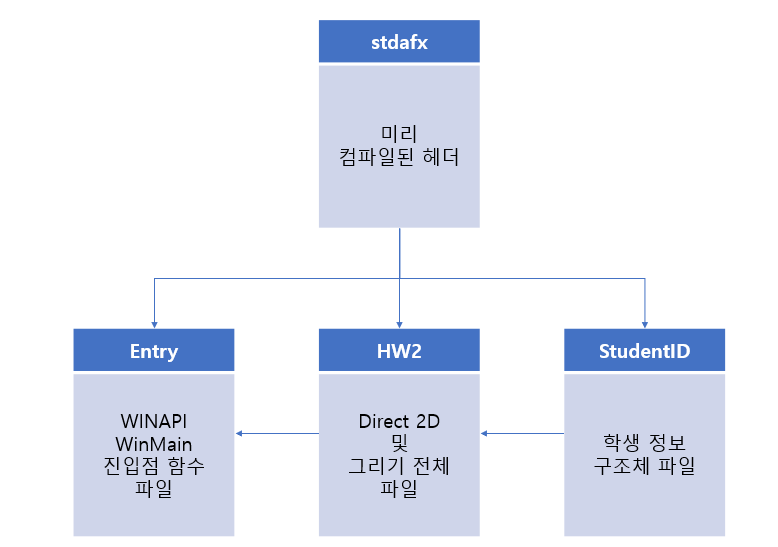
|  |
| --- |
| **게임 프로그래밍** |

- 과제 2 -

|  |  |
| --- | --- |
| 과목 | 게임 프로그래밍 |
| 교수 | 박종승 교수님 |
| 제출일 | 2021.04.16 |
| 학번 | 201601607 |
| 학부 | 컴퓨터공학부 |
| 이름 | 이명진 |
| E-mail | mpsa001@naver.com |
| 연락처 | 010-9929-6564 |

# 1: 코드 구조



StudentID  
- 학생 정보 구조체  
- 배열을 통해 학생의 정보를 생성하는 함수  
-> HW1의 내용을 가져옴

stdafx(미리 컴파일된 헤더)

HW2  
- 진입점을 제외한 모든 함수(Direct2D, 프로시저)들이 존재하는 파일

Entry  
- 진입점 함수인 WinMain 함수가 존재

# 2: 구현 주요 내용

학생의 정보 생성과 vector

HW1과 동일한 과정으로 전역으로 설계된 배열로 값을 임의로 받아와 생성 후 vector에 삽입한다  
- 이전과 동일한 형식을

|  |
| --- |
| StudentID.cpp의 학생 정보를 만드는 함수 |
| // 학생의 이름을 만들때 사용할 배열  std::string familyName[12] =  {  "김", "이", "박", "최",  "남", "정", "한", "류",  "신", "유", "마", "주"  };  std::string firstName[12] =  {  "형", "남", "명", "승",  "학", "의", "찬", "우",  "홍", "철", "규", "대"  };  std::string secondName[12] =  {  "현", "진", "준", "성",  "기", "지", "은", "훈",  "연", "원", "빈", "선"  };  // --------------- 구현부 --------------------  // 학생의 정보를 만들기 위한 함수  std::string rtnName() //이름 생성 함수  {  srand((unsigned int)time(NULL));  std::string studentName = familyName[rand() % 12] + firstName[rand() % 12] + secondName[rand() % 12];  return studentName;  }  int rtnScore() //점수 생성 함수  {  srand((unsigned int)time(NULL));  int score = rand() % 50 + 50;  return score;  } |
| HW2.cpp - vector에 학생 정보를 삽입 |
| // 데이터를 vector에 삽입  void dataPush()  {  student tmp = { rtnScore(), rtnName() };  vec.push\_back(tmp);  // 임시로 student 구조체 변수를 하나 생성한 후 vector에 넣는 동작을 수행한다  } |

vector의 sort

HW1과 동일하게 커스텀 비교 함수를 사용해서 student 타입의 vector를 sort할 수 있도록 하였다. 이후 정렬된 내용을 for문을 사용하고 변환과정을 거쳐 존재하는 박스들에 출력한다  
- 과제 내용의 정렬된 내용을 박스에 출력하는 과정을 구현

|  |
| --- |
| vector의 sort 함수 |
| // vector를 정렬하기 위한 커스텀 비교함수  bool compare(student a, student b)  {  return a.score > b.score;  }  // 커스텀 비교자를 사용해서 student 구조체를 사용하는 vector를 정렬한다  sort(vec.begin(), vec.end(), compare); |
|  |
| for (int i = 0; i < Vectorsize; i++)  {  m\_pRenderTarget->FillRectangle(getNextRectangle(i), m\_RectangleBrush); // 그다음 그려질 스택 위치를 얻어내서 사각형을 그려준다.  const char\* studentName = vec[i].name.c\_str(); // v[i].name을 const char\*형으로 바꾼다.    TCHAR temp[15]; // const char\*을 TCHAR형으로 바꿔야 하기 때문에 temp변수를 선언한다.  memset(temp, 0, sizeof(temp)); // 초기화  MultiByteToWideChar(CP\_ACP, MB\_COMPOSITE, studentName, -1, temp, 15);    static WCHAR studentInfo[100]; // name과 점수를 한번에 넣어아 햐므로 WCHAR 배열형 변수를 선언한다.  swprintf\_s(studentInfo, L"%s %d\n", temp, vec[i].score);  m\_pRenderTarget->DrawText(studentInfo, wcslen(studentInfo), m\_pTextFormat, getNextRectangle(i), m\_pTextBrush); // Text를 그려준다.  m\_pRenderTarget->DrawRectangle(getNextRectangle(i), m\_boundaryBrush); // 상자 테두리를 그려준다.  } |

이번에도 HW1처럼 string 타입을 변환하는 과정으로 진행하려 했지만 string을 사용할 수 없어 책의 다른 변환 과정으로 수행하였다.

사각형의 scale과 크기

과제의 내용에 상단의 가상의 직사각형과 하단의 직사각형의 크기를 설정하고 동시에 상단의 박스를 클릭한 경우 움직이면서 확대하며 하강하거나 반대로 축소되며 상승해야 하는 조건이 있다.

상단 박스와 하단 박스는 높이가 동일하지만 너비가 2배의 차이를 갖고 동시에 하단 박스의 높이와 너비의 비율은 1:4이다.  
- 많은 경우의 수가 존재하지만 힌트 코드의 내용대로 높이를 30으로 잡고 시작하였다.  
- 상단의 경우 30:60, 하단의 경우 30:120이 된다.

하강의 경우 높이의 확대는 불필요하므로 y 값만 변경하면 되는 상황이므로 이동 거리에 따른 y값의 차이와 상단의 박스와 하단에 생성될 박스의 크기를 계산, scale을 수행하면 된다  
- 식의 마지막에 +1을 수행해야 상단 박스의 크기에서 시작해 크기가 증가하며 하단 박스와 크기가 동일하게 커진다  
- 두 박스의 너비가 2배 차이기 때문

상승의 경우 역시 높이의 확대는 불필요하고 y의 값만 변경하면 된다. 하강과의 차이점은 +1로 하단의 박스와 크기를 맞추는 것이 아닌 가장 최근에 생성된 vector의 크기, 즉 상승 동작이 수행해야 할 제거할 박스의 좌표를 기준으로 크기를 맞춰야 한다  
- 식의 마지막에 가장 최근의 박스의 좌표를 읽어와 크기를 맞춰준다.

|  |
| --- |
| 하강(삽입)시의 크기 변환 |
| Size = ((current.y - lastdown.y) / (getNextRectangle(Vectorsize).top - 30)) + 1;  // 본래 크기에 추가적인 크기를 더해야 하므로 +1 을 해줘야  // 목표 지점까지 얼마만큼 왔는지 비율을 구해서 Size를 구해줍니다.  // 스케일 조절  D2D1\_MATRIX\_3X2\_F scale = D2D1::Matrix3x2F::Scale(D2D1::Size(Size, 1.0f), D2D1::Point2F(insertrectangle.left, insertrectangle.top)); |
| 상승(제거)시의 크기 변환 |
| if (current.y <= lastdown.y) // 만약 위로 드래깅 한다면  {  Size = (getNextRectangle(Vectorsize).top - 30) / ((lastdown.y - current.y) + (getNextRectangle(Vectorsize).top - 30));  // 이 경우에는 가장 최근의 vector에서 축소되는 것이므로 가장 최근의 vector의 위치를 읽어와야 한다.  // 목표 지점까지 얼마만큼 왔는지 비율을 구해서 Size를 구해줍니다. }  else //만약 아래로 드래깅 한다면  {  Size = ((current.y - lastdown.y) / (getNextRectangle(Vectorsize).top - 30)) + 1;  // 삽입 할때와 동일한 공식을 적용  }  D2D1\_MATRIX\_3X2\_F scale = D2D1::Matrix3x2F::Scale(D2D1::Size(Size, 1.0f), D2D1::Point2F(lastdown.x, lastdown.y));  // 읽어온 값을 기준으로 확대, 축소 동작을 수행 |
| 확신이 들지 않아 비교해 보았다 |
|  |

시계 회전과 반시계 회전

과제의 내용 중 하강에는 반시계 회전을, 상승에는 시계 회전을 구현하라는 내용이 존재한다.

