

제 3장 제어 메시지 처리하기

2018년 1학기 윈도우 프로그래밍

학습 목표

- 학습 목표

- 자동으로 움직이는 형상을 타이머를 이용해 윈도우에 표현할 수 있다.
- 마우스에서 발생한 메시지를 이용할 수 있다.
- 래스터 연산 방법에 관하여 학습한다.

- 내용

- 타이머 메시지 처리하기
- 마우스 메시지 처리하기

1. 키보드 입력에 따른 도형 이동

- 오른쪽 화살표 누른 동안 색상 변경하고 오른쪽 경계 안에서 원 이동하기

```
static bool flag;           // 방향 화살표키가 눌려졌는지 체크하기 위한 변수
switch (iMsg)
{
case WM_CREATE:
    GetClientRect (hwnd, &rectView);
    x = 20; y = 20;
    break;
case WM_PAINT:
    hdc = BeginPaint(hwnd, &ps);           // 키를 눌렀으면 회색 브러쉬 등록
    if (flag)
        SelectObject (hdc, GetStockObject(LTGRAY_BRUSH));
    Ellipse (hdc, x-20, y-20, x+20, y+20);
    EndPaint (hwnd, &ps);
    break;
case WM_KEYUP:
    flag = false;                         // 키에서 손을 떼면 false
    InvalidateRgn (hwnd, NULL, TRUE);
    break;
case WM_KEYDOWN:
    if (wParam == VK_RIGHT)
    {
        flag = true;                     // 오른쪽 화살표 키를 누르면 true
        x += 40;
        if(x + 20 > rectView.right)      x -= 40;
    }
    InvalidateRgn (hwnd, NULL, TRUE);
    break;
}
```

윈도우의 크기를 측정

- 클라이언트 영역의 크기 측정

```
BOOL GetClientRect(  
    HWND hwnd,  
    LPRECT lpRect  
);
```

- hwnd: 측정하기 원하는 윈도우의 핸들
- lpRect: RECT 구조의 공간의 주소

- WM_SIZE: 윈도우의 크기가 변경하면 발생하는 메시지

- lParam에 윈도우 높이와 폭 저장
 - HIWORD(lParam): 윈도우의 높이
 - LOWORD(lParam): 윈도우의 폭

- HIWORD(): 32bit 데이터에서 상위 16bit 데이터를 구하기 위한 매크로 함수
- LOWORD(): 32bit 데이터에서 하위 16bit 데이터를 구하기 위한 매크로 함수

2. 타이머 메시지

- 매 10초마다 알람을 하고 싶다면 어떻게 할까?
 - 10초마다 특정메시지를 발생시키고 그 메시지 처리부에서 알람 기능을 구현하면 된다.
 - 발생 메시지: WM_TIMER
- SetTimer() 함수로 타이머를 설치했을 경우 지정한 시간 간격으로 WM_TIMER 메시지가 반복적으로 큐에 넣어진다.
- 다수의 타이머가 설치되어 있을 경우
 - 각각의 타이머는 정해진 시간 간격으로 이 메시지를 큐에 저장하며
 - WM_TIMER에서는 wParam값으로 어떤 타이머에 의해 이 메시지가 발생했는지 조사한다.

타이머 메시지

- WM_TIMER 메시지는 다른 메시지들에 비해 우선순위가 낮게 설정
 - 먼저 처리해야 할 메시지가 있을 경우 곧바로 윈도우 프로시저로 보내지지 않을 수 있다.
 - 따라서 정확한 시간에 이 메시지가 전달되지 않는 경우도 있으므로 정확도를 요하는 작업에는 이 메시지를 사용하지 않는 것이 좋다.
 - 정확도를 요하는 작업에는 타이머 콜백 함수를 지정한다. 타이머 콜백 함수를 지정했을 경우는 이 메시지부를 수행하는 것이 아니라, 프로그래머가 만든 함수(타이머 콜백 함수)를 OS가 자동으로 주기적으로 호출해 준다.
- WM_TIMER 메시지에서
 - wParam: 타이머의 ID가 전달된다. 이 ID는 SetTimer()함수의 두번째 인자로 지정한 값으로 여러 개의 타이머를 구분하기 위한 것
 - lParam: 타이머 콜백 함수를 사용할 경우 콜백 함수명

타이머 메시지

- 타이머 설정함수

- 타이머를 설정하는 함수

WORD SetTimer (HWND hWnd, UINT_PTR nIDEvent, UINT uElapse, TIMERPROC lpTimerFunc);

- hWnd : 윈도우 핸들
- nIDEvent : 타이머 ID, 여러 개의 타이머를 구분하기 위한 정수
- uElapse : 시간간격 milisec 단위(1000분의 1초)
- lpTimerFunc : 시간간격 마다 수행할 함수
 - NULL이라고 쓰면 WndProc() 함수가 타이머메시지를 처리)

- 타이머 콜백 함수

- SetTimer 함수의 마지막 인자로 설정되는 타이머 콜백 함수

void CALLBACK TimerProc (HWND hwnd, UINT uMsg, UINT_PTR idEvent, DWORD dwTime);

- hWnd: 타이머를 소유한 윈도우 핸들
- uMsg: WM_TIMER 메시지
- idEvent: 타이머 id
- dwTime: 윈도우가 실행된 후의 경과시간

