**REPORT**

**| Embedded System**

**| Mailbox and Message Queue**

**| 컴퓨터정보공학과**

**| 12141540 박영창**

# 내용

4주차 실습 강의 시간에는 Mailbox, Message Queue에 관해 배웠습니다.

Mailbox와 Message Queue는 여러 개의 task를 서로 연동시켜 동작시키는 동기화(synchronization) 중 각 task에서 발생하거나 필요로 하는 값을 전달하는 내부통신(Intertask communication)을 위해 사용하는 개념들이다.

* Mail Box
  + 이름 그대로 우체통 같은 역할을 하며 각 task가 정보를 주고 받을 때, mailbox에 정보를 넣고, mailbox로부터 정보를 받아온다.
  + 선언
    - OS\_EVENT \* Mbox\_name[TASKS]

<설명>

OS\_EVENT \* 형으로 필요한 Task 개수만큼 선언

<인자>

Mbox\_name : 사용할 mailbox 이름

TASKS : 필요한 task 개수

* + 초기화
    - OS\_EVENT \* OSMboxCreate(void \*msg)

<설명>

Mailbox를 만든다/초기화한다

<인자>

msg : Mailbox의 초기값

* + Wait
    - Void\* OSMboxPend(OS\_EVENT \* ptr, INT16U timeout, INT8U \*err)

<설명>

Mailbox ptr로부터 정보를 받는다

<인자>

ptr : OSMboxCreate()에 의해 만들어진 Mailbox 정보

timeout : 대기시간

err : 에러 코드

* + Signal
    - INT8U OSMboxPost(OS\_EVENT \* ptr, void \* msg)

<설명>

Mailbox ptr에 msg를 전달한다

<인자>

ptr : OSMboxCreate()에 의해 만들어진 Mailbox 정보

msg : 전달하려는 정보

* Message Queue
  + Mail box와 마찬가지로 각 task가 정보를 주고 받을 때, message queue에 정보를 넣고 message queue로부터 정보를 받아온다.
  + 선언
    - OS\_EVENT \* MsgQueue\_name

Void \*MsgQueue\_name\_table[TASKS]

<설명>

OS\_EVENT \* 형으로 배열이 아닌 변수로 선언 후 Task 개수만큼 Table을 선언

<인자>

MsgQueue\_name : message queue 이름

MsgQueue\_name\_table : queue 크기 지정을 위한 table

TASKS : task 개수

* + 초기화
    - OS\_EVENT \* OSQCreate(void \*start, INT16U size)

<설명>

Message queue를 만든다

<인자>

start : 데이터를 저장할 공간의 시작주소

size : 저장할 수 있는 데이터의 개수

* + Wait
    - Void\* OSQPend(OS\_EVENT \* ptr, INT16U timeout, INT8U \* err)

<설명>

Queue ptr로부터 정보를 받아온다

<인자>

ptr : OSQCreate()에 의해 만들어진 queue 정보

timeout : 대기시간

err : 에러 코드

* + Signal
    - INT8U OSQPost(OS\_EVENT \* ptr, void \* msg)

<설명>

Queue ptr에 msg를 전달한다

<인자>

ptr : OSQCreate()에 의해 만들어진 queue 정보

msg : 전달하려는 정보

* 차이점
  + Mailbox는 한 번에 하나의 데이터 밖에 저장되지 않아 연속적으로 같은 Mailbox에 여러 task에서 signal을 보내면 mailbox의 용량이 넘쳐 에러가 발생할 수 있다. 반면에 Message Queue는 Mailbox와 동작 방식은 동일하지만 용량이 더 크다. 그래도 Queue 생성 시 용량을 지정하기 때문에 무한대로 저장할 수 있는 것은 아니다.

# 과제 목표

* 5개의 태스크 생성 ( 1 : decision , 4 : random )
* 4 개의 Random task에서 0에서 63까지 임의의 숫자를 1개씩 생성한다
* 4 개의 Random task는 Mailbox/Message Queue를 이용해 생성된 숫자를 Decision task로 보낸다.
* Decision Task는 4 개의 task로부터 4 개의 숫자를 받아 가장 작은 숫자를 보낸 task에게 ‘W’라는 문자를 보내고, 나머지 task들에게는 ‘L’이라는 문자를 보낸다. ( 숫자가 같을 경우는 높은 우선순위 task에게 보낸다. )
* ‘W’를 받은 task는 우선순위에 맞는 색깔로 화면을 칠한다

1 – Red 2 – Blue 3 – Green 4 – Brown

* 모든 task들은 4초간 쉬고 난 후에 위 과정을 반복한다.

# 문제 해결

**변수**

* **Mail Box 선언**
  + OS\_EVENT \*Mbox\_RandomNumbers[RANDOM\_TASKS]

생성된 난수를 보내고 받는 mailbox

* + OS\_EVENT \*Mbox\_Result[RANDOM\_TASKS]