하강 또는 상승시에 회전을 수행하면 되는 것이므로 회전도 x 값을 고려하지 않고 y값만 고려하면 된다. 따라서 회전의 각을 조절할 Angle 변수를 정할 식은 y값의 차이와 하단 박스와의 거리와의 비율에 따라서 구할 수 있다. 조건에서 360도 반시계, 시계 회전이 있으므로 360을 비율에 곱하고  
- 반시계 회전을 기준으로 작성하였기에 -1을 곱해준다.

|  |
| --- |
| 반시계 회전 |
| // 회전 조절  Angle = (current.y - lastdown.y) / (getNextRectangle(Vectorsize).top - 30) \* 360 \* -1;  D2D1\_MATRIX\_3X2\_F rotation =  D2D1::Matrix3x2F::Rotation(Angle, D2D1::Point2F(  (insertrectangle.right + insertrectangle.left) / 2,  (insertrectangle.top + insertrectangle.bottom) / 2)); |
| 시계 회전 |
| Angle = (current.y - lastdown.y) / (getNextRectangle(Vectorsize).top - 30) \* 360 \* -1;  D2D1\_MATRIX\_3X2\_F rotation = D2D1::Matrix3x2F::Rotation(Angle, D2D1::Point2F(lastdown.x, lastdown.y)); |

복합 변환은 책의 내용대로 Scaling -> Rotation -> Translation 순으로 구현하였다.

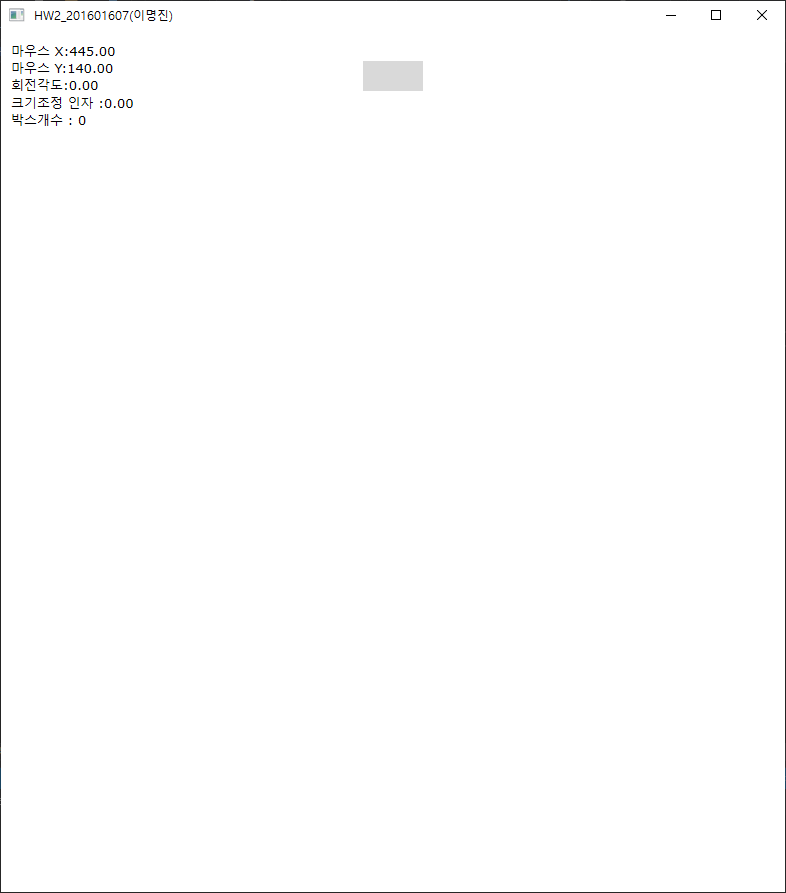
|  |
| --- |
| 복합 변환 |
| m\_pRenderTarget->SetTransform(scale \* rotation \* translation); // scale->rotation->translation 순 |

좌클릭과 클릭 해제

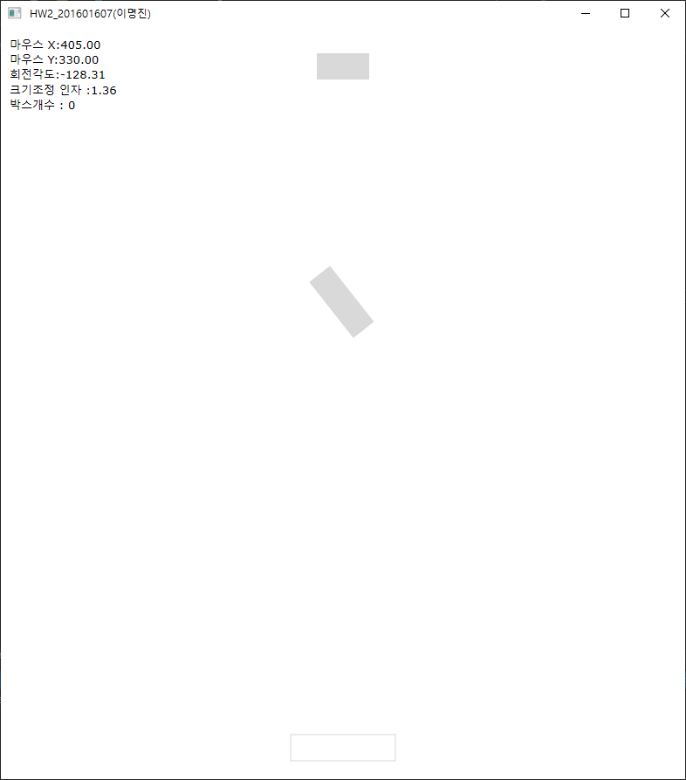
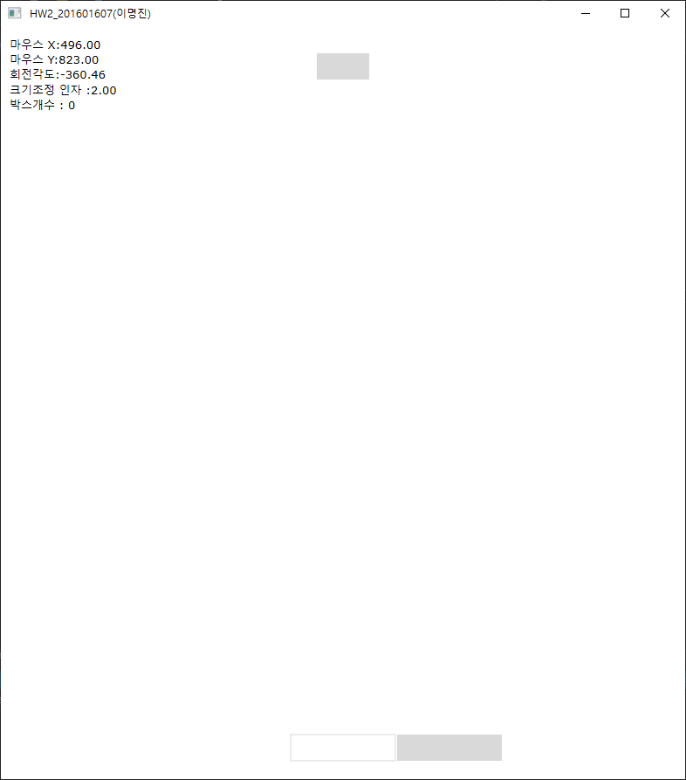
클릭에 대해서 상단의 박스를 클릭한 경우 삽입 과정을 수행해야 하며 하단의 가장 위에 배치된 박스를 클릭한 경우 제거를 수행해야 한다  
또한 클릭 중 해제에 대한 작업도 수행해야 하므로 이 상태를 나타낼 불리안 타입 변수들을 전역변수로 설정하고 각 상황에 맞는 작업을 수행하도록 조건문을 부여하였다.  
- 또한 이 변수들은 마우스의 이동 상태에서도 좌클릭의 상태를 판정할 수 있는 역할을 수행한다

