VC++

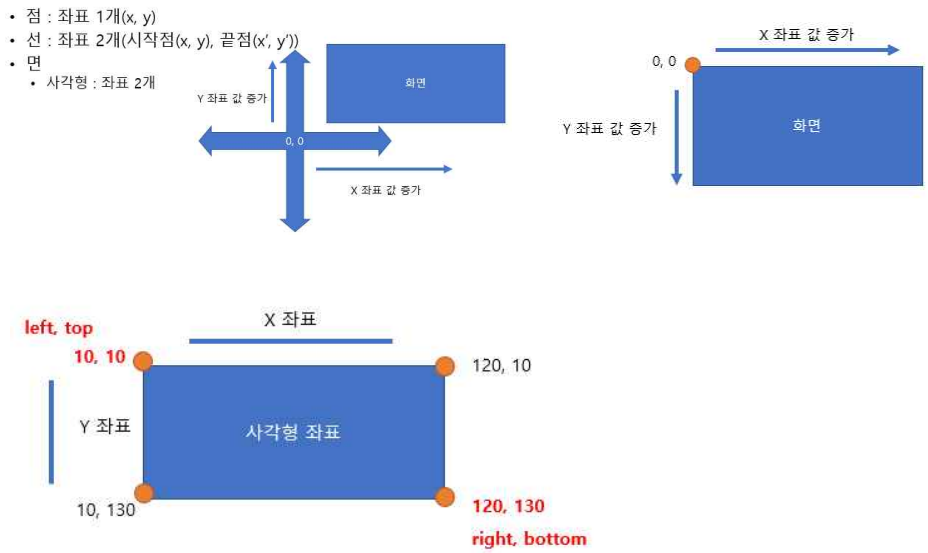
1. 중간고사

- 1번 문제 정답 : WndProc
- 시험 : 주관식 (단답형, 서술형)
- 문제 : 15 ~ 20
- 코드 보는 문제가 있음. 코드 작성 문제도 있을 수 있음
  Ex) WM_BUTTONDOWN (O) → Wm_lButtondown (X)
  대소문자 구별해서 답안 작성해야 함
- 10문제 : 1문제 (답 알려줌), 1문제 (매우 어려움), 나머지 (수업시간)

2. Win32에 대해 알아보기

- GUI 프로그램 작성 : 작성하는 App. 윈도우에 등록한다.
- 콜백 함수 : OS에 프로그램을 등록할 때, OS에게 동시에 전달한다.
- 사용자 입력 → OS가 해석
- 정보 확인 → 응용 프로그램에 전달 (WndProc)
- 응용 프로그램에 전달할 때는 MSG 형태로 전달
- WndProc 함수 인수 : 사용자가 입력한 정보를 처리한다.
- HWND : 사용자가 입력한 윈도우 자체를 가리킨다.
- UINT : 사용자가 무엇을 입력했는지를 나타낸다.
- WPARAM (Word Parameter) : char 자료형, 키보드 입력과 관련된 정보
- LPARAM (Long Parameter) : int 자료형, 마우스 입력과 관련된 정보

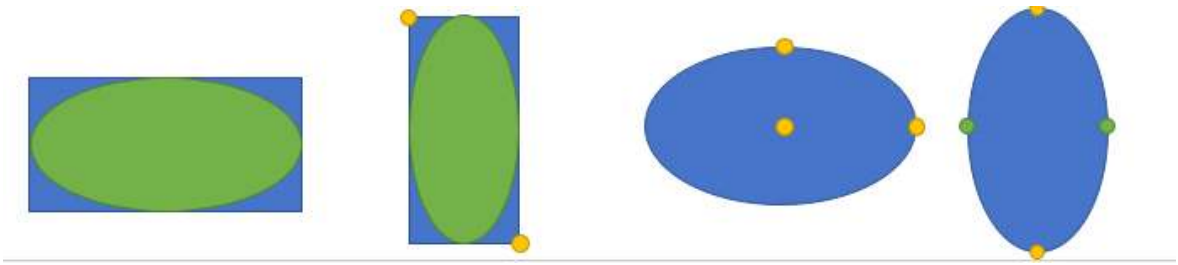
3. 화면 그리기와 윈도우의 화면 좌표계



Ⅰ. 화면 그리기

1. 윈도우(GUI 환경)에서의 화면 그리기

- 화면은 OS의 것이다. 따라서 그리기는 결국 OS에게 요청하는 것이다.
- 선, 면 (사각형) : 2D 그래픽은 그리는 순서가 가장 아래에 위치한다.
- 그리기 요청 API
    - 선 그리기 : MoveToEx( ) - 시작점 설정 , LineTo( ) - 끝점 설정 및 그리기 요청
    - 사각형 그리기 : Rectangle( ) - 2개의 좌표 값을 전달 (left, top, right, bottom)
    - 타원 그리기 : Ellipse( ) - 2개의 좌표 값을 전달 – 내접하는 원을 그린다.