결과값을 통해 결정된 ‘W’, ‘L’을 보내고 받는 mailbox

* **Message Queue 선언**
  + OS\_EVENT \*MsgQueue\_RandomNumbers

생성된 난수를 보내고 받는 queue

* + OS\_EVENT \*MsgQueue\_Result

결과값을 통해 결정된 ‘W’, ‘L’을 보내고 받는 queue

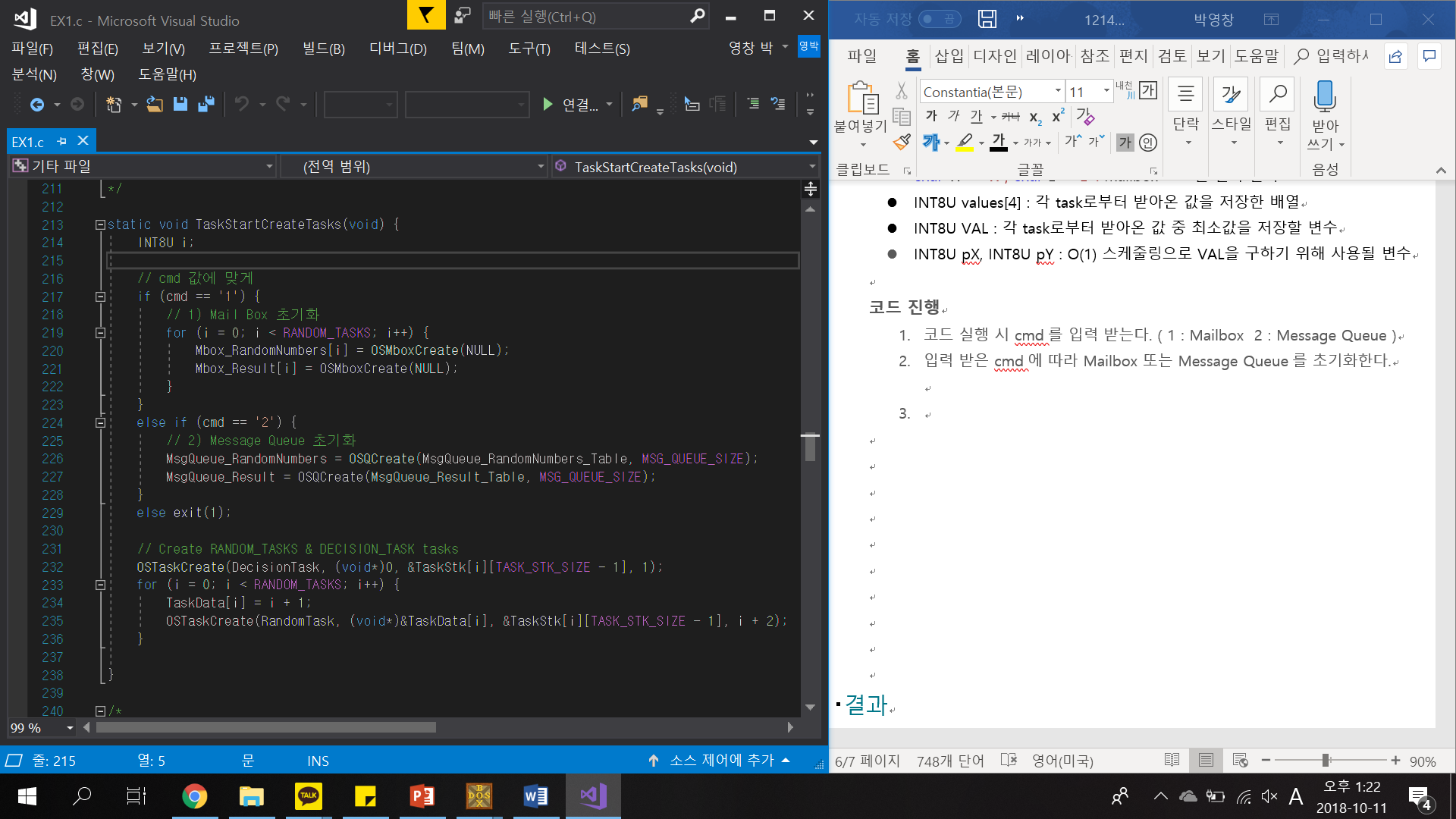
* + void \*MsgQueue\_RandomNumbers\_Table[MSG\_QUEUE\_SIZE]
  + void \*MsgQueue\_Result\_Table[MSG\_QUEUE\_SIZE]

queue size 지정

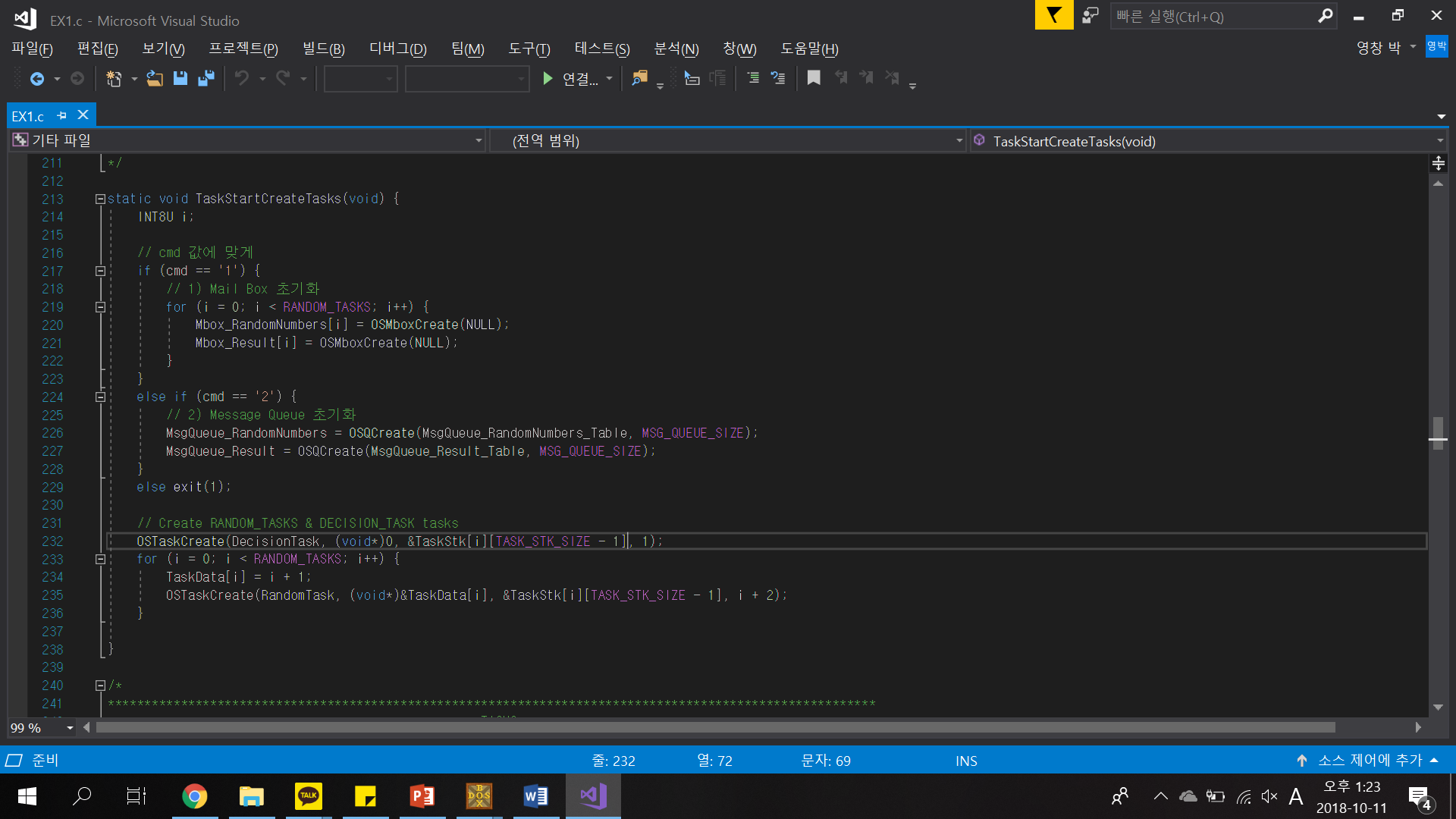
* **상수 define** 
  + #define RANDOM\_TASKS 4 : Random Task의 개수
  + #define DECISION\_TASK 1 : Decision task의 개수
  + #define MSG\_QUEUE\_SIZE 4 : Queue size
* INT8U const colors[4] : 우선순위에 맞는 색깔을 저장한 배열
* **스케줄링을 위한 변수**
  + INT8U const myMapTbl[]
  + INT8U const myUnMapTbl[]
  + INT8U myRdyTbl[8];
  + INT8U myRdyGrp;
* **Random Task**
  + INT8U err; : 에러 코드
  + INT8U number; : 난수 생성 후 저장
  + char \*result; : ‘W’ 또는 ‘L’의 결과를 저장
  + char \*data; : task 우선순위 정보
  + INT8U dispX; : 출력할 값의 x좌표
  + INT8U dispY; : 출력할 값의 y좌표
* **Decision Task**
  + INT8U err : 에러 코드
  + INT8U i : 반복문에 사용될 변수
  + INT8U idx : mailbox 배열에서 최소값을 갖는 가장 낮은 우선순위 인덱스
  + INT8U chk : message queue에서 최소값이 나왔는지 확인하는 변수
  + char \*tmp : mailbox로부터 값을 char\* 형으로 받아오는 변수
  + char W = 'W', char L = 'L' : mailbox로 보낼 결과 문자
  + INT8U values[4] : 각 task로부터 받아온 값을 저장한 배열
  + INT8U VAL : 각 task로부터 받아온 값 중 최소값을 저장할 변수
  + INT8U pX, INT8U pY : O(1) 스케줄링으로 VAL을 구하기 위해 사용될 변수