타이머 메시지

- WM_TIMER 메시지 처리방법

LRESULT CALLBACK **wndProc** (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)

```
{
    static int Timer1Count=0, Timer2Count=0;

    switch (iMsg) // 메시지 번호
    {
        case WM_CREATE:
            SetTimer (hwnd, 1, 70, NULL);           // 1번 아이디를 가진 타이머: 0.07초 간격
            SetTimer (hwnd, 2, 100, NULL);          // 2번 아이디를 가진 타이머: 0.1초 간격
            break;

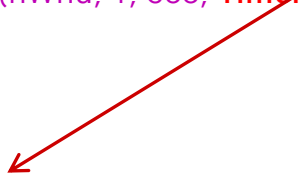
        case WM_TIMER:
            switch (wParam) {
                case 1:                               // 1번 아이디 타이머: 0.07초 간격으로 실행
                    Timer1Count++;
                    break;
                case 2:                               // 2번 아이디 타이머: 0.1초 간격으로 실행
                    Timer2Count++;
                    break;
            }
            break;
    }

    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```


타이머 메시지

- 타이머 콜백 함수 이용 방법

```
LRESULT CALLBACK wndProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    switch ( uMsg ){
        case WM_CREATE:
            SetTimer (hWnd, 1, 500, TimerProc); // 1번 아이디의 타이머가 0.5초 마다 TimerProc 타이머 함수 실행
            break;
    }
    return 0;
}
```



```
void CALLBACK TimerProc (HWND hWnd, UINT uMsg, UINT idEvent, DWORD dwTime ) //1번 아이디 타이머 함수
{
    HDC hdc;
    hdc = GetDC(hWnd);
    GetClientRect(hWnd, &rect);

    MyBrush = CreateSolidBrush(RGB(rand()%255, rand()%255, rand()%255) );
    MyPen = CreatePen(PS_SOLID, rand()%5, RGB(rand()%255, rand()%255, rand()%255) );
    OldBrush = (HBRUSH)SelectObject(hdc, MyBrush);
    OldPen = (HPEN)SelectObject(hdc, MyPen);

    Ellipse(hdc, rand()%(rect.right), rand()%(rect.bottom), rand()%(rect.right), rand()%(rect.bottom) );

    SelectObject(hdc, OldBrush);
    SelectObject(hdc, OldPen);
    DeleteObject(MyBrush);
    DeleteObject(MyPen);

    ReleaseDC(hWnd, hdc);
}
```

원 자동으로 이동하기

LRESULT CALLBACK **wndProc** (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)

```
{
    static int x=0;

