 예방 조건을 추가하여 자동차 이동
    - 이동하려는 위치에 다른 자동차가 있다면, 1칸 뒤로 이동
    - 모든 자동차가 각각의 목적지에 도착하면 프로그램 종료
  - void Prevention_Circular_Wait(char** load, Car_info* car[], int* std_num)
    - 전체 도로와 자동차 정보, 학번을 매개변수로 받음
    - 순환 대기 제거 방법 이용하여 모든 자동차를 목적지까지 이동시킴
    - No_prevention 함수의 내용을 바탕으로 순환 대기 예방 조건을 추가하여 자동차 이동
    - 교착 상태가 발생할 수 있는 교차로(4칸)에 대한 점유 방식에 교차로 접근 조건 추가
    - 모든 자동차가 각각의 목적지에 도착하면 프로그램 종료

프로젝트 설명

- 프로그램 전체 구조

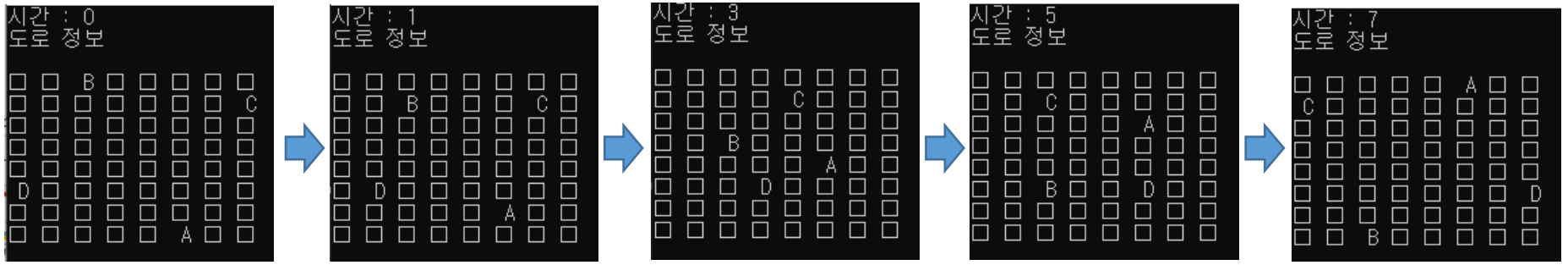


구현 결과 예시

- 자동차 기본 이동 조건(No Prevention) 예시(1)
  - 교착 상태(Deadlock) 발생 x

```
자동차(0) ID,도로 입력(문자 숫자) : A 5
자동차(1) ID,도로 입력(문자 숫자) : B 2
자동차(2) ID,도로 입력(문자 숫자) : C 1
자동차(3) ID,도로 입력(문자 숫자) : D 5

교착상태 예방 방법 선택.
1.No Prevention, 2.Hold and Wait, 3.No Preemption, 4.Circular Wait :1
```

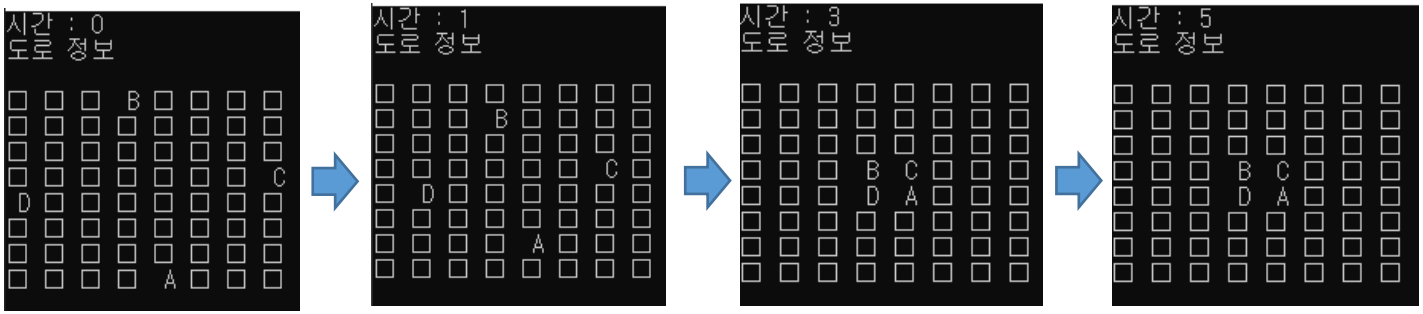


구현 결과 예시

- 자동차 기본 이동 조건(No Prevention) 예시(2)
  - 교착상태(Deadlock) 발생

```
자동차(0) ID,도로 입력(문자 숫자) : A 4
자동차(1) ID,도로 입력(문자 숫자) : B 3
자동차(2) ID,도로 입력(문자 숫자) : C 3
자동차(3) ID,도로 입력(문자 숫자) : D 4

교착상태 예방 방법 선택.
1.No Prevention, 2.Hold and Wait, 3.No Preemption, 4.Circular Wait :1
```