**코드 진행**

1. 코드 실행 시 cmd를 입력 받는다. ( 1 : Mailbox 2 : Message Queue )
2. 입력 받은 cmd에 따라 Mailbox 또는 Message Queue를 초기화한다.

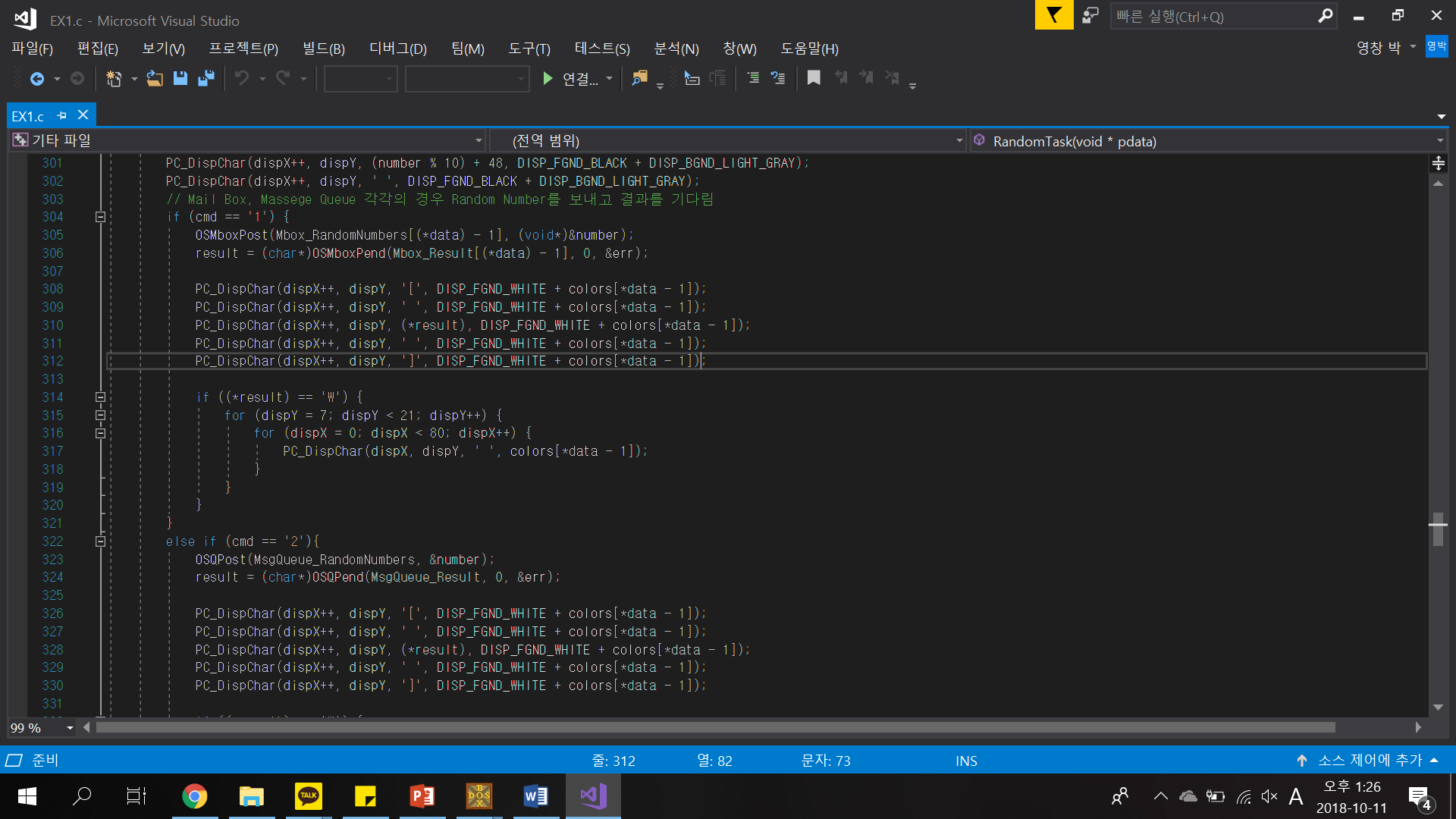


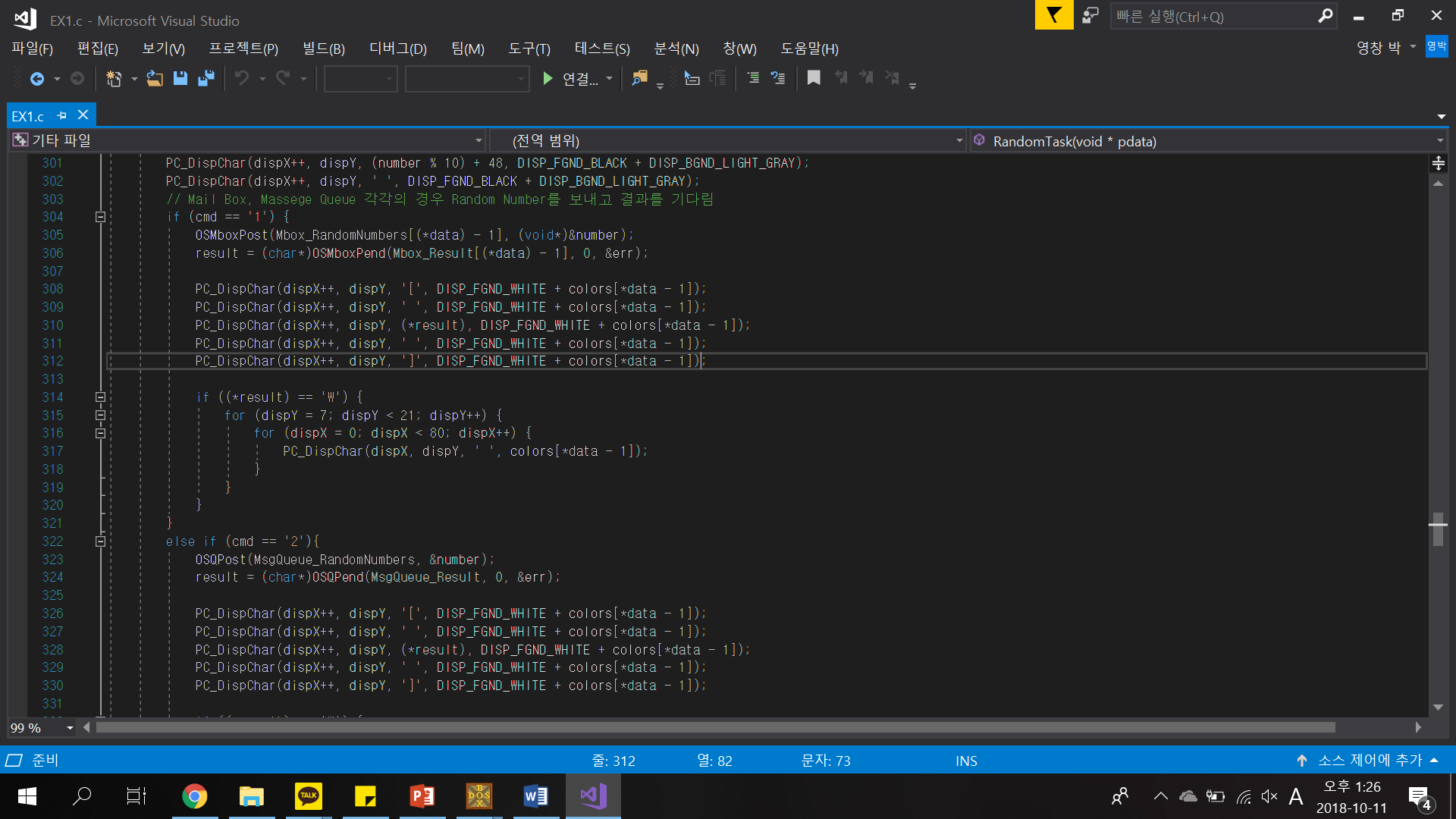
