

1. 목 적 : 미로 그리기

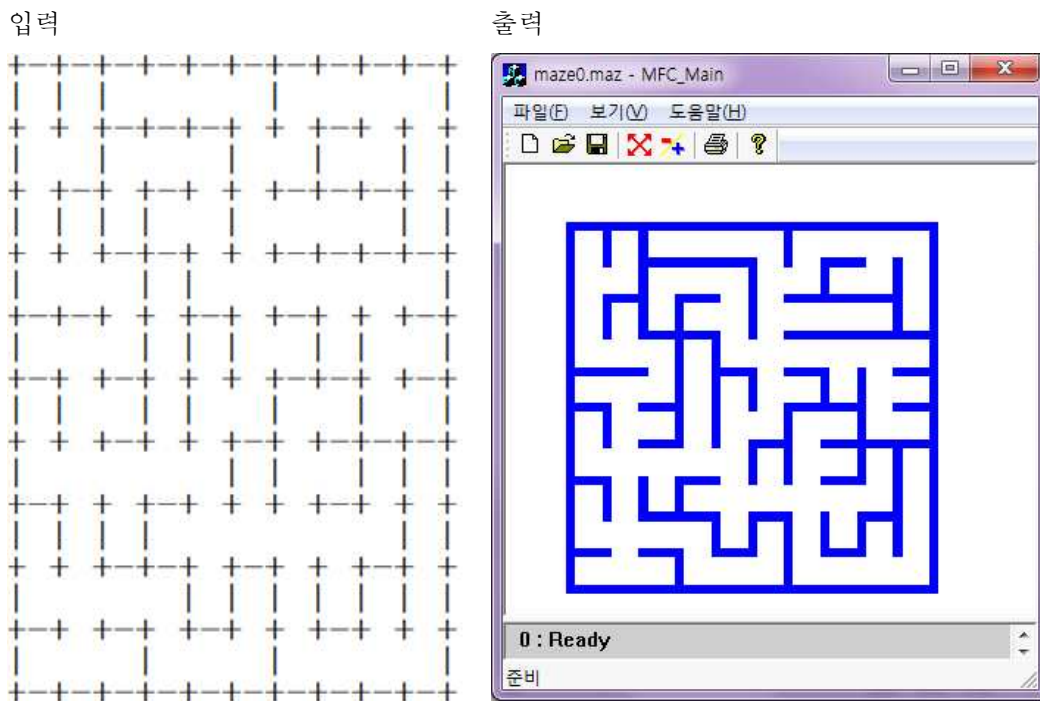
본 실험은 1주차 미로 만들기 프로젝트의 결과물인 ~.maz 파일을 읽어서 이를 윈도우 화면상에 표시함을 목적으로 한다. 실험목적인 GUI(Graphical User Interface) 구축과 동시에 3주차에서 행할 프로젝트를 위하여 학생들은 창의적이고 효율적인 자료구조를 설계하여야 한다. GUI를 용이하게 설계하기 위하여 MFC(Microsoft Foundation Class)를 기반으로 한 Graphic Toolkit이 주어질 예정이다.

2. 프로그래밍 문제

입력형식: 입력은 text파일로서 1주차 실험에서 만들었던 미로 만들기 프로젝트의 결과를 이용한다. (~.maz 확장자를 가지고 있다.) 텍스트 파일의 “+”는 미로 한 칸의 모서리를 의미하며 “-”는 가로벽, “|”는 세로벽을 “ ”(빈칸)은 방 또는 방과 방사이의 통로를 의미한다.

출력형식: 아래보인 예와 같은 형태의 입력받은 미로를 그림1에 보인 것과 같이 윈도우 창에 표시한다. (아래의 예 참조) 미로의 색이나 폭 등은 자유롭게 디자인한다.