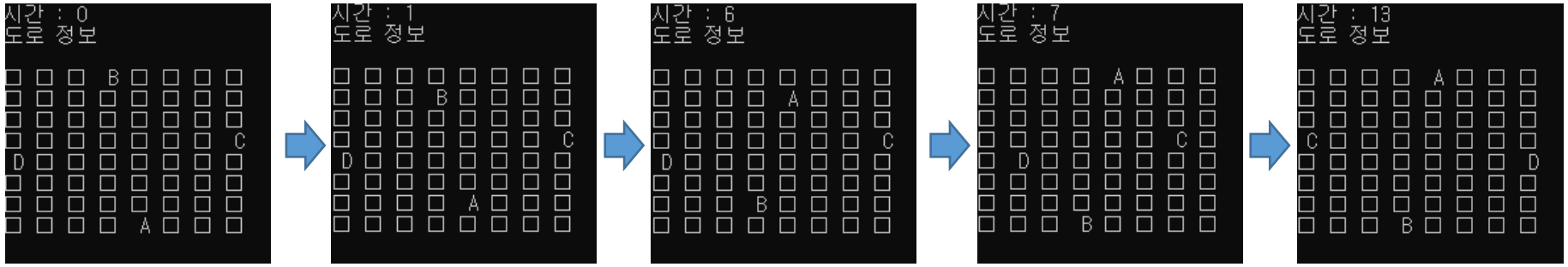


구현 결과 예시

- 점유대기(Hold and Wait) 제거 예시

```
자동차(0) ID,도로 입력(문자 숫자) : A 4
자동차(1) ID,도로 입력(문자 숫자) : B 3
자동차(2) ID,도로 입력(문자 숫자) : C 3
자동차(3) ID,도로 입력(문자 숫자) : D 4

교착상태 예방 방법 선택.
1.No Prevention, 2.Hold and Wait, 3.No Preemption, 4.Circular Wait :2
```

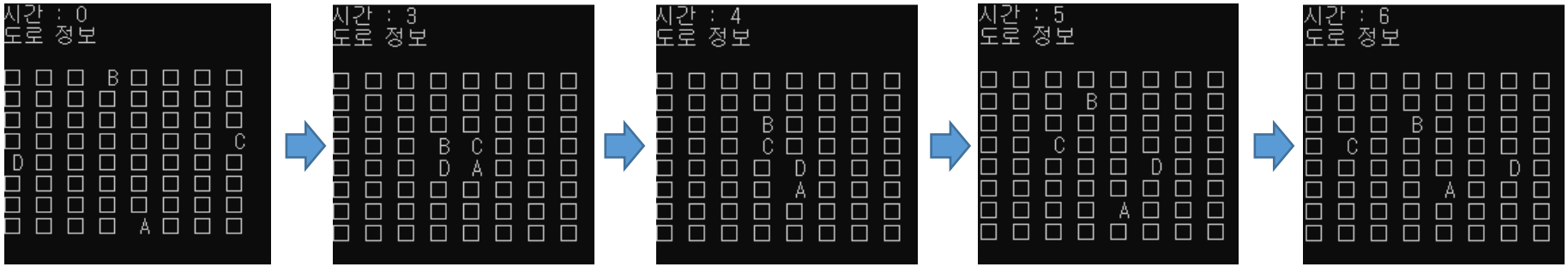


구현 결과 예시

- 비선점(No Preemption) 제거 예시

```
자동차(0) ID,도로 입력(문자 숫자) : A 4
자동차(1) ID,도로 입력(문자 숫자) : B 3
자동차(2) ID,도로 입력(문자 숫자) : C 3
자동차(3) ID,도로 입력(문자 숫자) : D 4

교착상태 예방 방법 선택.
1.No Prevention, 2.Hold and Wait, 3.No Preemption, 4.Circular Wait :3
```