1. 1개의 Decision Task와 4개의 Random Task를 생성한다.



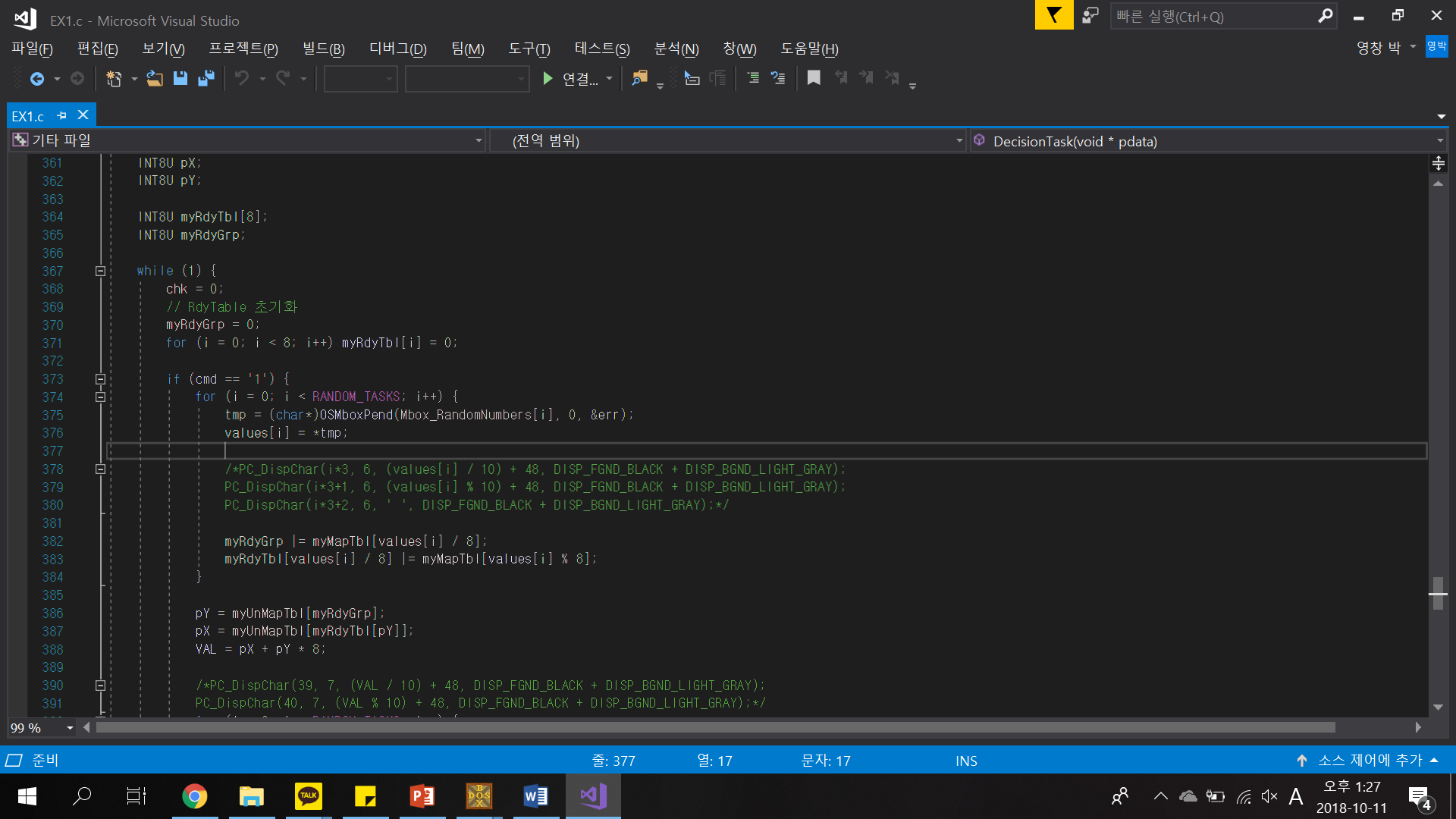
1. 각 Random Task는 난수를 생성해 mailbox 또는 message queue에 보내고, Decision Task는 이를 기다린 뒤 값을 전달받아 저장한다.