    switch (iMsg)                                // 메시지 번호
    {
        case WM_CREATE:
            GetClientRect (hwnd, &rectView);
            x = 20;    y = 20;
            break;
        case WM_PAINT:
            hdc = BeginPaint (hwnd, &ps);
            Ellipse (hdc, x-20, y-20, x+20, y+20);
            EndPaint (hwnd, &ps);
            break;
        case WM_KEYDOWN:
            if (wParam == VK_RIGHT)              // 오른쪽 키를 누를 때
                SetTimer (hwnd, 1, 70, NULL);    // 타이머 설정
            break;
        case WM_TIMER:                          // 시간이 경과하면 메시지 자동 생성
            x += 40;
            if (x + 20 > rectView.right)
                x -= 40;
            InvalidateRect (hwnd, NULL, TRUE);
            break;
        case WM_DESTROY :
            KillTimer (hwnd, 1);                 // 윈도우 종료 시 타이머도 종료
            PostQuitMessage (0) ;
            break ;
    }
    return DefWindowProc (hwnd, iMsg, wParam, lParam);
}
```

실습 3-1

- 제목

- 움직이는 도형에 꼬리 달기

- 내용

- 화면에 40x40의 보드가 그려진다.
- 화면의 가운데 주인공원이 있고, 특정 방향으로 자동 이동하고 있다.
 - 보드의 칸에 맞춰 이동한다.
 - 키보드 명령어에 따라 좌우상하로 방향을 바꿀 수 있다.
- 보드의 네 코너에서 특정 시간마다 꼬리원들이 나타나서 임의의 방향으로 이동, 또는 근처를 배회한다.
 - 보드의 가장자리에 도착하면 임의의 방향으로 바뀌서 이동한다.
- 주인공원을 이동하여 꼬리원들과 만나면 꼬리원들은 주인공 원의 뒤에 꼬리로 붙는다.
- 키보드 명령어:
 - 속도: '+'를 입력하면 주인공원의 속도가 점점 빨라지고, '-'를 입력하면 속도가 점점 느려진다.
 - 점프: 특정 키를 누르면 주인공원과 그 꼬리들은 그 자리에서 이동방향에 수직방향으로 점프하도록 한다.
 - 변형: 애벌레 머리 도형이 커졌다 작아졌다를 반복한다
 - 종료: q/Q를 입력하면 프로그램이 종료한다.

실습 3-2

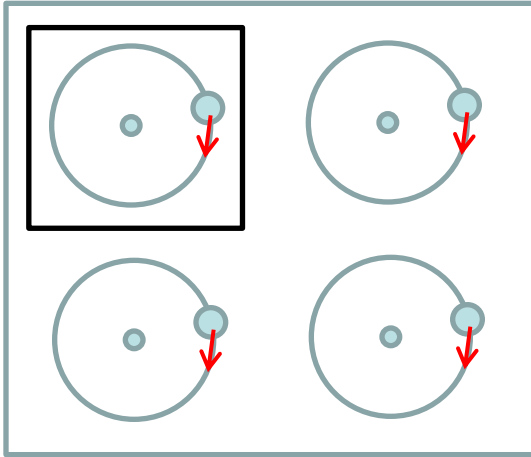
- 제목

- 궤도를 따라 공전하는 도형

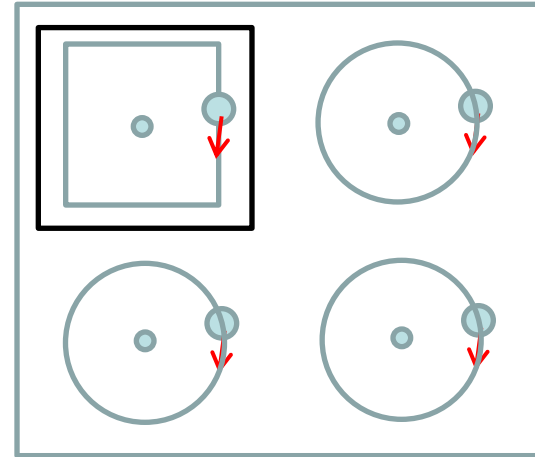
- 내용

- 화면을 사등분 하여 각 등분의 중앙에 빨간색 의 작은 원을 그린다.
 - 원의 색이 시간에 따라 랜덤하게 바뀐다.
- 각각의 빨간색 원을 중심으로 그 주위를 임의의 반지름을 가진 원의 궤도를 그리고 그 궤도를 따라 다양한 크기와 색의 원이 다양한 타이머에 따라 회전 한다. 회전 방향은 시계방향과 반시계방향이 모두 있다. 기본으로 궤도를 따라 도형은 회전하고 있다.
- 다음의 명령어를 실행한다.
 - 1/2/3/4: 각 등분 중 한 개를 선택하게 한다. 선택된 등분에 테두리를 그린다. 아래의 명령어는 선택된 등분의 궤도와 공전 도형에게 적용된다.
 - **c/C**: 방향을 거꾸로 이동한다 (시계 → 반시계, 반시계 → 시계)
 - **m/M**: 이동하는 회전 도형이 바뀌어서 이동한다. (원 → 사각형, 사각형 → 원)
 - **r/R**: 배경 도형이 사각형이 된다.
 - **t/T**: 배경 도형이 삼각형이 된다.
 - **q**: 프로그램이 종료
- 원의 좌표값 구하기
 - `sin()`, `cos()`함수 사용 (`#include <math.h>`)
 - 위의 `sin`, `cos` 함수는 라디언 값을 인자로 받음

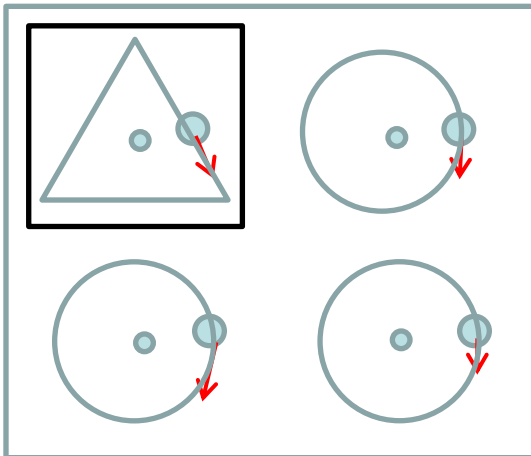
실습 3-2



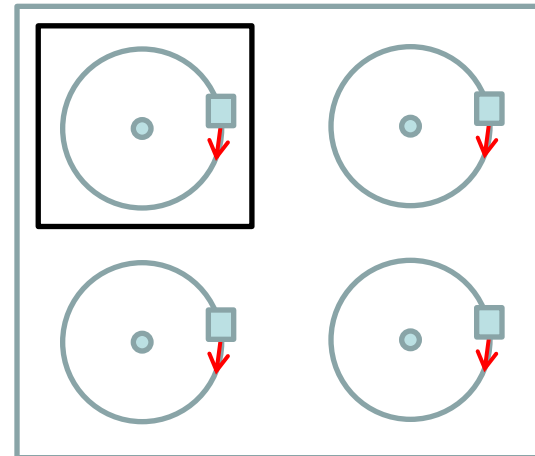
배경도형: 원, 회전도형: 원



배경도형: 사각형, 회전도형: 원



배경도형: 삼각형, 회전도형: 원



배경도형: 원, 회전도형: 사각형

3. 마우스 메시지

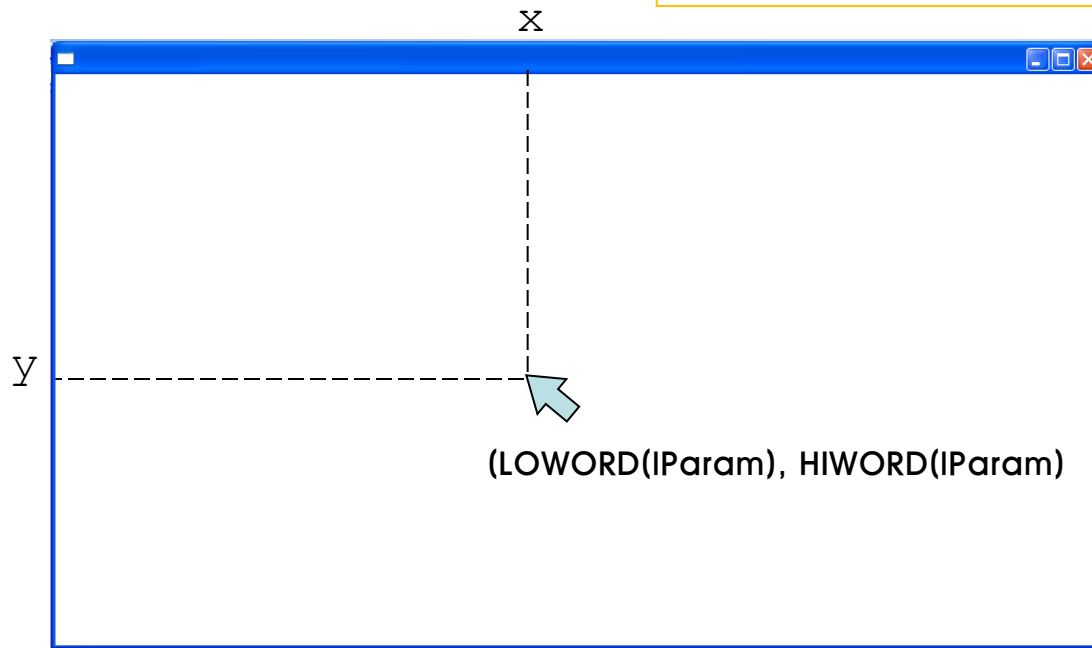
- 좌우에 버튼이 2개 있는 마우스의 이벤트
 - WM_LBUTTONDOWN
 - 왼쪽 마우스 버튼을 눌렀을 때 발생하는 메시지
 - WM_LBUTTONUP
 - 왼쪽 마우스 버튼을 떼었을 때 발생하는 메시지
 - WM_RBUTTONDOWN
 - 오른쪽 마우스 버튼을 눌렀을 때 발생하는 메시지
 - WM_RBUTTONUP
 - 오른쪽 마우스 버튼을 떼었을 때 발생하는 메시지
 - WM_MOUSEMOVE
 - 마우스를 움직일 때 발생하는 메시지
 - WM_LBUTTONDOWNBLCLK / WM_RBUTTONDOWNBLCLK
 - 버튼 더블 클릭 눌렀을 때 발생하는 메시지
 - 윈도우 클래스가 반드시 CS_DBLCLKS 스타일을 가져야 한다

마우스 좌표 구하기

- 마우스에 대한 데이터 값은 IParam 에 저장

- `int y = HIWORD (IParam)`
- `int x = LOWORD (IParam)`

- **HIWORD**: 32bit 데이터에서 상위 16bit 데이터를 구하기 위한 매크로 함수
- **LOWORD**: 32bit 데이터에서 하위 16bit 데이터를 구하기 위한 매크로 함수



마우스로 원 선택하기