입출력 예:

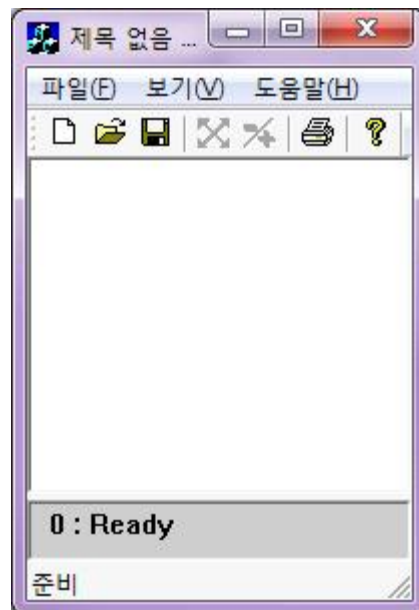


[그림 1] 입력(좌)에 대한 출력(우)예시.


요구사항: 단순히 입력된 정보를 읽어서 화면에 표시할 것이 아니라, DFS(Depth First Search :깊이 우선 탐색), BFS(Breadth First Search :넓이 우선 탐색) 등의 방법으로 미로의 경로를 탐색하기 위해 미로의 정보를 효율적으로 담고 있는 자료구조를 구축하여 입력된 정보를 저장한다. 즉, 프로그램이 미로를 화면에 표시하고 나서도 미로에 대한 정보를 계속 유지하고 있어야 한다. 구축한 자료구조에 입력된 정보를 저장하고 이를 바탕으로 윈도우에 미로를 표시하도록 한다. 또한, 앞으로 설명할 내용을 바탕으로 하여, 파일을 열때, ~.maz 확장자를 기본으로 열도록 설정한다.

윈도우 설계 : 실험 설계가 용이하도록 기본 MFC Graphic Toolkit 프로그램이 주어질 예정이다. 본 프로그램은 MFC Window 프로그래밍이 생소한 학생들이 쉽게 이용할 수 있도록 하여 다른 프로그램을 작성하는데 도움을 주는 것을 목적으로 한다. 이번 실험을 통해 구현하려는 프로그램의 전체적인 화면과 수행순서를 설명한다면 아래와 같다.

1. 초기에 프로그램을 실행시키면 [그림 2]와 같은 화면이 나온다.

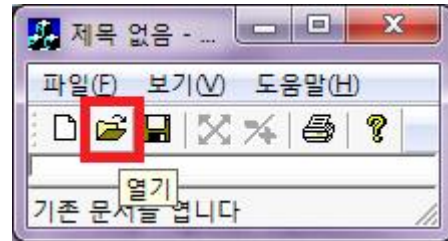


[그림 2] 초기 화면.

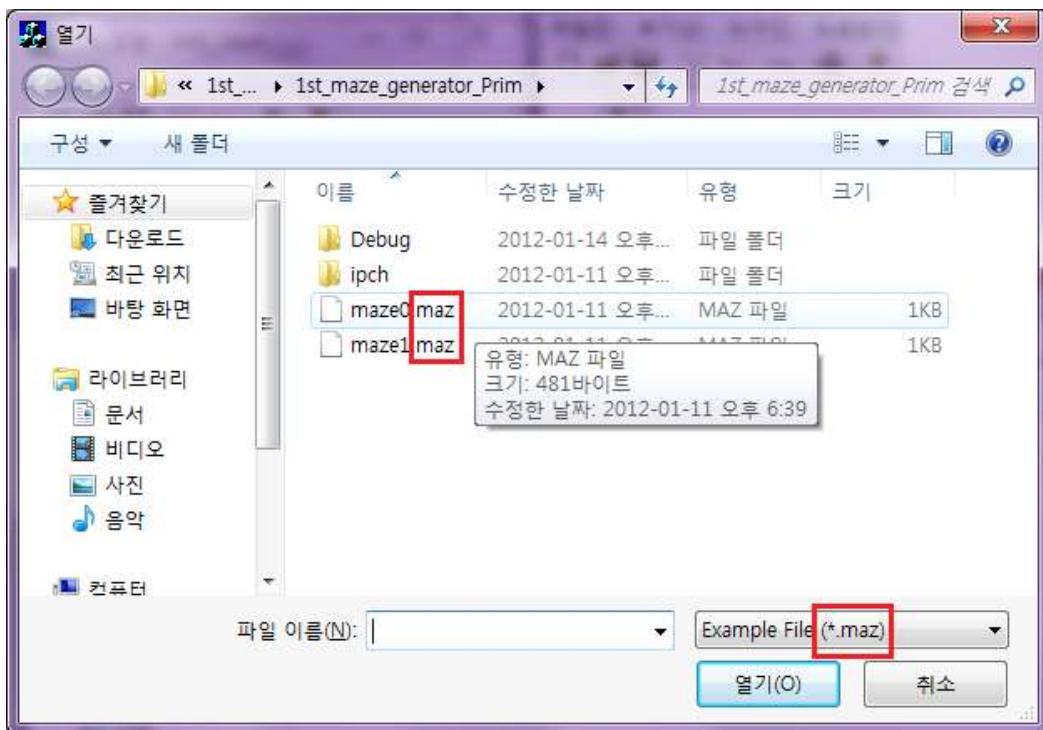
2. 사용자가 메뉴의 [파일]→[열기]([그림 3]), 또는 툴바의 ([그림 4])를 누르면, [그림 5]와 같은 화면이 나온다.