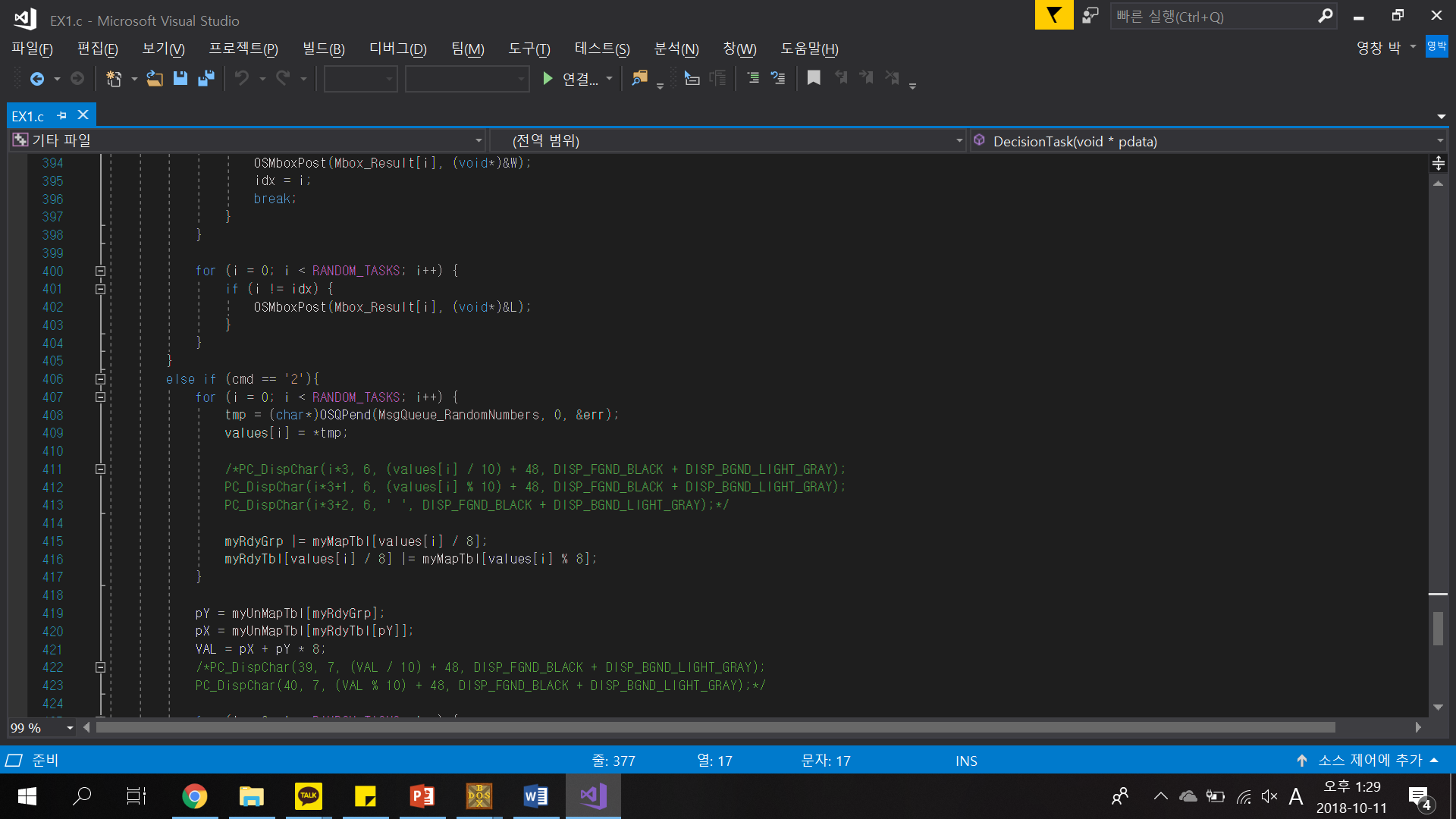
<Random Task>



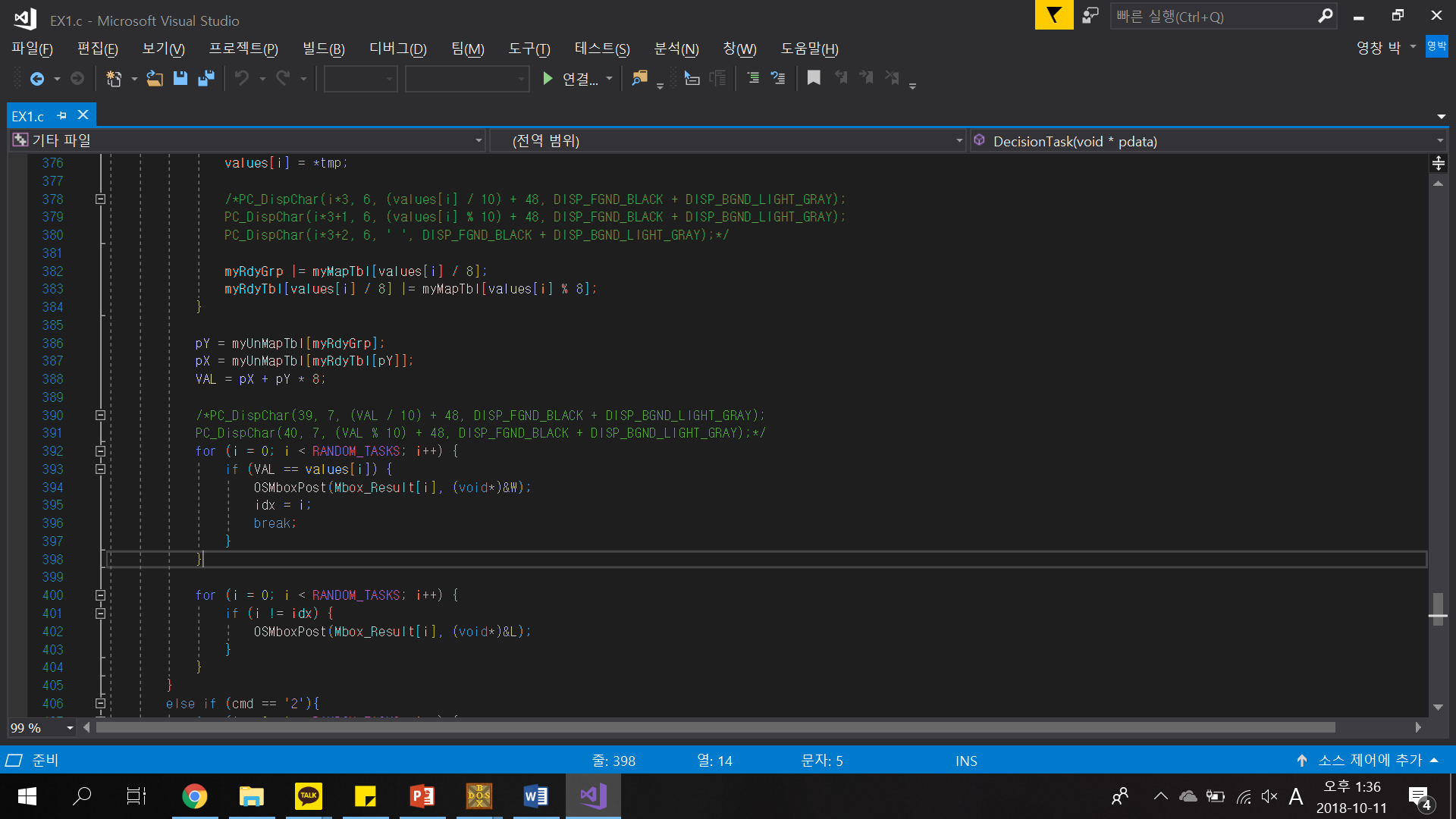


<Decision Task>

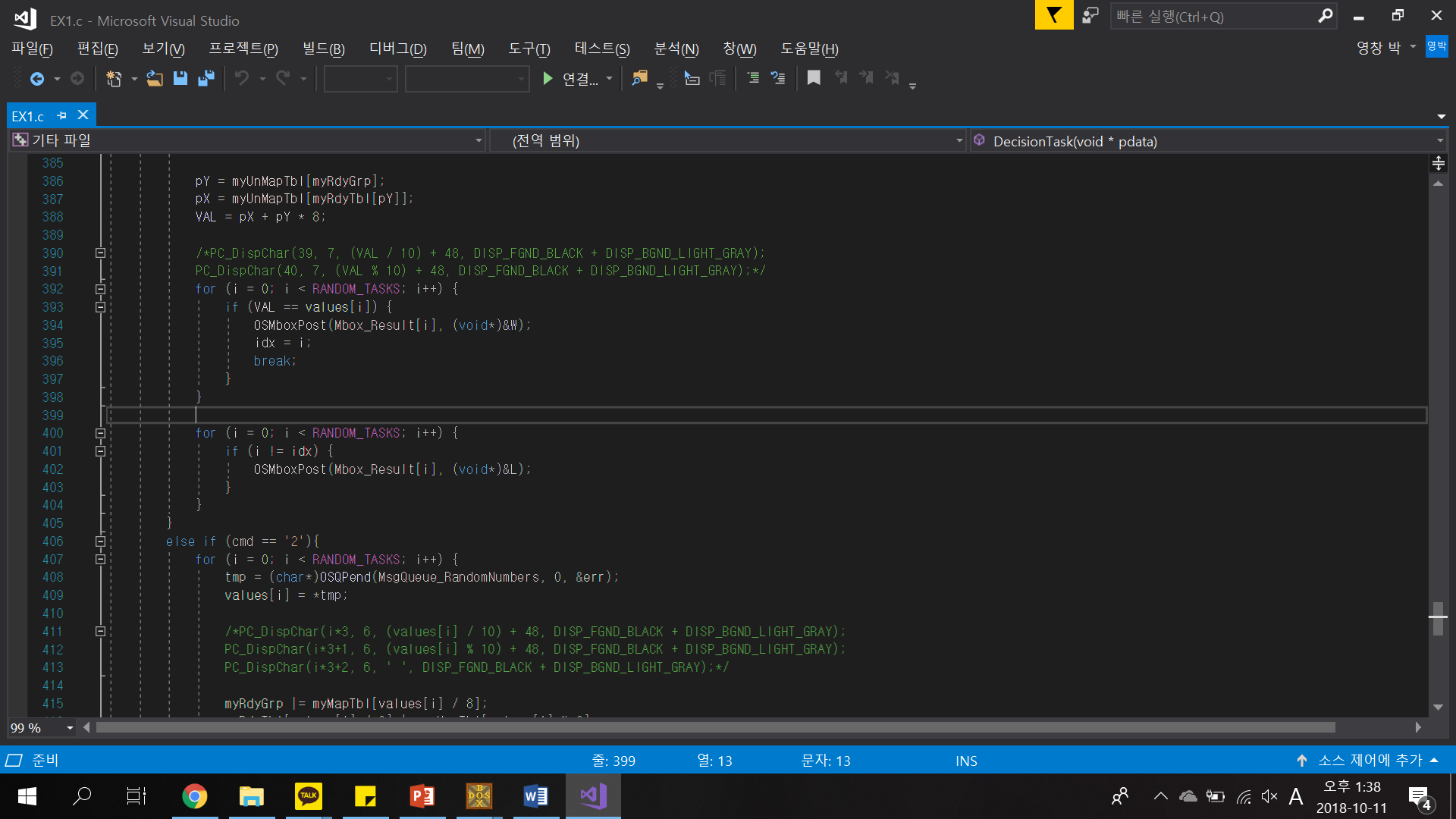