|  |
| --- |
| 좌클릭과 좌클릭 해제 |
| // 좌클릭 해제에 대한 동작  case WM\_LBUTTONUP:  {  // 삽입, 삭제 동작을 표시하는 flag를 모두 초기화  inserttingflag = false;  deletingflag = false;  // 각도, 크기 초기화  Angle = 0;  Size = 0;  break;  }  // 좌클릭에 대한 동작  case WM\_LBUTTONDOWN:  {  if (LOWORD(lParam) >= insertrectangle.left && LOWORD(lParam) <= insertrectangle.right  && HIWORD(lParam) >= insertrectangle.top && HIWORD(lParam) <= insertrectangle.bottom) { // 만약 상단의 가운데 사각형이 눌려졌으면  inserttingflag = true; // insertting모드를 켜준다.  lastdown.x = LOWORD(lParam);  lastdown.y = HIWORD(lParam);  }  else if (vec.size() == 0)  break;  else  {  D2D1\_RECT\_F lastRectangle = getNextRectangle(Vectorsize - 1); // 이전의 vector를 가리킨 경우라면  if (LOWORD(lParam) >= lastRectangle.left && LOWORD(lParam) <= lastRectangle.right  && HIWORD(lParam) >= lastRectangle.top && HIWORD(lParam) <= lastRectangle.bottom)  {  // deleting모드를 켜준다.  deletingflag = true;  lastdown.x = LOWORD(lParam);  lastdown.y = HIWORD(lParam);  }  }  break;  } |
| 마우스의 움직임 처리 |
| // 마우스의 움직임에 따른 동작  case WM\_MOUSEMOVE:  {  // 현재 마우스의 윈도우 상 좌표  current.x = LOWORD(lParam);  current.y = HIWORD(lParam);  // 삽입 동작이라면  if (inserttingflag == true)  {  // 만약 maxsize를 초과한다면  if (Vectorsize >= maxsize)  {  inserttingflag = false; // inserttingflag를 false로 처리해준다.  ::MessageBox(0, L"the number of box is 8, it's limit", L"Fatal Error", MB\_OK);//메세지를 띄워준다.  ::MessageBeep(MB\_OK);  break;  }  D2D1\_RECT\_F currentVector = getNextRectangle(Vectorsize);  if (LOWORD(lParam) >= currentVector.left &&  LOWORD(lParam) <= currentVector.right &&  HIWORD(lParam) >= currentVector.top &&  HIWORD(lParam) <= currentVector.bottom)  {  // inserttingflag가 true이고 들어와야할 스택에 마우스 커서가 들어왔으면  inserttedflag = true; // inserttedflag를 true로 맞춰준다.  Angle = 0; //Angle ,Size 초기화  Size = 0;  inserttingflag = false;  }  InvalidateRect(hwnd, NULL, false);  }  // 제거 동작이라면  else if (deletingflag == true)  {  if (LOWORD(lParam) >= insertrectangle.left && LOWORD(lParam) <= insertrectangle.right  && HIWORD(lParam) >= insertrectangle.top && HIWORD(lParam) <= insertrectangle.bottom)  {  // deletingflag가 true이고 insertingrectangle에 마우스 커서가 들어왔으면  deletedflag = true; // deletedflag를 true로 맞춰준다.  Angle = 0; //Angle,Size 초기화  Size = 0;  deletingflag = false;  }  InvalidateRect(hwnd, NULL, false);  }  else  {  RECT rect;  rect.left = 10; rect.top = 10.5; rect.right = 236; rect.bottom = 190.5;  InvalidateRect(hwnd, &rect, true);  }  return 0;  } |

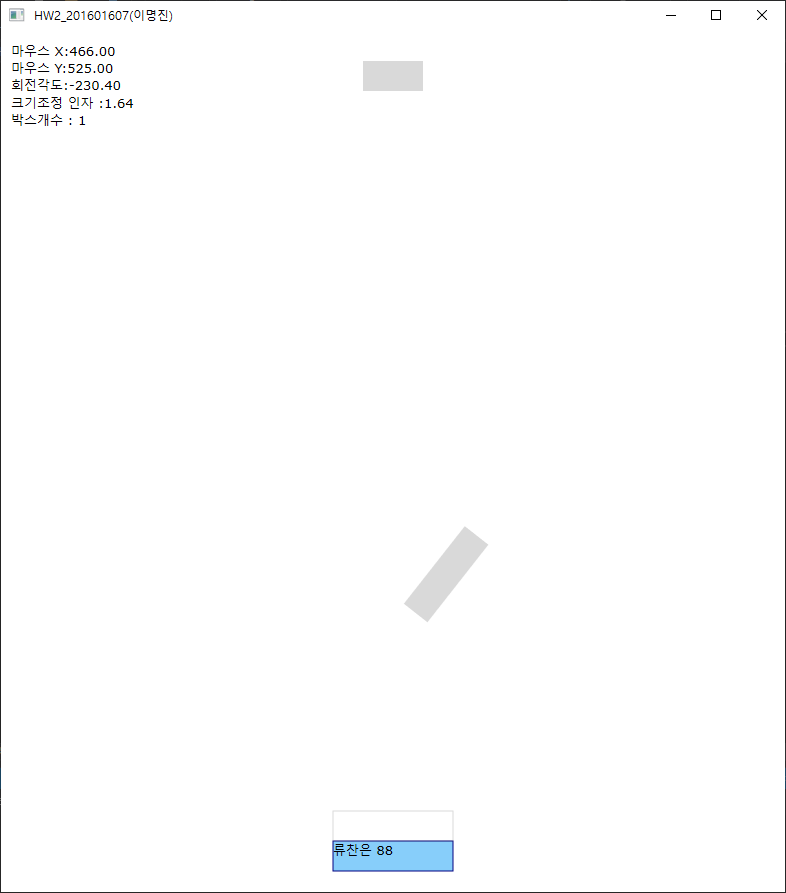
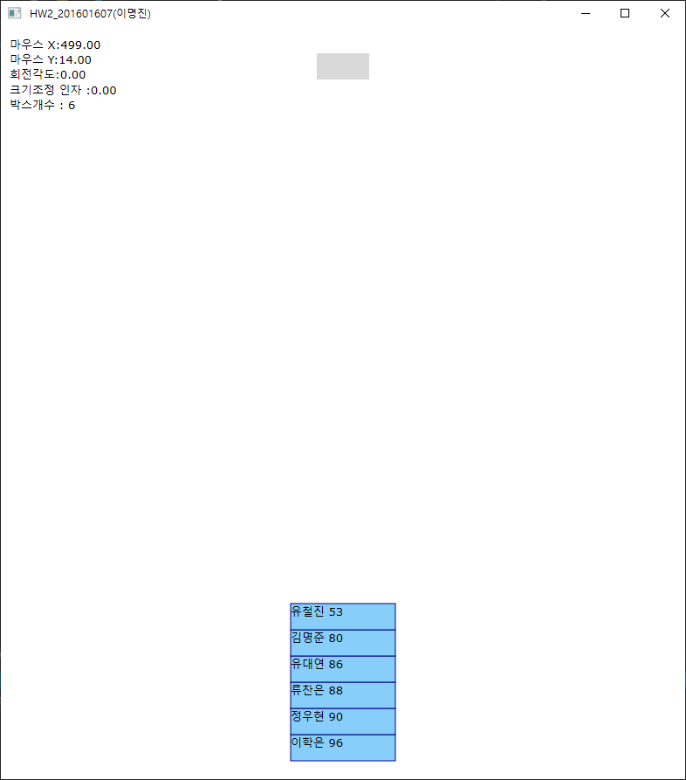
# 3: 동작 결과