[그림 3] 파일 열기(1).



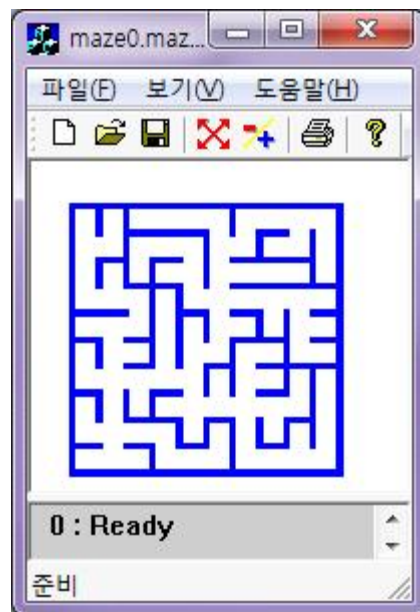
[그림 4] 파일 열기(2).



[그림 5] 파일 열기(3).

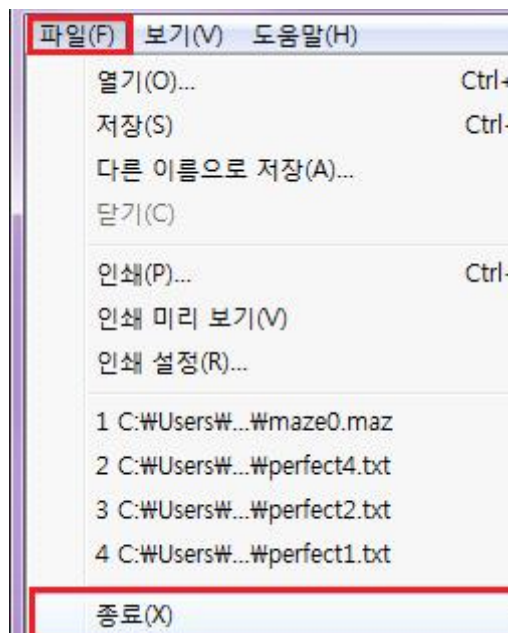
불러오려는 파일의 기본 확장자가 ~.maz로 설정됐음을 확인 할 수 있다(프로젝트 수행원이 처리해야할 요구사항 중 하나이다).

3. 1주차 미로 만들기 프로젝트의 출력인 ~.maz 파일을 열면 [그림 6]과 같이 윈도우에 미로가 그려진다.



[그림 6] 출력.

4. [그림 7]이나 [그림 8]처럼 [파일] → [종료] 버튼이나 윈도우의 오른쪽 상단의 X 아이콘을 누르면 프로그램이 종료된다.



[그림 7] 종료 (1).



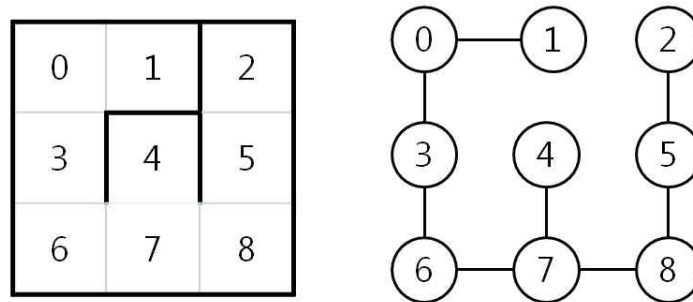
[그림 8] 종료 (2).