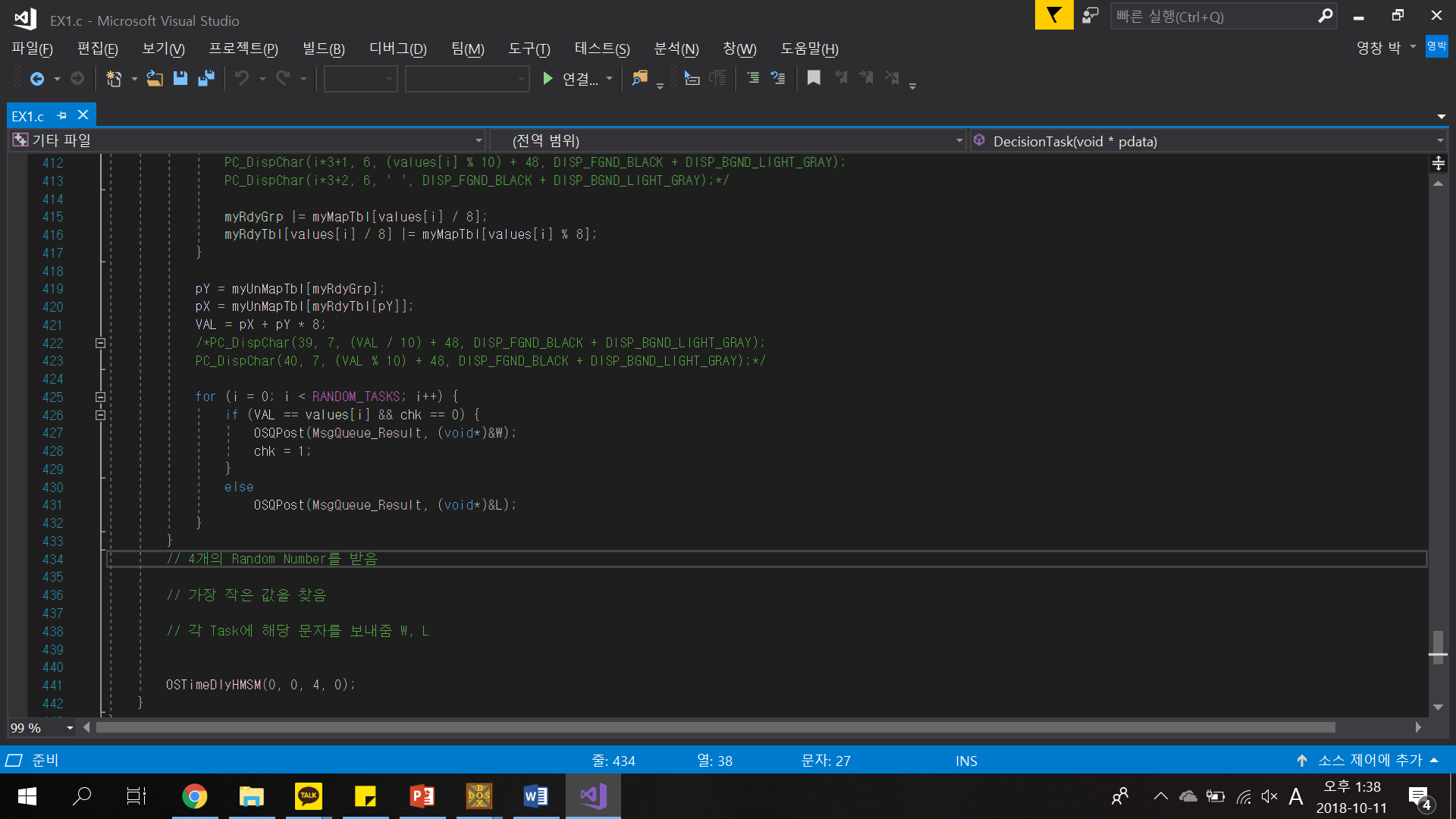




1. Decision Task에서 O(1) 스케줄러를 이용해 최소값을 구한다.



1. 각 task로부터 받은 값과 최소값을 비교해 같은 값을 가지는 가장 적은 인덱스(가장 높은 우선순위)에게만 W를, 나머지에게는 L를 보낸다.