```
#include <math.h>
#define BSIZE 40          // 반지름

LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static int x, y;
    static BOOL Selection;
    int mx, my;

    switch (iMsg)
    {
        case WM_CREATE :
            x = 50;          y = 50;
            Selection = FALSE;           // 원이 선택되었나, FALSE : 아직 안되었음
            break;
        case WM_PAINT :
            hdc = BeginPaint (hwnd, &ps) ;
            //--- 만약 원이 선택되었다면, 4각형을 그린다. 아니면 원만 그린다.
            if (Selection)
                Rectangle(hdc, x-BSIZE, y-BSIZE, x+BSIZE, y+BSIZE);
            Ellipse(hdc, x-BSIZE, y-BSIZE, x+BSIZE, y+BSIZE);
            EndPaint (hwnd, &ps) ;
            break;
        case WM_LBUTTONDOWN :           // 왼쪽 버튼 누르면
            mx = LOWORD(lParam);
            my = HIWORD(lParam);
            if (InCircle (x, y, mx, my)) // 원의 중심점, 마우스 좌표 비교
                Selection = TRUE;        // 원 안에 있으면 '참'
            InvalidateRect (hwnd, NULL, TRUE);
            break;
    }
}
```

마우스로 원 선택하기(계속)

```
case WM_LBUTTONDOWN :  
    Selection = FALSE;  
    InvalidateRect (hwnd, NULL, TRUE);  
    break;
```

// 왼쪽 버튼을 놓으면

```
case WM_DESTROY:  
    PostQuitMessage (0);  
    break;
```

```
}
```

```
return (DefWindowProc (hwnd, iMsg, wParam, lParam));
```

```
}
```

//--- (x, y)와 (mx, my)의 길이가 반지름보다 짧으면 true, 아니면 false

BOOL InCircle (int x, int y, int mx, int my)

```
{
```

```
    if (LengthPts (x, y, mx, my) < BSIZE)
```

```
        return TRUE;
```

```
    else
```

```
        return FALSE;
```

```
}
```

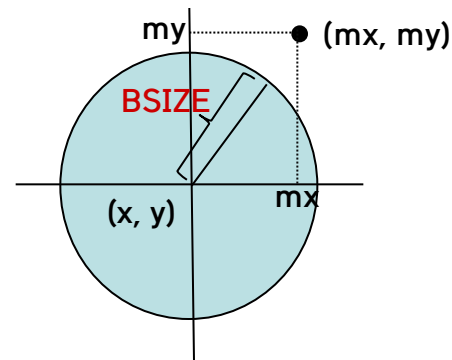
//--- (x1, y1)과 (x2, y2)간의 길이

float LengthPts (int x1, int y1, int x2, int y2)