3. 실험방법

3-1 문제해결

본 실험에서 해결할 문제는 아니지만, (사실상 이번 실험은 미로 프로젝트 3주차 실험을 수행하기 위한 기초 작업이다). 다음 실험에서는 불러들인 미로를 탈출하는 경로를 찾아 화면에 표시하는 프로그램을 설계할 것이다.

1주차에서 살펴본 미로의 정의에 따라 미로의 인접한 두 방 사이에 벽이 없을 경우 두 방 사이에 경로가 존재한다. 미로의 각 방을 vertex, 인접한 두 방 사이의 경로를 두 vertex 간의 edge로 보면 [그림 9]와 같이 미로는 그래프의 일종으로 볼 수 있다.



[그림 9] 미로의 그래프 변환.

따라서 그래프 이론에 따라 DFS(Depth First Search : 깊이 우선 탐색), BFS(Breadth First Search : 넓이 우선 탐색) 두 방법으로 미로의 경로를 찾아볼 수 있다. 두 이론에 대한 내용을 조사하여 예비보고서에 정리하고, 어떤 자료구조를 사용해야 시간복잡도와 공간복잡도를 최소화할 수 있을지 자신만의 방법을 고안해내도록 한다. DFS와 BFS 방식을 고려하지 않을시 미로 프로젝트 3주차 실험을 진행하는데 있어서 어려움을 겪을 수 있으므로 반드시 자료구조 설계 전에 DFS 및 BFS 방법을 이해하도록 하자. 또한 DFS의 경우 recursive가 아닌 iterative한 방법으로 작성해야 할 것이다. iterative하게 DFS 탐색을 어떻게 구현할 것인지 생각해보자.

3-2. 자료구조 설계

이 프로그램에서는 미로의 각 방과 이웃 방에 대한 관계정보를 저장해야 한다. DFS와 BFS를 수행할 것을 고려하여 어떤 정보가 어떤 방식으로 저장될 것인지 생각해보자. 각 방은 인접한 상하좌우 모든 방과의 벽의 유무에 대한 정보를 가지고 있어야 하는가? 벽이나 이웃 방에 대한 정보는 어떻게 저장할 것인가? 그래프 이론과 연관 지어서도 생각해보자. 물론 이정보를 통해 화면에 미로를 그릴 수도 있어야 한다.

3-3 프로그램 구성

주어진 MFC Toolkit 프로그램의 MFC_Main\User Code directory의 usercode.cpp에 자신의 소스를 작성하면 된다. user code를 구성하고 있는 함수들은 다음과 같다.

- **bool readFile(const char* filename)**

사용자가 파일 열기 dialog에서 선택한 파일 이름이 filename 파라미터로서 전달되면,

그 파일의 내용을 읽고, 메모리에 필요한 형태로 변환하여 저장하는 함수. 파일읽기가 성공하면 true, 실패하면 false를 반환한다. 파일 읽기가 끝나면, 이 함수 내에서 화면 영역이 적절히 설정되어야 한다. (화면에 그림을 그리는 작업은 자동으로 실행된다.)

- **bool writeFile(const char* filename)**

사용자가 불러들인 파일을 기반으로 다른 내용을 저장할 때 필요한 함수이다. 성공하면 true, 실패하면 false를 반환한다. 이번 프로젝트에서는 크게 사용할 일이 없다.

- **void freeMemory()**

readFile이나 그 후 사용자가 작성한 다른 함수에서 할당한 동적 메모리를 해제시켜준다. 현재 보고 있던 파일에서 새로운 파일을 불러올 때 자동적으로 실행된다.

- **void drawMain(CDC *pDC)**

화면에 그림을 그려주는 함수. CMFC_MainView::OnDraw()에서 호출하며, 프로젝트 수행자가 크게 건드릴 필요가 없는 함수이다.

- **static void drawDirect(CDC *pDC)**

화면에 직접 그림을 그리는 방식으로 그림을 그리는 함수.