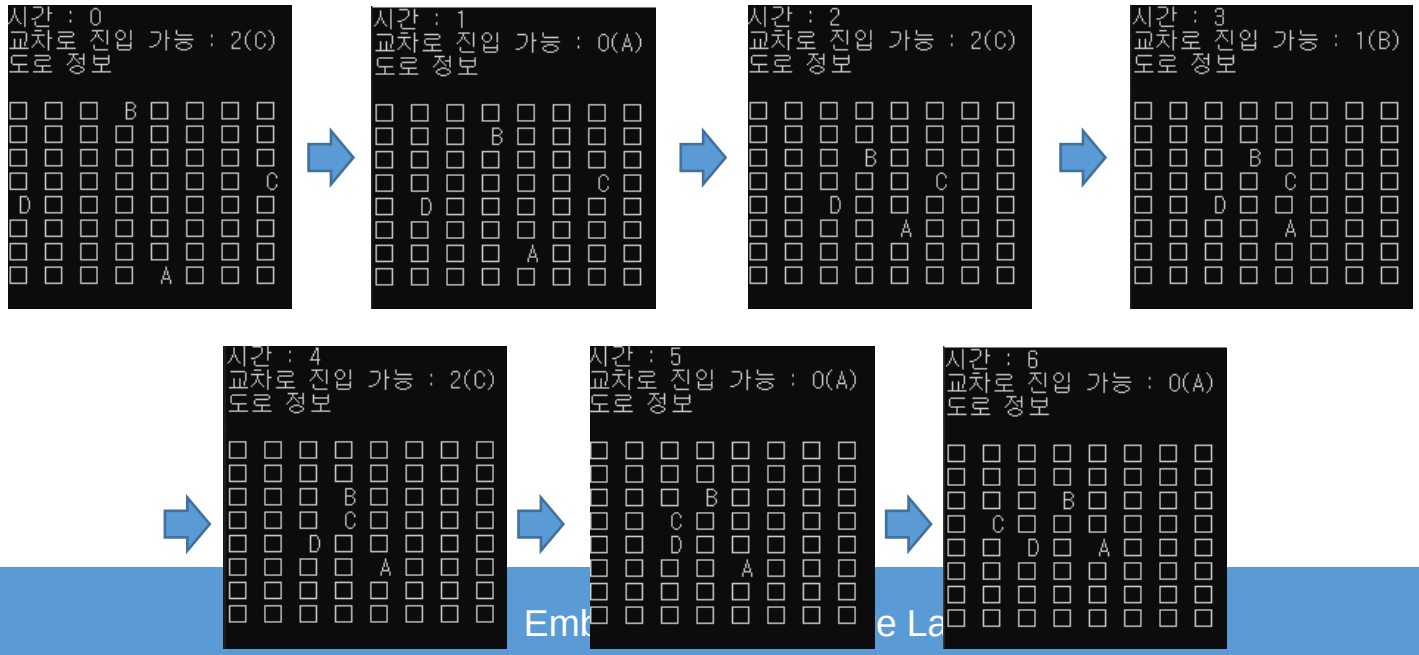


구현 결과 예시

- 순환대기(Circular Wait) 제거 예시(1)
  - 교차로 : 도로가 다른 도로와 만나는 지점
    - (3,3), (3,4), (4,3), (4,4)

```
자동차(0) ID,도로 입력(문자 숫자) : A 4
자동차(1) ID,도로 입력(문자 숫자) : B 3
자동차(2) ID,도로 입력(문자 숫자) : C 3
자동차(3) ID,도로 입력(문자 숫자) : D 4

교착상태 예방 방법 선택.
1.No Prevention, 2.Hold and Wait, 3.No Preemption, 4.Circular Wait :4
교차로 진입 순서(학번) 입력 : 202124425
```