2. WM_PAINT

- 내부에 구현된 코드를 어떤 상태에서든지 반드시 화면에 출력한다.
- 화면의 모든 변경 → WM_PAINT가 호출된다
    - 창의 크기 조절, 창의 최대화, 창의 최소화, 화면 밖으로 이동
    - 프로그램이 실행된 다음 OS에 의해서 자동적으로 한번 호출
    - 프로그래머의 요청에 의해 OS가 호출해주는 경우

3. HDC (Handle Device Context)

- 화면에 그리기를 요청할 때, 반드시 필요한 자료 구조
- 획득 방법 1) WM_PAINT에서 BeginPaint( ) 획득, 해제는 EndPaint( )
- 획득 방법 2) WM_PAINT에서 GetDC( ) 획득, 해제는 ReleaseDC( )

II. 사용자 입력의 마우스 및 자유선 그리기

1. 마우스 입력

- WM_L/RBUTTONDOWN, WM_L/RBUTTONUP, WM_MOUSEMOVE
- H/W 관련된 사용자 입력은 모든 단계가 별도로 구성된다.
- 단계가 별도로 구성되기 때문에 그때마다 WM가 발생한다.
- 콜백 함수 : OS가 호출할 수 있는 함수. **일반 함수와 동일한 속성**
- 이전에 발생한 콜백에서 정보를 보관하려면 전역 변수를 사용해야 한다.
- x , y 좌표를 획득 방법
    - lParam에서 정보를 획득
    - LOWORD( ) 매크로 → x 좌표
    - HIWORD( ) 매크로 → y 좌표

2. 자유선 그리기

III. 이벤트와 API

1. 마우스 이벤트

- WM_L/RBUTTONDOWN-UP-MOVE
- 모든 이벤트는 단독 (콜백 함수로 정보 전달)으로 동작한다.
- 전역 변수 필요한 이유 : 콜백 함수 간의 정보를 유지
- 플래그 변수 : 일반 변수와 동일한다. 상태 정보를 갖는다.

2. 키보드 이벤트

- 윈도우 메시지
    - WM_KEYDOWN/UP
    - ASCII 코드로 눌려진 키의 값이 전달되었다.
- 문자열 처리
    - 화면에 그림 대신 글자를 그리는 API
    - 자료형 : WCHAR (== wchar_t)
    - wsprintf( ) : 선언된 문자열 변수에 지정 형식으로 내용을 채우는 함수
    - TextOut( ) API : 화면에 지정된 문자열을 출력하는 API
    - lstrlenW( ) : 유니코드 문자열의 정확한 길이를 알려주는 함수

3. HDC 이용하는 API 및 HDC 속성 변경

- HDC
    - 화면 정보를 보관하는 정보체 – 내용을 알 수 없다.
    - 화면 그리기에 필요한 정보 포함 – 변경 요청
    - 변경은 OS에게 요청해서 변경이 발생
- 2개의 속성 변경
    - 선 객체의 속성 변경 : 기본 선은 굵기 1px에 검은색
    - 면 객체의 속성 변경 : 기본 면은 하얀색
- 변경 방법 (예 – 선 변경)
    - 선 객체 생성 → OS에게 선 객체를 전달 → 원래 선 객체 반환 → 선을 사용 → OS에게 원래 선 객체를 전달
- API를 이용하여 변경
    - HDC 정보는 간접적으로 접근 불가 → OS에 요청
    - SelectObject( ) API : HDC 객체 정보 변경
    - DeleteObject( ) API : 변경을 위해 생성한 HDC 객체 해제
    - HPEN, HBRUSH 객체들을 생성하여 테스트 진행
    - 변경 순서
        1. HDC 객체 생성
        2. 객체 변경 요청 (SelectObject)
        3. 현재 OS가 사용하는 객체를 반환
        4. 사용한다
        5. 객체 변경 요청
        6. 객체의 삭제 및 해제 (DeleteObject)