- 기본 화면으로 아무런 동작도 하지 않은 상태

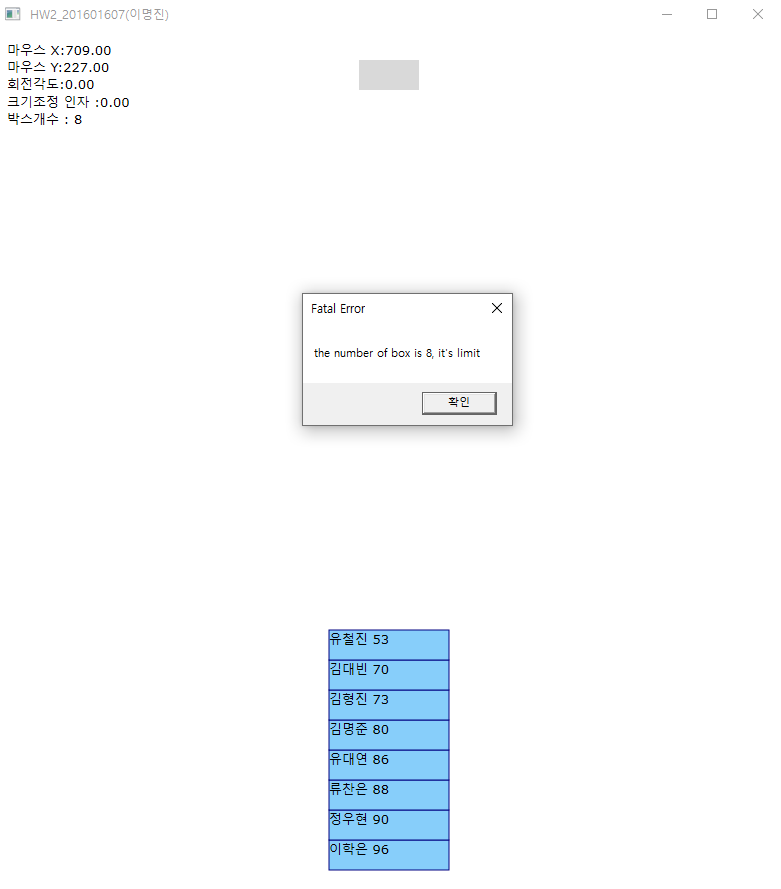


- 드래그 시 박스가 반시계 방향으로 회전한다  
- 마우스가 이동하면서 동시에 좌상단의 값들 또한 변경된다.  
- 아래로 이동할수록 아래의 박스 윤곽과 비슷하게 확대된다.

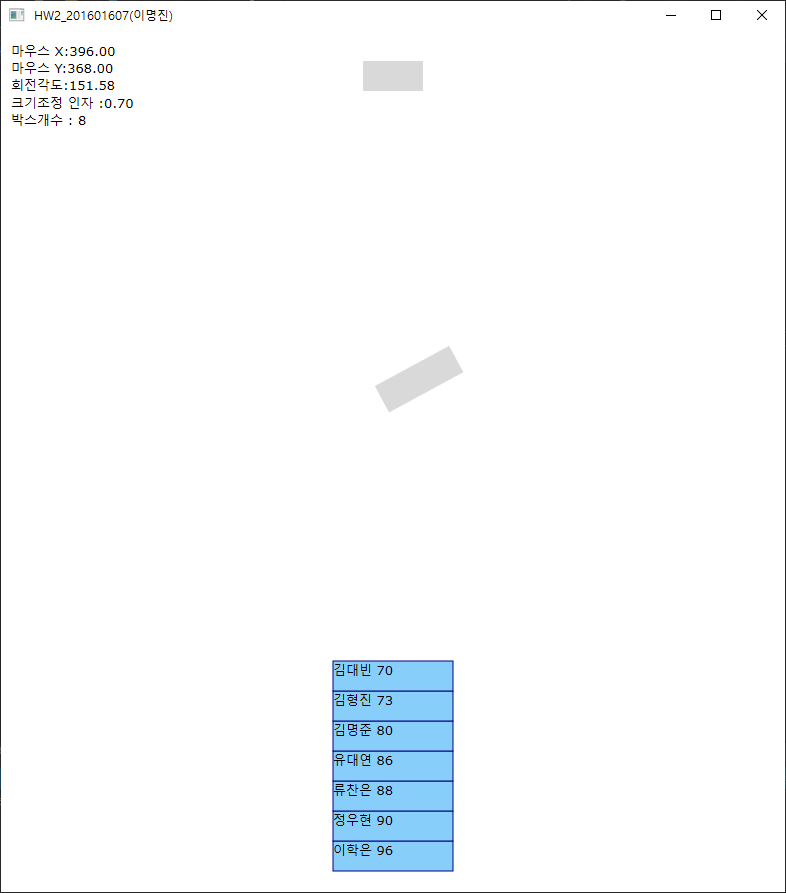
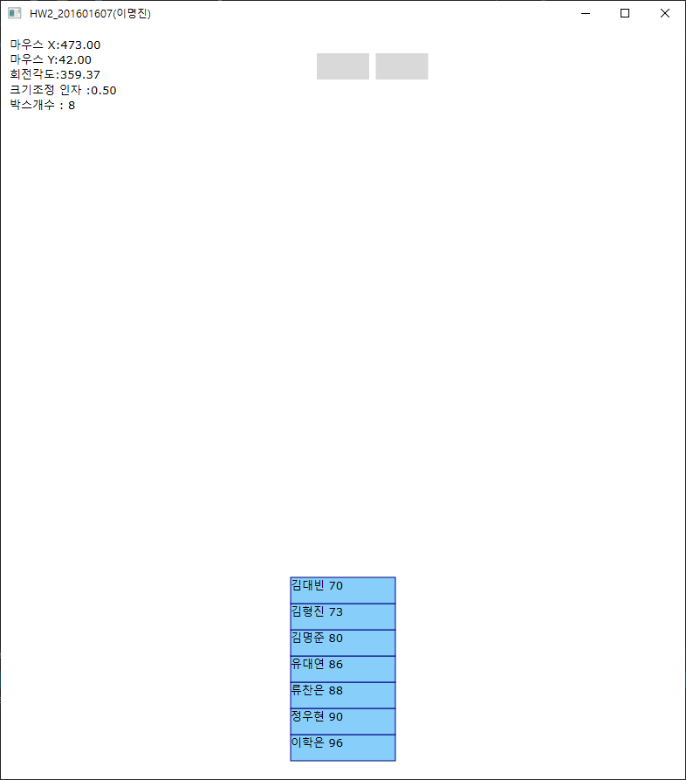


- vector의 내용은 별도의 입력 없이 자동적으로 생성되고 삽입된다.  
- 박스를 쌓음에 따라 정렬이 수행되고 박스 내의 내용도 정렬된다  
- 따라서 가장 낮은 점수의 학생 정보가 가장 위의 박스에 위치하게 된다.