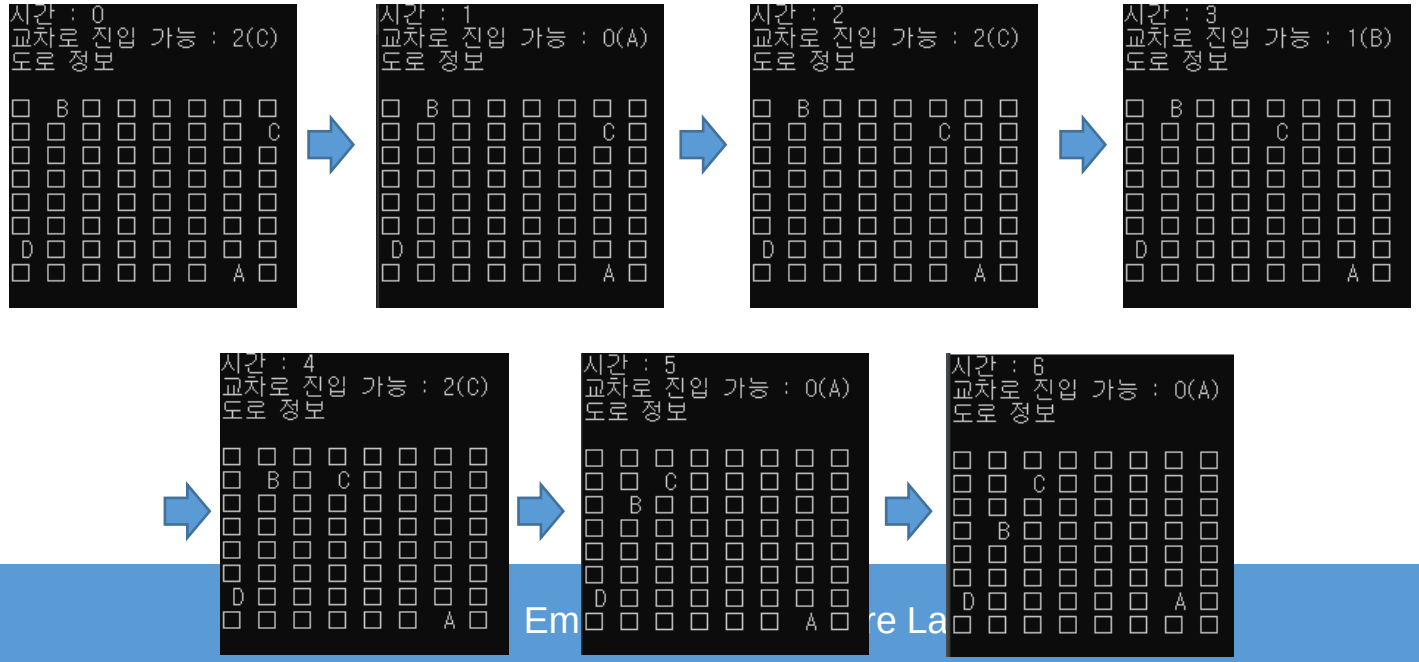


구현 결과 예시

- 순환대기(Circular Wait) 제거 예시(2)
  - 교차로 : 도로가 다른 도로와 만나는 지점
    - (1,1), (1,6), (6,1), (6,6)

```
자동차(0) ID,도로 입력(문자 숫자) : A 6
자동차(1) ID,도로 입력(문자 숫자) : B 1
자동차(2) ID,도로 입력(문자 숫자) : C 1
자동차(3) ID,도로 입력(문자 숫자) : D 6

교착상태 예방 방법 선택.
1.No Prevention, 2.Hold and Wait, 3.No Preemption, 4.Circular Wait :4
교차로 진입 순서(학번) 입력 : 202124425
```