- **static void drawBuffered()**


버퍼에 화면을 저장했다가 한꺼번에 표시하는 방식으로 그림을 그리는 함수.

보다 자세한 내용은 이어지는 3-4절의 프로그램 편집부분을 참고하도록 하자.

3-4. 프로그램 편집

주어진 MFC Graphic Toolkit Project의 MFC_Main\User Code\usercode.cpp에 다음 단계들을 따라서 프로그램을 편집하자.

Step 1. 파일 열기

- a. User Code\usercode.cpp의 함수 bool readFile(const char *filename)에 사용자의 코드를 삽입한다. 이 함수는 프로그램에서 “파일”→“열기”메뉴를 선택하거나 툴바의 를 클릭하면 실행되는 함수이다. ~.maz 텍스트파일을 읽어 들여 이를 자신이 설계한 자료구조 내에 적절한 형태로 변환하여 저장하는 소스를 작성하자. 이 때 fopen 함수의 파일제목 파라미터에는 readFile의 입력 파라미터인 filename 변수를 전달해주면 알아서 사용자가 연 파일이름이 전달된다.

예)

```
File *fp;  
fp = fopen(filename,"r");
```

- b. 화면에 그림을 그리기 위한 작업으로는 setWindow()함수를 호출한다. setWindow 함수에 대한 설명은 Step 2. 화면에 그림 그리기에 있으니 이를 참고한다.

- c. 파일 읽기가 성공하면 true, 실패하면 false를 반환한다.
- d. File Open dialog의 기본 확장자를 바꾸려면 다음과 같이 한다.
MFC_Main.cpp의 함수 OnFileOpen()에 다음과 같은 코드가 있다.

```
CFileDialog dlg(true, ".dat", NULL, OFN_HIDEREADONLY | OFN_FILEMUSTEXIST,
"Example File (*.dat)|*.dat|All Files (*.*)|*.*||", NULL);
```

위 코드에서 밑줄 친 부분을 원하는 확장자(~.maz)로 바꿔주면 된다.

Step 2. 화면에 그림 그리기

화면에 그림을 그리는 방법에는 두 가지 방법이 있다. 하나는 화면에 직접 그림을 그리는 방법이고 다른 하나는 버퍼를 이용하여 그림을 그리는 방법이다. 첫 번째 방법은 가장 일반적이고 간편한 방법이나, 속도가 느리고 그려지는 과정이 화면에 보이므로 그려질 자료가 많은 경우 그다지 좋은 방법은 아니다. 버퍼를 이용하여 그리는 방법은 화면에 보일 그림들은 미리 버퍼에 그려놓고 한꺼번에 화면에 보여주는 방법으로 화면에 직접 그리는 것보다 속도가 빠르고 화면 갱신이 한 번에 이루어지지만 이를 위해 추가로 메모리를 사용하여야 한다.

위의 방법 중 한 가지를 선택하는 방법은 다음과 같다.

- Visual C++에서 [Project]-[Settings...] 메뉴를 선택한다.
- Project Setting dialog에서 [C/C++]탭을 선택한다.
- [Preprocessor definitions] edit box에 다음 중 하나를 추가한다.
 - GRAPHICS_DIRECT -화면에 직접 그림을 그릴 경우
 - GRAPHICS_BUFFERED -버퍼를 이용하여 그림을 그릴 경우
- [OK]버튼을 누르고 다시 컴파일 한다.

이번 프로젝트에서는 자료구조에 저장된 메모리를 빠르게 화면에 표시하기 위해 buffer를 이용하는 방법을 사용하도록 한다. 그리기 함수 호출 시 모든 좌표는 사용자가 지정한 가상의 physical window상의 좌표를 따라야 한다. physical window란 사용자가 화면에 그림을 그리기 위한 도화지 같은 역할을 하는 객체이다. 이 때 화면상의 출력은 본 프로그램에 의하여 알맞은 window 좌표로 변환된다. 화면에 그림을 그리는 내용의 코드는 User Code\usercode.cpp 내부에 미리 정의 해둔 drawBuffered()함수에 작성한다.