```
{
```

```
    return (sqrt((x2-x1)*(x2-x1) +(y2-y1)*(y2-y1)));
```

```
}
```



마우스 드래그로 원 이동하기

```
LRESULT CALLBACK wndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
    static int x, y;
    static BOOL Selection;
    int mx, my;

    switch (iMsg)
    {
        case WM_LBUTTONDOWN :
            mx = LOWORD(lParam);
            my = HIWORD(lParam);
            if (InCircle(x, y, mx, my))
                Selection = TRUE;                // mx, my : 마우스 좌표
            InvalidateRect (hwnd, NULL, TRUE);
            break;
        case WM_LBUTTONUP :
            InvalidateRect (hwnd, NULL, TRUE);
            Selection = FALSE;
            break;
        case WM_MOUSEMOVE:
            mx = LOWORD(lParam);
            my = HIWORD(lParam);
            if (Selection)                        // 원이 선택된 상태로 움직이면
            {
                x = mx;
                y = my;
                InvalidateRect (hwnd, NULL, TRUE);    // 원과 사각형 그리기
            }
            break;
    }

    return (DefWindowProc (hwnd, iMsg, wParam, lParam));
}
```

4. 래스터 연산

- 객체 움직이기 위해 다시 그리기

- 화면 전체를 다시 그리기 한다.
- 빠른 움직임을 위해 윈도우 전체를 삭제하고 다시 그리는 것은 not good!

- 래스터 연산

- 윈도우의 배경색과 그리는 색을 연산한 결과 색상으로 그림
- AND, OR, XOR 등 비트간의 이진 연산과 NOT 연산의 조합으로 지정됨
- 그리기 연산은 래스터 디바이스에만 적용되며 벡터 디바이스에는 적용되지 않음
- 래스터 연산 함수 사용
 - 움직이거나 동적으로 표현되는 상태인 경우: 재 출력할 필요 없이 해당 메시지에서 GetDC()로 즉각 출력한다.
 - 선을 그릴 때 마우스를 드래그하면, 마우스 이동 메시지에서 이전의 선을 지우고 새로운 선을 그려야 한다.
- Raster Operation (Bitwise Boolean 연산)
 - Raster: 이미지를 점들의 패턴으로 표현하는 방식 (cf. Vector)

래스터 연산

- **int SetROP2 (HDC hdc, int fnDrawMode);**

- 두 픽셀 사이에 bit 연산을 수행하도록 mix 모드를 설정할 수 있는 기능
 - hdc: 디바이스 컨텍스트 핸들
 - fnDrawMode: 그리기 모드

그리기 모드	의미
R2_BLACK	픽셀은 항상 0(검정색)이 된다
R2_COPYPEN	픽셀은 사용된 펜의 색상으로 칠해진다
R2_MASKNOTPEN	펜의 색상을 반전시켜 배경과 AND 연산한다
R2_MASKPEN	펜의 색상과 배경을 AND 시킨다