IV. 게임 관련

1. RECT 자료형

- 사각형의 좌표 표시를 위해 사용
    - 2개의 좌표 값을 보관하는 자료형
    - left, top : 좌상단 x, y 좌표
    - right, bottom : 우하단 x, y 좌표
    - 좌상단, 우하단에 해당하는 x, y 좌표를 정확하게 입력한다

```
typedef struct tagRECT
{
    LONG    left;
    LONG    top;
    LONG    right;
    LONG    bottom;
} RECT, *PRECT, NEAR *NPRECT, FAR *LPRECT;
```

- 단순 구조체 (정보체)
    - left, top, right, bottom으로 구성된 단순한 형태
    - left, top : 반드시 좌상단의 x, y 좌표로 구성
    - right, bottom : 반드시 우하단의 x, y 좌표로 구성

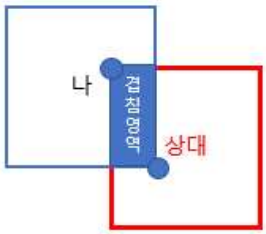
- 나의 객체 이동
    - 화면에 나의 객체 정보 표현을 위해 RECT 자료형 이용
    - WM_KEYDOWN에서 키 값을 확인하고, 좌표를 이동
    - 좌/우 이동 : left 또는 right 값을 하나만 변경 → 직사형 형태로 늘어남
    - 이동 시에는 left/right, top/bottom 값은 항상 같이 변경해야 함
- 단순 자료형
    - 반드시 해당 위치 좌표를 입력해야 정상 동작하는 API가 존재
        - 해당 API를 사용하지 않고도 계산은 가능
    - 겹침 확인을 위해서는 반드시 RECT 자료형을 사용

2. 표준

- H/W 장비의 표준화
    - a 키 입력 → 0x76 값이 전달 → OS가 받음 : a / a
    - OS는 항상 보고 있는 키보드가 별도로 존재
    - 콜백 함수 호출을 위한 준비
        - 누구한테 전달할 것인지, 호출 준비, 인수 준비 ...
        - 콜백 함수를 호출해서 응용 프로그램에 정보를 전달
    - OS가 항상 보고 있는 키보드 : 가상 키보드

3. 겹침 확인/처리 API

- IntersectRect( )
    - 전달 좌표 : 나
    - 상대 좌표 → RECT 자료형이 필수
    - 반환되는 정보 : 겹침 여부, 겹침 영역
        - 겹쳤다면 겹침이 발생한 영역 정보가 RECT 자료형으로 반환



- IntersectRect( ) API
    - OS에게 겹침 발생 여부를 확인할 때 사용
    - 2개의 반환 값
        - 겹침이 발생했는지 여부 : TRUE (겹침이 있다), FALSE (겹침이 없다)
        - 겹침이 발생한 영역에 대한 RECT 자료형 반환
    - BOOL IntersectRect(LPRECT dst, const RECT* src1, const RECT* src2)
        - src1/2 : 겹침을 확인할 RECT 자료형
        - dst : 겹침이 발생, 겹침이 발생한 영역 정보

4. 객체 추적

- 나와 상대
    - 상대가 나를 추적하는 방법
    - 좌표를 이용해서 추적
    - RECT의 좌상단 x, y 좌표 값의 크기를 이용하여 추적
        - 상대가 나를 기준으로 왼/오른쪽에 존재하는지 구분
        - 상대가 나를 기준으로 위/아래에 존재하는지 구분

5. 주기적인 호출

- OS에게 알람을 요청
    - SetTimer( ) API : 알람 설정, msec 단위로 시간 설정
    - KillTimer( ) API : 알람 해제. SetTimer에서 설정한 ID 값이 필요
    - 알람이 도착하는 윈도우 메시지 : WM_TIMER
    - 알람이 도착했을 때, ID를 구분하기 위한 방법
        → wParam에 SetTimer의 ID으로 구분

기타
WM_CREATE 버튼 생성
- 클래스의 생성자와 동일한 역할 담당
- 시작과 동시에 화면에 버튼 윈도우를 생성하도록 구성

WM_DESTROY 버튼 해제

CreateWindow( ) API
- 윈도우를 생성하는 API
- 버튼도 윈도우이므로 CreateWindow( )로 생성

버튼 클릭 이벤트
- 버튼이 클릭되면 구분자(ID) 값이 WM_COMMAND로 전달
- CreateWindow에 설정한 구분자(ID) 값을 확인하여 필요한 작업 수행
- 정확한 확인을 위해 구분자(ID) 값을 선언하여 사용