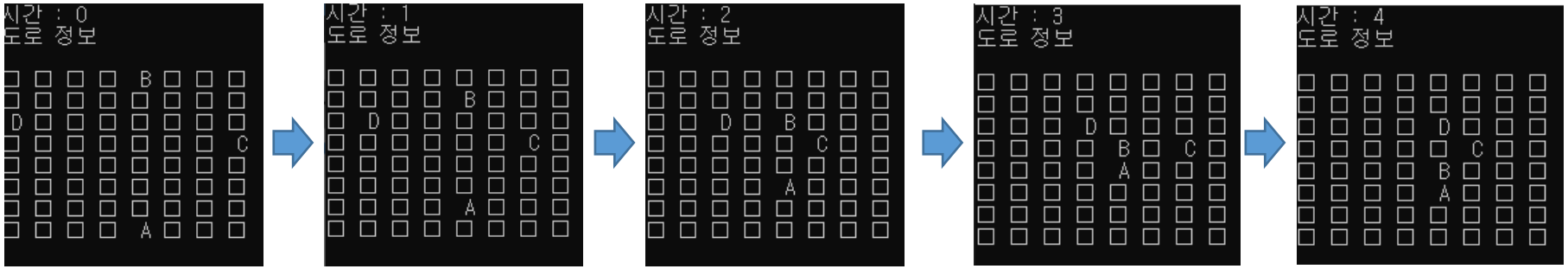


구현 결과 예시

- 같은 도로 사용 예시

```
자동차(0) ID,도로 입력(문자 숫자) : A 4
자동차(1) ID,도로 입력(문자 숫자) : B 4
자동차(2) ID,도로 입력(문자 숫자) : C 3
자동차(3) ID,도로 입력(문자 숫자) : D 2

교착상태 예방 방법 선택.
1.No Prevention, 2.Hold and Wait, 3.No Preemption, 4.Circular Wait :3
```



과제 채점 기준

- Due Date : 4/26(수) 자정 **(기한 엄수)**

감점	**지각**	(4/26에서 4/27로 넘어가는 **자정**까지 제출)	감점 x	블랙보드에 업로드 해주세요. (블랙보드가 문제되는 경우, 메일로 보내주세요.)
		~ 1일 지각	(총점수) * 0.75	총 점수에서 25% 감점
		~ 3일 지각	(총점수) * 0.50	총 점수에서 50% 감점
		3일 이상 지각	0	과제 마감 시점을 기준으로 3일 이후로는 받지 않습니다.
	코드 미제출	보고서 코드가 있을 때,	(보고서 점수) * 0.50	
		보고서에 코드가 없을 때,	구현점수 x	
	메인 함수 수정		**상관 없음**	
	cpp 파일 제출			c 파일로 확장자 변환 후 구현 채점기준 적용
	Copy		**0**	

과제 채점 기준

- 평가 기준
  - <구현(60%) + 보고서(40%)>

: 구현과 보고서의 평가는 기능별로 평가를 할 예정이며, 구현의 경우는 각 기능을 명시된 설명에 맞게 구현하였는지, 구현하지 못하였다면 자신이 구현한 부분에 대해 어느 정도 설명이 되어 있는지를 보고 평가할 예정입니다

- ✓ **필수** : **Visual Studio 2019**로 구현 권장
  - ✓ 해당 환경에서 작동하지 않거나 오류 발생시 **불이익** 있을 수 있음
- ✓ 각 함수에 대한 설명은 주석을 참조하여 구현
- ✓ 결과 화면에 수행해 본 예시에 대한 분석 필수(보고서)
- ✓ **보고서 작성 시 구현하지 못한 내용 작성**
- ✓ 프로젝트 제출 x, .cpp 파일 x
  - ✓ 프로젝트 제출 또는 .cpp 파일 제출의 경우 **감점** 예정
- ✓ **Copy시 0점 처리는 물론, 추가 불이익이 있음**
  - ✓ **표절 검사기 사용 예정**
  - ✓ 두 명의 조교가 line by line으로 채점할 예정

과제 채점 기준

- 평가 기준
  - 구현 부분(60점)
    - createCar, update_ load 함수 : 각 5점
    - No_prevention : 10점
    - Prevention_Hold_and_Wait : 10점
    - Prevention_No_Preemption : 10점
    - Prevention_Circular_Wait : 20점
    - 코드에 대한 간단한 주석
    - 코드 수정
      - **Main문 수정 금지**
      - Skeleton 코드 내 빈칸으로 작성된 **6가지 함수 구현 필수**
      - 변수, 함수 등 추가로 선언 가능
  - 구현 예외사항
    - 상하 방향 자동차 시작 위치와 좌우 방향 자동차 시작 위치가 겹치는 경우
      - E.g. 상 방향 도로 7번 + 좌 방향 도로 7번 → 시작위치 (7,7)
      - 해당 경우는 채점하지 않을 예정으로 구현(예외처리)하지 않아도 무방함

과제 채점 기준

- 평가 기준
  - 보고서 부분(40점)
    - 구현된 함수의 기능 명세 및 사용 방식 설명
    - 주어진 코드 분석, 작성한 코드 분석
    - 결과 화면에 대한 분석
      - **모든 보고서 작성은 본인이 직접 작성한 내용이어야 함** (강의노트 paste X)
  - 제출 방법
    - **.c 파일**과 **보고서 파일** 제출 (프로젝트 제출 x, .cpp 파일 x)
  - 합산 점수는 추후 과제 점수 비율에 따라 조정될 예정

Q&A

AJOU UNIVERSITY

Embedded & Software Lab.