R2_MASKPENNOUT	배경색을 반전시켜 배경과 OR 연산한다
R2_MERGEPEN	펜의 색상과 배경을 OR 시킨다
R2_MERGEPENNOT	배경색을 반전시켜 펜의 색상과 OR 연산한다
R2_NOP	픽셀은 아무런 영향을 받지 않는다
R2_NOT	배경색을 반전시킨다
R2_NOTCOPYPEN	펜의 색상을 반전시켜 칠한다
R2_NOTMASKPEN	R2_MASKPEN의 반전효과
R2_NOTMERGEPEN	R2_MERGEPEN의 반전효과
R2_NOTXORPEN	R2_XORPEN의 반전효과
R2_WHITE	픽셀은 항상 1(흰색)이 된다
R2_XORPEN	펜의 색상과 배경을 XOR 시킨다

래스터 연산

- 그리기 모드에서

- R2_COPYPEN;

- 펜이나 브러시의 default 동작
 - 바탕색은 무시하고, 그리고자 하는 색을 보여 줌

- R2_XORPEN;

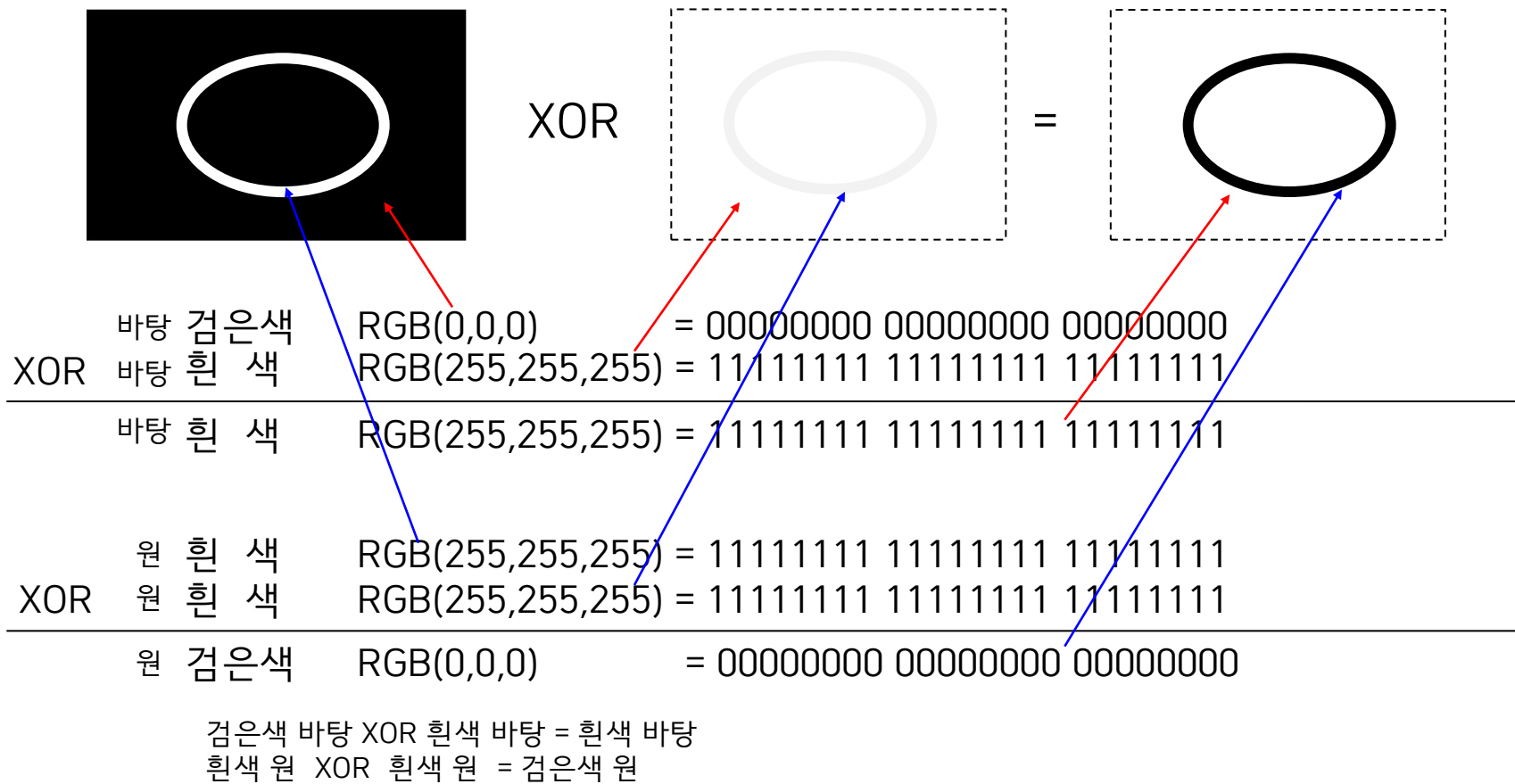
- 바탕색과 그리는 색 사이의 XOR 연산을 수행
 - XOR 연산 : 두 개의 비트가 다를 때만 true(1), 같으면 false(0)

- 사용 예)

- SetROP2 (hdc, R2_COPYPEN); // 설정된 펜의 색으로 선을 그린다.
 - SetROP2 (hdc, R2_MASKPEN); // 펜의 색과 배경색을 AND 연산
 - SetROP2 (hdc, R2_XORPEN); // 펜의 색과 배경색을 XOR 연산
// 펜: (255, 0, 0), 배경색 (0, 0, 255) → 화면에 그려지는 펜 색: (255, 0, 255)

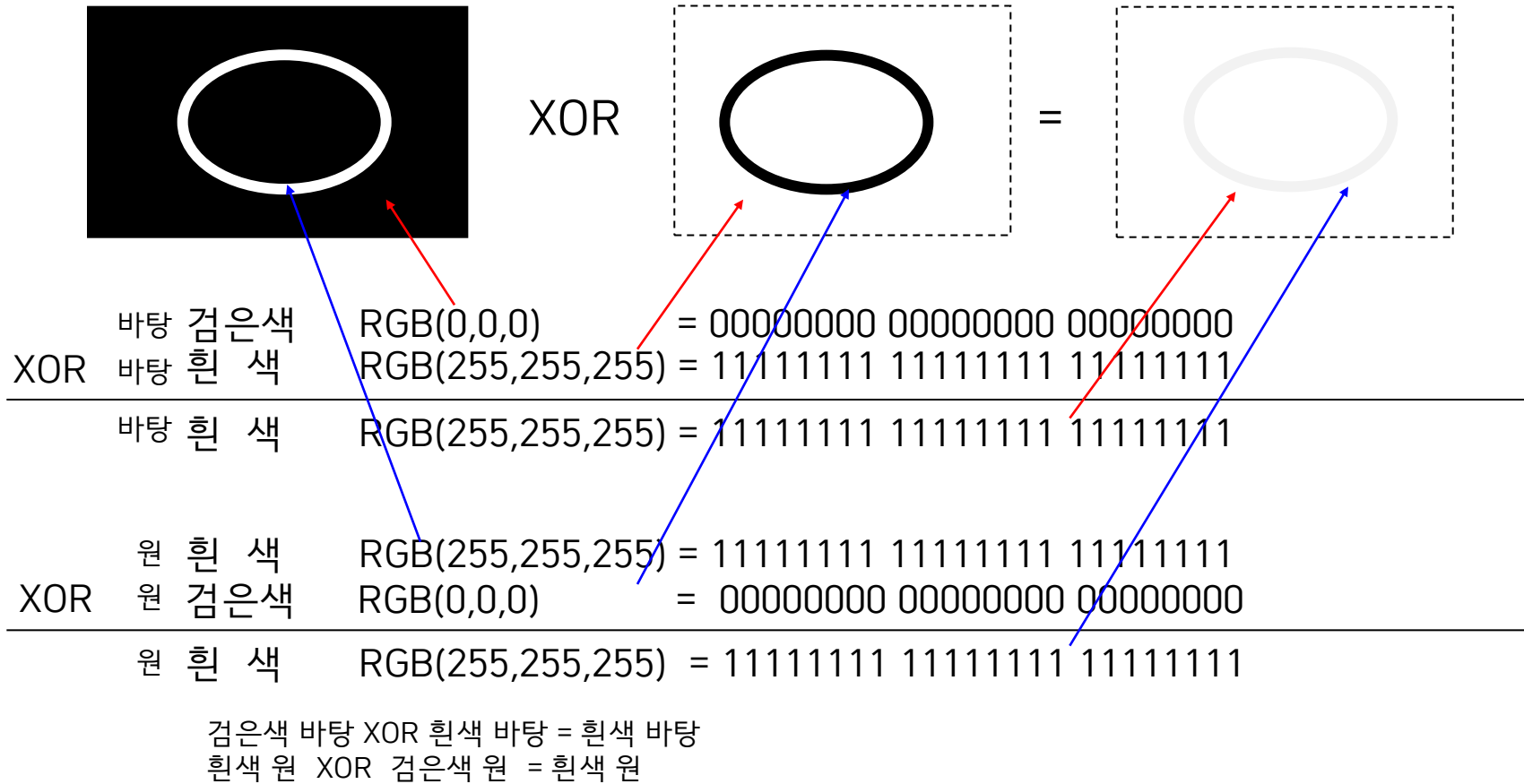
래스터 연산으로 지우기

- SetRop2 (hdc, R2_XORPEN);



래스터 연산으로 지우기

- SetRop2 (hdc, R2_XORPEN);



고무줄 효과가 있는 직선그리기

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM wParam, LPARAM lParam)
```

```
{
```

```
    static int startX, startY, oldX, oldY;
```

```
    static BOOL Drag;
```

```
    int endX, endY;
```

```
    switch (iMsg)
```

```
    {
```

```
    case WM_CREATE :
```

```
        startX = oldX = 50;    startY = oldY = 50;
```

```
// 시작 좌표
```

```
        Drag = FALSE;
```

```
        return 0 ;
```

```
    case WM_PAINT :
```

```
        hdc = BeginPaint (hwnd, &ps) ;
```

```
        MoveToEx(hdc, startX, startY, NULL);
```

```
// 이동하고 선으로 연결
```

```
        LineTo(hdc, oldX, oldY);
```

```
        EndPaint (hwnd, &ps) ;
```

```
        return 0 ;
```

```
    case WM_LBUTTONDOWN :
```

```
// 버튼을 누르면 드래그 동작 시작
```

```
        Drag = TRUE;
```

```
        break;
```

```
    case WM_LBUTTONUP :
```

```
// 버튼을 놓으면 드래그 종료
```

```
        Drag = FALSE;
```

```
        break;
```