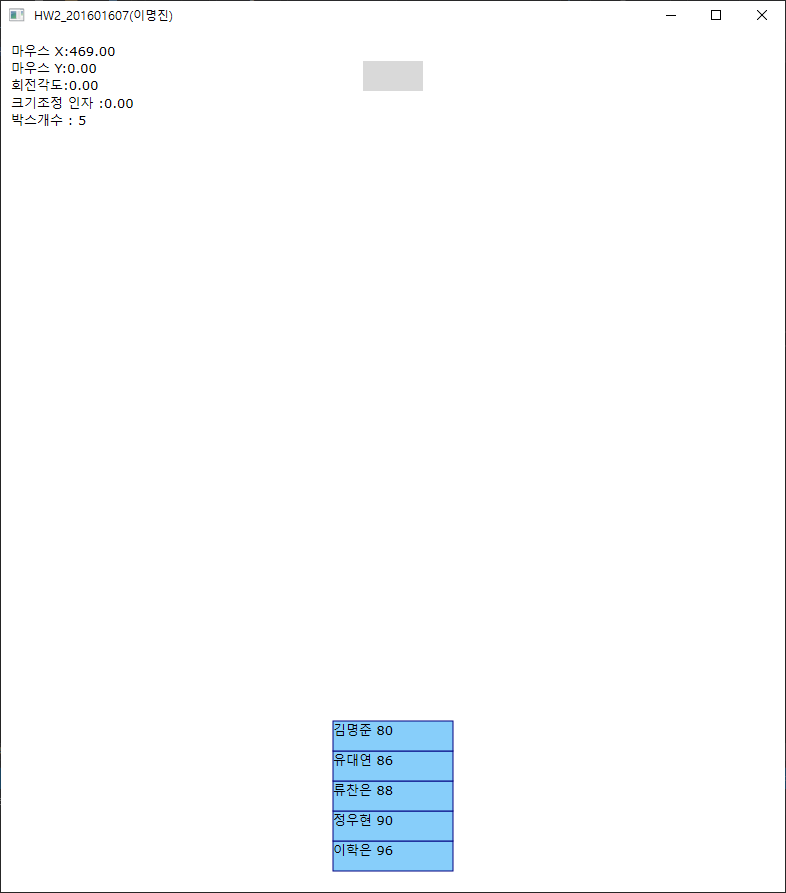




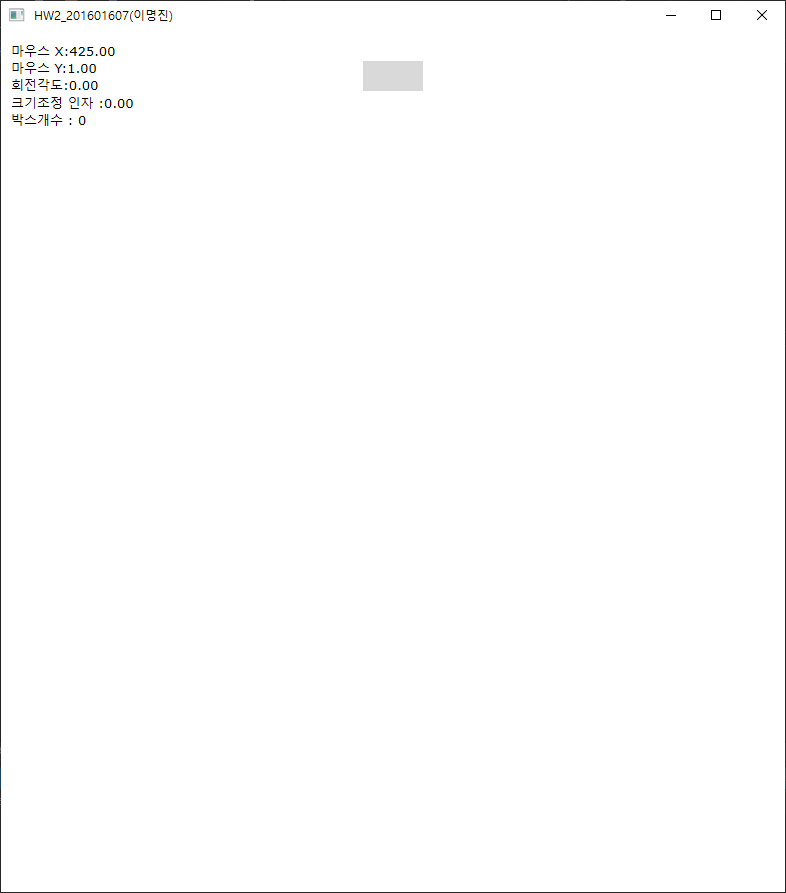
- 개수에 대한 제한을 확인하지 못하여 HW1처럼 최대 vector의 개수를 8로 설정하였다.  
- 최대 개수에 도달 후 추가 동작을 수행하면 경고를 출력한다.



- 제거의 경우 시계 방향으로 회전하며 제거된다.



- 제거도 정상적으로 수행된다.

  
- 완전히 제거한 후

# 4: 참고 자료

- C++와 DirectX로 배우는 2D 게임 프로그래밍 기초 – 박종승 저 (강의 교과서)

- 201501438\_박지현 학우의 힌트 코드

# 5: 전체 코드

|  |
| --- |
| stdafx.h |
| #pragma once  // Windows Header Files:  #include <windows.h>  // C RunTime Header Files  #include <stdlib.h>  #include <malloc.h>  #include <memory.h>  #include <tchar.h>  #include <math.h>  #include <time.h>  // DX Header Files:  #include <d2d1.h>  #include <d2d1helper.h>  #include <dwrite.h>  #include <wincodec.h>  #include <string>  // C++ Header  #include <iostream>  #include <stack>  #include <string>  #include <vector>  #include <algorithm>  // 자원 안전 반환 매크로.  #define SAFE\_RELEASE(p) { if(p) { (p)->Release(); (p)=NULL; } } |

|  |
| --- |
| stdafx.cpp |
| #include "stdafx.h" |

|  |
| --- |
| StudentID.h |
| #pragma once  #include "stdafx.h"  /\*  학생의 정보와 관련된 Header 파일  \*/  // 학생 정보 구조체  struct student  {  int score;  std::string name;  };  // vector의 정렬을 위한 커스텀 비교 함수  bool compare(student a, student b);  // 학생의 정보를 만들기 위한 함수  std::string rtnName(); //이름 생성 함수  int rtnScore(); //점수 생성 함수 |

|  |
| --- |
| StudentID.cpp |
| #include "StudentID.h"  /\*  학생의 정보와 관련된 CPP 파일  \*/  // --------------- 선언부 --------------------  // 전역 변수  //std::vector<student> vec; // 학생의 정보를 저장할 vector  // 학생의 이름을 만들때 사용할 배열  std::string familyName[12] =  {  "김", "이", "박", "최",  "남", "정", "한", "류",  "신", "유", "마", "주"  };  std::string firstName[12] =  {  "형", "남", "명", "승",  "학", "의", "찬", "우",  "홍", "철", "규", "대"  };  std::string secondName[12] =  {  "현", "진", "준", "성",  "기", "지", "은", "훈",  "연", "원", "빈", "선"  };  // --------------- 구현부 --------------------  // 학생의 정보를 만들기 위한 함수  std::string rtnName() //이름 생성 함수  {  srand((unsigned int)time(NULL));  std::string studentName = familyName[rand() % 12] + firstName[rand() % 12] + secondName[rand() % 12];  return studentName;  }  int rtnScore() //점수 생성 함수  {  srand((unsigned int)time(NULL));  int score = rand() % 50 + 50;  return score;  }  // vector를 정렬하기 위한 커스텀 비교함수  bool compare(student a, student b)  {  return a.score > b.score;  } |

|  |
| --- |
| HW2.h |
| #pragma once  #include "stdafx.h"  #include "StudentID.h"  using namespace std;  class DemoApp  {  HWND m\_hwnd;  ID2D1Factory\* m\_pD2DFactory;  ID2D1HwndRenderTarget\* m\_pRenderTarget;  ID2D1StrokeStyle\* m\_pStrokeStyleDash;  ID2D1SolidColorBrush\* m\_pOriginalShapeBrush;  ID2D1SolidColorBrush\* m\_boundaryBrush;  ID2D1SolidColorBrush\* m\_RectangleBrush;  ID2D1SolidColorBrush\* m\_pFillBrush;  ID2D1SolidColorBrush\* m\_pTextBrush;  //DWrite  IDWriteFactory\* m\_pDWriteFactory;  IDWriteTextFormat\* m\_pTextFormat;  public:  //생성,소멸자  DemoApp();  ~DemoApp();  HRESULT Initialize(HINSTANCE hInstance);  void RunMessageLoop();  HRESULT CreateDeviceIndependentResources();  HRESULT CreateDeviceResources();  HRESULT OnRender();  void DiscardDeviceResources();  void OnResize(UINT width, UINT height);  void drawInit();  void drawTempInsertingRectangle();  void drawTempdeletingRectangle();  static LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);  }; |