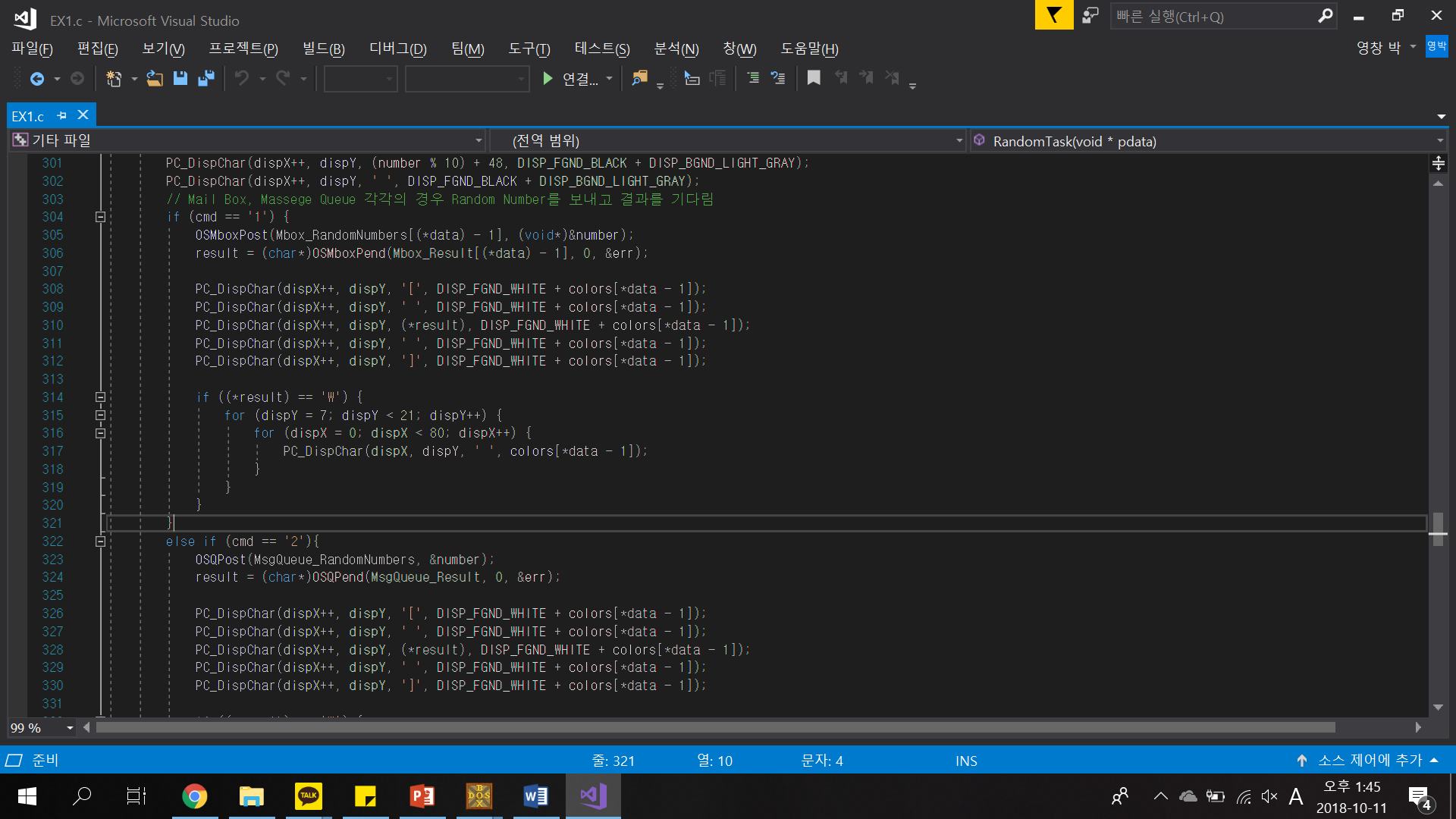
 

< Mailbox > < Message Queue >

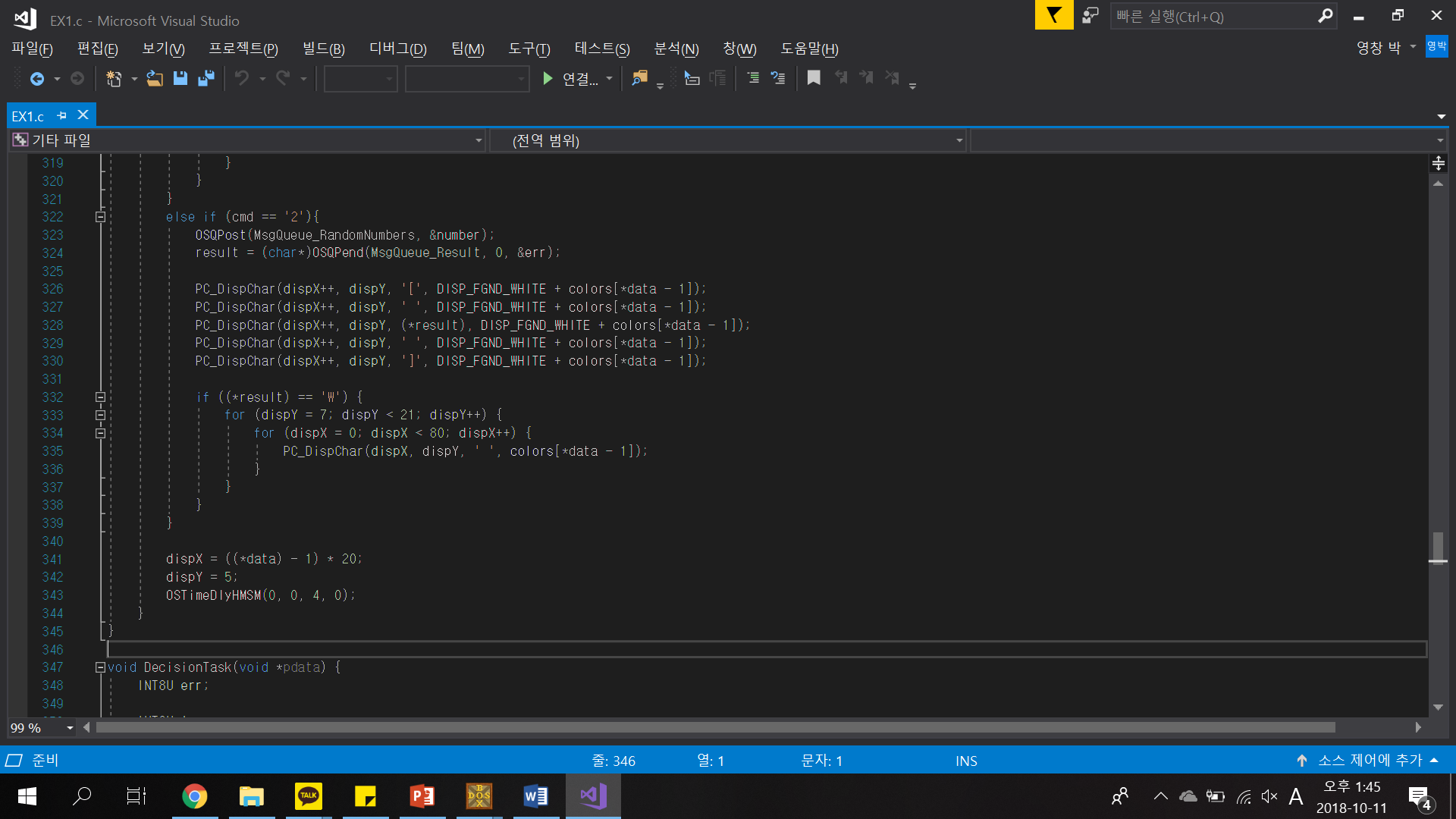
가장 빠른 최소값을 갖는 인덱스 idx 저장 최소값을 발견하면 W를 보냄과 동시에

후 W를 보내고, idx를 제외한 곳에 L을 보냄 chk 를 1로 바꿔 순서대로 확인하면서 보냄

1. Random Task에서 W 또는 L을 받아와서 만약 W라면 해당 Task Number에 맞는 색으로 배경을 칠한다.



< Mailbox >



< Message Queue >

1. 위 과정을 4초에 한번씩 반복한다.

# 결과

< Mailbox > < Message Queue >