고무줄 효과가 있는 직선그리기(계속)

```
case WM_MOUSEMOVE:
    hdc = GetDC(hwnd);
    if (Drag)
    {
        SetROP2(hdc, R2_XORPEN);
        SelectObject(hdc, (HPEN)GetStockObject(WHITE_PEN));
        endX = LOWORD(lParam);
        endY = HIWORD(lParam);

        MoveToEx(hdc, startX, startY, NULL);
        LineTo(hdc, oldX, oldY);

        MoveToEx(hdc, startX, startY, NULL);
        LineTo(hdc, endX, endY);

        oldX = endX; oldY = endY;
    }
    ReleaseDC(hwnd, hdc);
    break;

return (DefWindowProc (hwnd, iMsg, wParam, lParam));
}
```

// 흰 바탕
// 펜의 XOR 연산
// 흰 펜
// 흰 바탕 XOR 흰 펜 = 검은색 펜

// 지우기 : 흰 바탕 XOR 검은 펜 = 흰 선

// 그리기 : 흰 바탕 XOR 흰 펜 = 검은 선

// 현 지점을 이전 지점으로 설정

실습 3-4

- 제목

- 연습문제 3-1에 마우스를 이용한 기능 추가하기

- 내용

- 40x40보드를 그린다.
- 주인공원은 자동으로 이동하고 있다.
 - 주인공 원은 좌/우/상/하 방향 이동이 가능하다.
 - 벽에 닿으면 방향을 바꾼다.
- 화면에는 꼬리원 외에 임의의 개수의 먹이와 폭탄 (폭탄의 개수는 먹이의 2배) 이 있다.
 - 먹이와 폭탄은 다른 색 또는 다른 모양으로 표시한다.
 - 먹이를 먹으면 폭탄이 하나 삭제된다.
 - 폭탄을 먹으면 꼬리가 하나 사라진다.
- 왼쪽 마우스 버튼 기능
 - 왼쪽 마우스 버튼을 빈 보드에 누르면 그 방향으로 원의 이동 방향이 바뀐다.
 - 왼쪽 마우스 버튼으로 주인공 원의 내부를 클릭하면 원의 크기가 커졌다 작아진다.
 - 왼쪽 마우스 버튼으로 먹이를 클릭하고 드래그하면 먹이의 위치가 바뀐다.
- 오른쪽 마우스 버튼 기능:
 - 오른쪽 마우스 버튼을 누르면 주인공원은 그 자리에 멈추고 먹이가 임의의 방향으로 움직인다.
 - 다시 오른쪽 마우스 버튼을 누르면 먹이가 멈추고 주인공원이 다시 움직이기 시작한다.