|  |
| --- |
| HW2.cpp |
| #define \_CRT\_SECURE\_NO\_WARNINGS // 경고를 무시하기 위해 사용  #include "HW2.h"  // --------------- 선언부 --------------------  // 변수  int maxsize = 8; // 최대 개수  int gap = 30; // 상단과 하단의 거리 차이  int centerx, centery = 0; // 전체 스크린의 중심x,y좌표  int Vectorsize = 0; // 현재 학생정보 vector의 사이즈  float Angle = 0; // 각도 조절 인자  float Size = 0; // 사이즈 조절 인자  D2D\_POINT\_2F lastdown; // 가장 최근에 눌려진 좌표  D2D\_POINT\_2F current; // 현재 마우스 좌표  D2D1\_RECT\_F insertrectangle;  // 플래그 변수  boolean inserttingflag = false; // 삽입 중  boolean inserttedflag = false; // 삽입 완료  boolean deletingflag = false; // 삭제 중  boolean deletedflag = false; // 삭제 완료  vector <student> vec; // 학생 데이터들이 담길 vector  // 함수  void dataPush(); // 데이터를 벡터에 삽입  D2D1\_RECT\_F getNextRectangle(int Vectorsize); // 다음 사각형의 정보를 반환할 함수  // --------------- 구현부 --------------------  // 생성자: 클래스 변수 초기화  DemoApp::DemoApp() :  m\_hwnd(NULL),  m\_pD2DFactory(NULL),  m\_pRenderTarget(NULL),  m\_pStrokeStyleDash(NULL),  m\_pOriginalShapeBrush(NULL),  m\_boundaryBrush(NULL),  m\_RectangleBrush(NULL),  m\_pFillBrush(NULL),  m\_pTextBrush(NULL),  m\_pDWriteFactory(NULL),  m\_pTextFormat(NULL)  {  }  // 소멸자: 모든 인터페이스 객체를 반납  DemoApp::~DemoApp()  {  SAFE\_RELEASE(m\_pD2DFactory);  SAFE\_RELEASE(m\_pRenderTarget);  SAFE\_RELEASE(m\_pStrokeStyleDash);  SAFE\_RELEASE(m\_pOriginalShapeBrush);  SAFE\_RELEASE(m\_boundaryBrush);  SAFE\_RELEASE(m\_RectangleBrush);  SAFE\_RELEASE(m\_pFillBrush);  SAFE\_RELEASE(m\_pTextBrush);  SAFE\_RELEASE(m\_pDWriteFactory);  SAFE\_RELEASE(m\_pTextFormat);  }  // 메시지 루프 함수: 윈도우 프로시저로 메시지를 보내는 역할  void DemoApp::RunMessageLoop()  {  MSG msg;  while (GetMessage(&msg, NULL, 0, 0))  {  TranslateMessage(&msg);  DispatchMessage(&msg);  // 윈도우에서 오는 메시지를 윈도우 프로시저로 보내는 과정 속에서 반복 호출  }  }  // 초기화 함수  HRESULT DemoApp::Initialize(HINSTANCE hInstance)  {  HRESULT hr;  hr = CreateDeviceIndependentResources(); // 장치 독립적 자원 생성  if (SUCCEEDED(hr))  {  // 자원 생성에 성공한 경우  // RegisterClassEx, CreateWindow, ShowWindow, UpdateWindow를 순차적으로 호출한다.  WNDCLASSEX wcex = { sizeof(WNDCLASSEX) };  wcex.style = CS\_HREDRAW | CS\_VREDRAW;  wcex.lpfnWndProc = DemoApp::WndProc;  wcex.cbClsExtra = 0;  wcex.cbWndExtra = sizeof(LONG\_PTR);  wcex.hInstance = hInstance;  wcex.hbrBackground = NULL;  wcex.lpszMenuName = NULL;  wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);  wcex.lpszClassName = L"HW2\_201601607(이명진)";  RegisterClassEx(&wcex);  m\_hwnd = CreateWindow  (  L"HW2\_201601607(이명진)", L" HW2\_201601607(이명진)",  WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT,  800, 900, NULL, NULL, hInstance, this  );  hr = m\_hwnd ? S\_OK : E\_FAIL;  if (SUCCEEDED(hr))  {  ShowWindow(m\_hwnd, SW\_SHOWNORMAL);  UpdateWindow(m\_hwnd);  }  }  return hr;  }  // 진입점 함수는 Entry.CPP로 분리함  // 장치 독립적 자원을 생성하는 함수  // 팩토리 객체를 생성함  HRESULT DemoApp::CreateDeviceIndependentResources()  {  HRESULT hr = S\_OK;  hr = D2D1CreateFactory(D2D1\_FACTORY\_TYPE\_SINGLE\_THREADED, &m\_pD2DFactory);  if (SUCCEEDED(hr))  {  hr = DWriteCreateFactory(  DWRITE\_FACTORY\_TYPE\_SHARED,  \_\_uuidof(IDWriteFactory),  reinterpret\_cast<IUnknown\*\*>(&m\_pDWriteFactory)  );  }  if (SUCCEEDED(hr))  {  hr = m\_pDWriteFactory->CreateTextFormat  (  L"Verdana", NULL, DWRITE\_FONT\_WEIGHT\_REGULAR, DWRITE\_FONT\_STYLE\_NORMAL, DWRITE\_FONT\_STRETCH\_NORMAL,  13.0f, L"en-us", &m\_pTextFormat  );  }  if (SUCCEEDED(hr))  {  // 점선 획 스타일 객체를 생성함.  float dashes[] = { 10.0f, 2.0f };  hr = m\_pD2DFactory->CreateStrokeStyle(D2D1::StrokeStyleProperties(  D2D1\_CAP\_STYLE\_FLAT, D2D1\_CAP\_STYLE\_FLAT, D2D1\_CAP\_STYLE\_FLAT, D2D1\_LINE\_JOIN\_MITER,  10.0f, D2D1\_DASH\_STYLE\_CUSTOM, 0.0f),  dashes, ARRAYSIZE(dashes), &m\_pStrokeStyleDash  );  }  return hr;  }  // 장치 의존적 자원을 생성하는 함수  HRESULT DemoApp::CreateDeviceResources()  {  HRESULT hr = S\_OK;  if (!m\_pRenderTarget)  {  RECT rc;  GetClientRect(m\_hwnd, &rc);  D2D1\_SIZE\_U size = D2D1::SizeU(rc.right - rc.left, rc.bottom - rc.top);  hr = m\_pD2DFactory->CreateHwndRenderTarget  (  D2D1::RenderTargetProperties(),  D2D1::HwndRenderTargetProperties(m\_hwnd, size),  &m\_pRenderTarget  );  if (SUCCEEDED(hr))  {  // 변환 전 모양의 외곽선을 위한 붓을 생성.  hr = m\_pRenderTarget->CreateSolidColorBrush(D2D1::ColorF(D2D1::ColorF::Gray), &m\_pOriginalShapeBrush);  }  if (SUCCEEDED(hr))  {  // 변환 후 영역을 채우는 용도의 붓을 생성.  hr = m\_pRenderTarget->CreateSolidColorBrush(D2D1::ColorF(D2D1::ColorF::White), &m\_pFillBrush);  }  if (SUCCEEDED(hr))  {  // 텍스트를 쓰기 위한 붓을 생성.  hr = m\_pRenderTarget->CreateSolidColorBrush(D2D1::ColorF(D2D1::ColorF::Black), &m\_pTextBrush);  }  if (SUCCEEDED(hr))  {  // 변환 후 모양의 외곽선을 위한 붓을 생성.  hr = m\_pRenderTarget->CreateSolidColorBrush(D2D1::ColorF(D2D1::ColorF::Navy), &m\_boundaryBrush);  }  if (SUCCEEDED(hr))  {  // 변환 후 모양의 외곽선을 위한 붓을 생성.  hr = m\_pRenderTarget->CreateSolidColorBrush(D2D1::ColorF(D2D1::ColorF::LightSkyBlue), &m\_RectangleBrush);  }  }  return hr;  }  // 장치 의존적 자원을 반납하는 함수  // CreateDeviceResources() 함수에서 생성한 모든 자원을 반납  void DemoApp::DiscardDeviceResources()  {  SAFE\_RELEASE(m\_pRenderTarget);  SAFE\_RELEASE(m\_boundaryBrush);  SAFE\_RELEASE(m\_pFillBrush);  SAFE\_RELEASE(m\_pOriginalShapeBrush);  