- 실습에서는 메시지 박스를 출력하여 이벤트 받았음을 표시

버튼 윈도우 핸들은 전역 변수로 선언
- CALLBACK 함수의 함정의 함정을 회피하기 위함

WM_DESTROY에서 윈도우 핸들을 제거
- 버튼 역시 윈도우의 자원이므로, 프로그램 종료 시 해제해야 함

==========
HWND g_button;
==========
#define IDM_BTN_CLICK 999
==========
case IDM_BTN_CLICK:
MessageBox(hWnd, L"NFIASFNV", L"casijcisa", MB_OK
==========
case WM_CREATE:
g_button = CreateWindow(L"button", L"클릭", WS_CHILD | WS_VISIBLE, x, y, 가로, 세로, hWnd, (HMENU)IDB_BTM_CLICK, hInst, NULL);
==========
종료 시
DestroyWindow(g_button);

GDI (Graphic Device Interface)

펜 객체
- HPEN 자료형으로, 선 색상을 지정하는 객체
- CreatePen( ) API로 원하는 펜을 생성

브러시 객체
- HBRUSH 자료형으로 면 색상을 지정하는 객체
- CreateSolidBrush( ) API로 원하는 브러스를 생성

자원 관리
- OS의 모든 자원은 항상 OS에게 요청하여 획득하는 형태
- 요청한 자료형은 일반적으로 OS가 생성하고, 포인트를 반환
- 생성한 자료형이나 핸들은 반드시 해제하여야 함.

GDI 객체 사용과 해제
- GDI 객체 변경
- HGDIOBJ SelectObject (HDC, HGDIOBJ);
- 반환 값으로 항상 현재 사용 중인 객체가 반환
- 현재 객체는 반드시 보관하고, 변경 객체가 모두 사용되면 다시 설정해야 함

- GDI 객체의 삭제 (자원 해제)
- BOOL DeleteObject (HGDIOBJ) ;

특정 시간마다 OS에 의한 호출
- 특정 시간이나 이벤트를 기다려야 하는 경우
- 자체적으로 시간을 처리할 경우, 다양한 문제가 발생할 수 있음
- OS에게 호출을 요청하고 별도의 작업을 수행
- 모든 윈도우 메시지가 독립적으로 수행되므로 문제 없음
- 일종의 스레드 동작 효과를 볼 수 있음

윈도우 메시지를 호출해 줄 것을 요청
- WndProc( ) 함수에서 해당 메시지를 처리해야 함
- 구분을 위해 전달되는 정보는 wParam을 통해 전달
- 주기적인 호출을 요청하기 위한 API와 해제를 위한 전용 윈도우 메시지를 이용
- 처리는 주기적인 메시지를 처리하기 위한 전용 윈도우 메시지를 이용

윈도우 메시지
- WM_TIMER 를 통해 요청한 메시지가 전달
- 주기적인 요청을 할 경우, 항상 WM_TIMER 메시지로 호출
- 기존 윈도우 메시지와 동일한 형태로 처리

주기적인 호출 요청 API
- UINT_PTR SetTimer(
```
HWND        hWnd,          // 호출을 받을 윈도우
UINT_PTR            nIDEvent,          // WPARAM에 전달될 ID 값
UINT                uElapse,           // 시간 주기
TIMERPROC   lpTimerFunc    // 호출을 처리할 함수
);
```
- 호출 시점부터 WM_TIMER가 지정 시간 주기마다 OS에 의해 호출

호출 해제 API
- BOOL KillTimer(
```
HWND        hWnd,          //  호출을 받던 윈도우
UINT_PTR            nIDEvent           // 해제할 ID 값
);
```
- 호출 시점 이후로 OS에 의해 호출되던 WM_TIMER가 더 이상 호출되지 않음

활용
- 자동적으로 이동해야 하는 객체의 자동 이동에 이용할 수 있음
- 키보드 메시지 처리를 통한 사각형 이동 형식으로 활용 가능
- ID 값을 고정하고, 시간 값을 수정하며 빠른 호출 수행이 가능
- 게임의 다수 건물 짓기와 같은 시간 재기 등에도 사용 가능
    > 실제로는 스레드를 이용하여 처리하는 것이 정상 처리