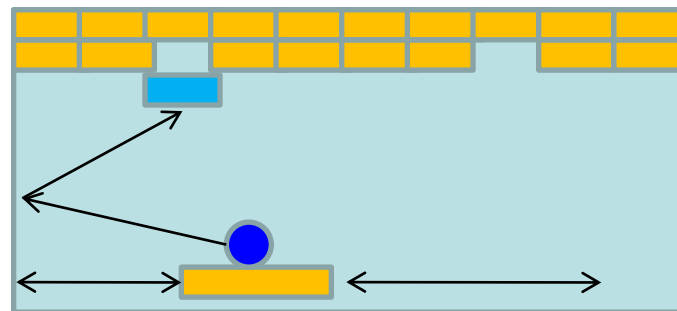
실습 3-5

- 제목

- 벽돌 깨기 게임 만들기

- 내용

- 화면의 상단에 2*10 개의 벽돌이 있다.
 - 벽돌들은 시간에 따라 좌우로 왔다갔다한다.
- 화면의 하단에 바가 있고 마우스를 이용하여 바를 움직인다.
 - 바닥의 벽돌을 마우스로 선택하고 드래그하여 이동한다.
- 공이 튀기면서 벽돌에 1번 닿으면 벽돌의 색이 바뀌며 한 칸 내려온다.
- 공이 튀기면서 벽돌에 2번 닿으면 벽돌이 없어진다.
- 색이 변한 벽돌의 개수와 없어진 개수를 화면에 출력한다.
- 키보드 명령어
 - +/- 입력: 공의 이동 속도가 늘어난다.
- 벽돌이 모두 없어지면 게임이 종료된다.



실습 3-6

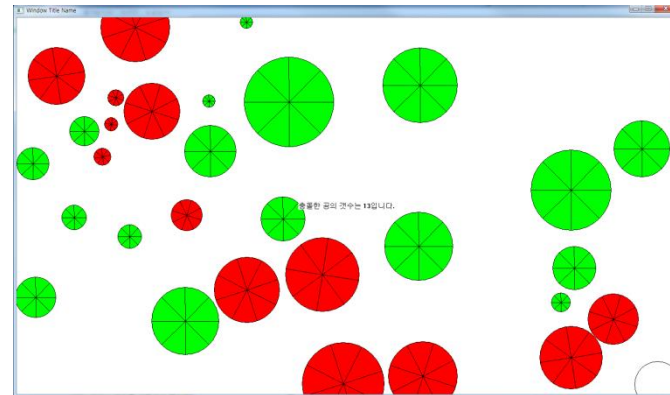
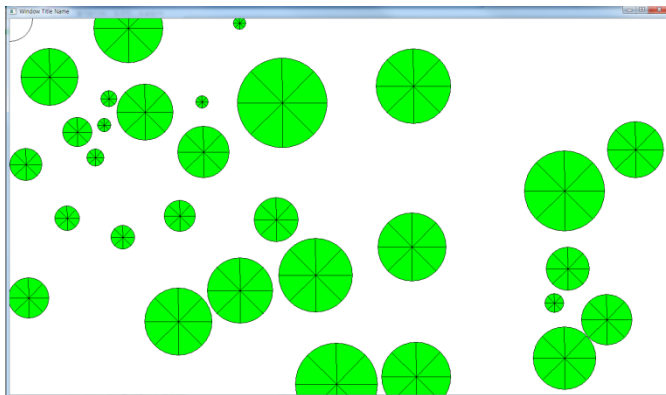
실습 3-7

- 제목

- 장애물 피하여 도형 이동하기

- 내용

- 프로그램이 시작하면 화면에 장애물 역할을 할 크고 작은 원들이 많이 나타나고 주인공 역할을 할 흰색 원이 중심좌표 (0,0), 반지름 50으로 나타난다.
- 장애물 역할의 원들의 내부에 바퀴모양의 직선을 그린다.
 - 몇몇 원들은 시계 방향으로 회전하고 있다.
- 마우스 드래그로 흰색 원을 반대쪽 모서리까지 이동 시키면 기록으로 부딪힌 장애물 숫자를 화면에 출력한다.
 - 이동하는 원과 부딪친 원은 시계방향으로 회전한다.
 - 이미 회전하고 있던 원들은 반대방향으로 회전한다.
 - 부딪친 원의 색은 주인공 원과 특정 래스터 연산을 하여 결정한다.
- 주인공인 흰 원이 가장자리에 도달하면 회전을 멈춘다.
- 키보드 명령어를 입력하여 새로 게임을 시작할 수 있도록 한다.



실습 3-8