SAFE\_RELEASE(m\_pTextBrush);  }  //윈도우 프로시저 함수  // 내용을 그리는 함수  LRESULT CALLBACK DemoApp::WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)  {  if (message == WM\_CREATE)  {  LPCREATESTRUCT pcs = (LPCREATESTRUCT)lParam;  DemoApp\* pDemoApp = (DemoApp\*)pcs->lpCreateParams;  SetWindowLongPtrW(hwnd, GWLP\_USERDATA, PtrToUlong(pDemoApp));  /\* 클래스 객체의 포인터를 접근하는 기능의 구현을 위한 함수  부가적인 윈도우 메모리의 해당 오프셋에 주어진 값을 저장함  - 1번 인자: 해당 윈도우  - 2번 인자: 오프셋  \*/  return 1;  }  DemoApp\* pDemoApp = reinterpret\_cast<DemoApp\*>(static\_cast<LONG\_PTR>(GetWindowLongPtrW(hwnd, GWLP\_USERDATA)));  /\*  GetWindowLongPtrW(): 부가적으로 윈도우 메모리의 값을 얻는 함수  \*/    // 발생되는 메시지들에 대해 의도하는 작업을 수행함  if (pDemoApp)  {  switch (message)  {  case WM\_SIZE:  {  UINT width = LOWORD(lParam);  UINT height = HIWORD(lParam);  pDemoApp->OnResize(width, height);  }  return 0;  case WM\_DISPLAYCHANGE:  {  InvalidateRect(hwnd, NULL, FALSE);  }  return 0;  // 좌클릭 해제에 대한 동작  case WM\_LBUTTONUP:  {  // 삽입, 삭제 동작을 표시하는 flag를 모두 초기화  inserttingflag = false;  deletingflag = false;  // 각도, 크기 초기화  Angle = 0;  Size = 0;  break;  }  // 좌클릭에 대한 동작  case WM\_LBUTTONDOWN:  {  if (LOWORD(lParam) >= insertrectangle.left && LOWORD(lParam) <= insertrectangle.right  && HIWORD(lParam) >= insertrectangle.top && HIWORD(lParam) <= insertrectangle.bottom) { // 만약 상단의 가운데 사각형이 눌려졌으면  inserttingflag = true; // insertting모드를 켜준다.  lastdown.x = LOWORD(lParam);  lastdown.y = HIWORD(lParam);  }  else if (vec.size() == 0)  break;  else  {  D2D1\_RECT\_F lastRectangle = getNextRectangle(Vectorsize - 1); // 이전의 vector를 가리킨 경우라면  if (LOWORD(lParam) >= lastRectangle.left && LOWORD(lParam) <= lastRectangle.right  && HIWORD(lParam) >= lastRectangle.top && HIWORD(lParam) <= lastRectangle.bottom)  {  // deleting모드를 켜준다.  deletingflag = true;  lastdown.x = LOWORD(lParam);  lastdown.y = HIWORD(lParam);  }  }  break;  }  case WM\_PAINT:  {  pDemoApp->OnRender();  ValidateRect(hwnd, NULL);  return 0;  }  // 마우스의 움직임에 따른 동작  case WM\_MOUSEMOVE:  {  // 현재 마우스의 윈도우 상 좌표  current.x = LOWORD(lParam);  current.y = HIWORD(lParam);  // 삽입 동작이라면  if (inserttingflag == true)  {  // 만약 maxsize를 초과한다면  if (Vectorsize >= maxsize)  {  inserttingflag = false; // inserttingflag를 false로 처리해준다.  ::MessageBox(0, L"the number of box is 8, it's limit", L"Fatal Error", MB\_OK);//메세지를 띄워준다.  ::MessageBeep(MB\_OK);  break;  }  D2D1\_RECT\_F currentVector = getNextRectangle(Vectorsize);  if (LOWORD(lParam) >= currentVector.left &&  LOWORD(lParam) <= currentVector.right &&  HIWORD(lParam) >= currentVector.top &&  HIWORD(lParam) <= currentVector.bottom)  {  // inserttingflag가 true이고 들어와야할 스택에 마우스 커서가 들어왔으면  inserttedflag = true; // inserttedflag를 true로 맞춰준다.  Angle = 0; //Angle ,Size 초기화  Size = 0;  inserttingflag = false;  }  InvalidateRect(hwnd, NULL, false);  }  // 제거 동작이라면  else if (deletingflag == true)  {  if (LOWORD(lParam) >= insertrectangle.left && LOWORD(lParam) <= insertrectangle.right  && HIWORD(lParam) >= insertrectangle.top && HIWORD(lParam) <= insertrectangle.bottom)  {  // deletingflag가 true이고 insertingrectangle에 마우스 커서가 들어왔으면  deletedflag = true; // deletedflag를 true로 맞춰준다.  Angle = 0; //Angle,Size 초기화  Size = 0;  deletingflag = false;  }  InvalidateRect(hwnd, NULL, false);  }  else  {  RECT rect;  rect.left = 10; rect.top = 10.5; rect.right = 236; rect.bottom = 190.5;  InvalidateRect(hwnd, &rect, true);  }  return 0;  }  case WM\_DESTROY:  {  PostQuitMessage(0);  }  return 1;  }  }  return DefWindowProc(hwnd, message, wParam, lParam);  }  HRESULT DemoApp::OnRender()  {  HRESULT hr = CreateDeviceResources();  /\*  렌더링 시 항상 호출되는 함수  - 최초 호출에는 장치 의존적 자원들을 생성  - 이후에는 렌더 타겟이 유효, 동작하지 않음  \*/  if (SUCCEEDED(hr)) // 렌더 타겟이 유요한 경우 실행됨  {  // 그리기를 시작함.  m\_pRenderTarget->BeginDraw();  // 렌더타겟 변환을 항등 변환으로 리셋함.  m\_pRenderTarget->SetTransform(D2D1::Matrix3x2F::Identity());  // 렌더타겟 내용을 클리어함.  m\_pRenderTarget->Clear(D2D1::ColorF(D2D1::ColorF::White));  D2D1\_SIZE\_F rtSize = m\_pRenderTarget->GetSize();  // deletingflag 또는 deletedflag가 true이면 잠시 vectorsize를 줄여준다.  if (deletingflag == true || deletedflag == true)  {  Vectorsize--;  }  // 커스텀 비교자를 사용해서 student 구조체를 사용하는 vector를 정렬한다  sort(vec.begin(), vec.end(), compare);  // 정렬된 내용을 차례로 출력  // vector의 내용이 변경된다면 그 내용이 반영되어 출력된다  for (int i = 0; i < Vectorsize; i++)  {  m\_pRenderTarget->FillRectangle(getNextRectangle(i), m\_RectangleBrush); // 그다음 그려질 스택 위치를 얻어내서 사각형을 그려준다.  const char\* studentName = vec[i].name.c\_str(); // v[i].name을 const char\*형으로 바꾼다.    TCHAR temp[15]; // const char\*을 TCHAR형으로 바꿔야 하기 때문에 temp변수를 선언한다.  memset(temp, 0, sizeof(temp)); // 초기화  MultiByteToWideChar(CP\_ACP, MB\_COMPOSITE, studentName, -1, temp, 15);    static WCHAR studentInfo[100]; // name과 점수를 한번에 넣어아 햐므로 WCHAR 배열형 변수를 선언한다.  swprintf\_s(studentInfo, L"%s %d\n", temp, vec[i].score);  m\_pRenderTarget->DrawText(studentInfo, wcslen(studentInfo), m\_pTextFormat, getNextRectangle(i), m\_pTextBrush); // Text를 그려준다.  m\_pRenderTarget->DrawRectangle(getNextRectangle(i), m\_boundaryBrush); // 상자 테두리를 그려준다.  }  // 줄어든 VectorSize를 증가  if (deletingflag == true || deletedflag == true)  {  Vectorsize++;  }  drawInit();  // 만약 삽입하고 있는과정이라면 드래그하는 동안의 상자를 그려준다.  if (inserttingflag == true)  {  drawTempInsertingRectangle();  m\_pRenderTarget->SetTransform(D2D1::Matrix3x2F::Identity());  m\_pRenderTarget->DrawRectangle(getNextRectangle(Vectorsize), m\_pOriginalShapeBrush);  // 상자 위치를 표시해준다.  }  m\_pRenderTarget->SetTransform(D2D1::Matrix3x2F::Identity());  if (deletingflag == true)  {  drawTempdeletingRectangle();  }  // 삽입 완료 시  if (inserttedflag == true)  {  Vectorsize++; // vectorsize 증가  dataPush(); // 벡터에 data를 넣어준다.  inserttedflag = false;  }  // 삭제 완료 시  if (deletedflag == true)  {  Vectorsize--;  vec.pop\_back(); // 벡터에서 값을 빼온다.  deletedflag = false;  }  // 그리기 연산들을 제출함.  m\_pRenderTarget->EndDraw();  }  if (hr == D2DERR\_RECREATE\_TARGET)  {  hr = S\_OK;  DiscardDeviceResources();  // 자원 반환  }  return hr;  }  // 사각형을 그릴 부분  void DemoApp::drawTempInsertingRectangle()  {  // 이동 조절  D2D1\_MATRIX\_3X2\_F translation = D2D1::Matrix3x2F::Translation(current.x - lastdown.x, current.y - lastdown.y);  // 최근 눌려진 좌표에서 현재 좌표까지 거리만큼 이동시켜준다.  Size = ((current.y - lastdown.y) / (getNextRectangle(Vectorsize).top - 30)) + 1;  // 본래 크기에 추가적인 크기를 더해야 하므로 +1 을 해줘야  // 목표 지점까지 얼마만큼 왔는지 비율을 구해서 Size를 구해줍니다.    // 스케일 조절  D2D1\_MATRIX\_3X2\_F scale = D2D1::Matrix3x2F::Scale(D2D1::Size(Size, 1.0f), D2D1::Point2F(insertrectangle.left, insertrectangle.top));  // 회전 조절  Angle = (current.y - lastdown.y) / (getNextRectangle(Vectorsize).top - 30) \* 360 \* -1;  D2D1\_MATRIX\_3X2\_F rotation = D2D1::Matrix3x2F::Rotation(Angle, D2D1::Point2F((insertrectangle.right + insertrectangle.left) / 2, (insertrectangle.top + insertrectangle.bottom) / 2));    // 조절을 적용  m\_pRenderTarget->SetTransform(scale \* rotation \* translation); // scale->rotation->translation 순  m\_pRenderTarget->FillRectangle(insertrectangle, m\_pOriginalShapeBrush); //insertrectangled을 변환  }  void DemoApp::drawTempdeletingRectangle()  {  Vectorsize--;  D2D1\_MATRIX\_3X2\_F translation = D2D1::Matrix3x2F::Translation(current.x - lastdown.x, current.y - lastdown.y); // 최근 눌려진 좌표에서 현재 좌표까지 거리만큼 이동시켜준다.    if (current.y <= lastdown.y) // 만약 위로 드래깅 한다면  {  Size = (getNextRectangle(Vectorsize).top - 30) / ((lastdown.y - current.y) + (getNextRectangle(Vectorsize).top - 30));  // 이 경우에는 가장 최근의 vector에서 축소되는 것이므로 가장 최근의 vector의 위치를 읽어와야 한다.  // 목표 지점까지 얼마만큼 왔는지 비율을 구해서 Size를 구해줍니다.  }  else //만약 아래로 드래깅 한다면  {  Size = ((current.y - lastdown.y) / (getNextRectangle(Vectorsize).top - 30)) + 1;  // 삽입 할때와 동일한 공식을 적용  }    D2D1\_MATRIX\_3X2\_F scale = D2D1::Matrix3x2F::Scale(D2D1::Size(Size, 1.0f), D2D1::Point2F(lastdown.x, lastdown.y));  // 읽어온 값을 기준으로 확대, 축소 동작을 수행    Angle = (current.y - lastdown.y) / (getNextRectangle(Vectorsize).top - 30) \* 360 \* -1;  D2D1\_MATRIX\_3X2\_F rotation = D2D1::Matrix3x2F::Rotation(Angle, D2D1::Point2F(lastdown.x, lastdown.y));  m\_pRenderTarget->SetTransform(scale \* rotation \* translation);  m\_pRenderTarget->FillRectangle(getNextRectangle(Vectorsize), m\_pOriginalShapeBrush); // 최근 입력된 박스를 기준으로 변환하여 그려줍니다.  Vectorsize++;  }  void DemoApp::drawInit()  {  m\_pOriginalShapeBrush->SetOpacity(0.3); // brush에 투명도를 조정  D2D1\_SIZE\_F rtSize = m\_pRenderTarget->GetSize(); // rendertarget의 size를 이용하여  centerx = rtSize.width / 2; //전체 스크린의 중앙 점을 구해낸다.  centery = rtSize.height / 2;  // 렌더타겟 변환을 항등 변환으로 리셋함. 사각형을 그림.  m\_pRenderTarget->SetTransform(D2D1::Matrix3x2F::Identity());  insertrectangle = D2D1::Rect(centerx - 30, gap, centerx + 30, gap + 30); // 가운데 상단에  m\_pRenderTarget->FillRectangle(insertrectangle, m\_pOriginalShapeBrush); // 삽입을 위한 조그만한 사각형을 그려줌  // 캡션 텍스트를 표시함.  m\_pRenderTarget->SetTransform(D2D1::Matrix3x2F::Identity());  WCHAR currentState[100];  swprintf\_s(currentState, L"마우스 X:%.2f\n마우스 Y:%.2f\n회전각도:%.2f\n크기조정 인자 :%.2f\n박스개수 : %d\n", current.x, current.y, Angle, Size, vec.size());  m\_pRenderTarget->DrawText(currentState, wcslen(currentState), m\_pTextFormat, D2D1::RectF(10.0f, 10.5f, 236.0f, 190.5f), m\_pTextBrush);  }  void DemoApp::OnResize(UINT width, UINT height)  {  if (m\_pRenderTarget)  {  m\_pRenderTarget->Resize(D2D1::SizeU(width, height));  }  }  // 다음에 그려질 사각형 좌표를 얻는 함수  D2D1\_RECT\_F getNextRectangle(int Vectorsize)  {  float top = 870 - 30 \* (Vectorsize + 2);  return D2D1::RectF(centerx - 60, top, centerx + 60, top + 30);  // 중심을 기준으로 좌-우로 120  // 높이가 30  // 1 : 4 비율이 나온다  }  // 데이터를 vector에 삽입  void dataPush()  {  student tmp = { rtnScore(), rtnName() };  vec.push\_back(tmp);  // 임시로 student 구조체 변수를 하나 생성한 후 vector에 넣는 동작을 수행한다  } |

|  |
| --- |
| Entry.cpp – 진입점 함수 |
| #define \_CRT\_SECURE\_NO\_WARNINGS  #include "stdafx.h" // 미리 컴파일된 헤더  #include "HW2.h" // D2D 헤더  //진입점 함수  int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, LPSTR, int)  {  if (SUCCEEDED(CoInitialize(NULL)))  {  {  DemoApp app;  if (SUCCEEDED(app.Initialize(hInstance)))  {  app.RunMessageLoop();  }  }  CoUninitialize();  }  return 0;  }  /\*  WINAPI WinMain(): 진입점 함수  DemoApp 클래스의 인스턴스 생성  - 클래스 초기화 함수 Initialize()  - 메시지 루프 함수 RunMessageLoop()  스레드에서 COM 라이브러리 사용을 위한 함수  - CoInitialize(): 스레드가 COM 라이브러리를 사용을 위한 로딩과 초기화 수행  - CoUninitialize(): 사용 종료 후 스레드에서 COM 라이브러리 언로드, 자원 반납  \*/ |