이제, 이 프로젝트를 작성하기 위해 윈도우 영역을 설정하고, 메시지를 표시하고, 도형을 화면에 그리는 등의 역할을 하는 함수들을 알아보겠다.

a. Window 영역 지정하기

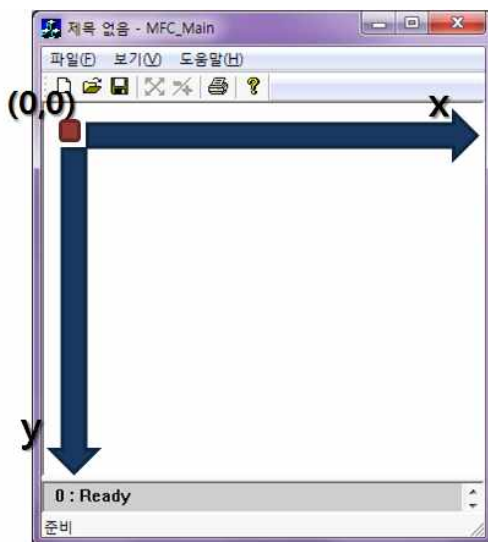
syntax void setWindow(double xorg, double yorg, double xmax, double ymax, int
UpsideDown_Flag)

parameter

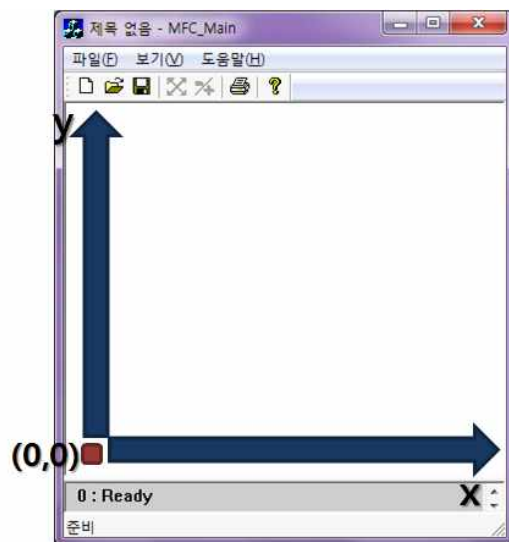
xorg, yorg	Physical Window의 원점의 x, y 좌표 (그림 10, 11 참고)
xmax, ymax	Physical Window의 Maximum x, y 좌표 (그림 10, 11 참고)
UpsideDown_Flag	1 : Window 좌표계 (원점 : 왼쪽 위) 0 : 일반 좌표계 (원점 : 왼쪽 아래) (그림 10, 11 참고)

기능

(xorg, yorg)부터 (xmax, ymax)까지를 physical window의 영역으로 설정한다. (xorg, yorg)는 화면에 표시되는 모든 그림들의 좌표의 최소값보다 작아야 하며, (xmax, ymax)는 화면에 표시되는 모든 그림들의 좌표의 최대값보다 커야 한다. 즉 창의 영역을 결정하는 함수로 봐도 무방하다.



[그림 10] Window 좌표계.



[그림 11] 일반 좌표계.

b. Message 창에 message 출력하기

syntax void showMessage(const char* pszMsg)

parameter

pszMsg	메시지 창에 보일 메시지
--------	---------------

기능

Message 창에 pszMsg(문자열 포인터 변수)를 출력한다.

c. Message 창의 message 지우기

syntax void clearMessage()

기능 Message 창의 메시지를 지운다.

d. Buffer를 이용한 선 그리기

syntax void DrawLine_I(double x1, double y1, double x2, double y2, int nWidth, COLORREF nColor)

parameter

x1, y1	선의 시작점의 좌표
x2, y2	선의 끝점의 좌표
nWidth	선의 두께 지정
nColor	선의 색을 지정한다. 색의 값은 "0x00ggbbrr" (gg, bb, rr 은 각각 green, blue, red 의 강도에 대한 16진수) 형식으로 지정할 수도 있고, Windows Macro 인 "RGB(r,g,b)"를 이용하여 지정할 수도 있다. (r, g, b = 0~255 사이의 정수) COLORREF type에 대한 자세한 설명은 Visual C++ 도움말을 참고한다.

기능

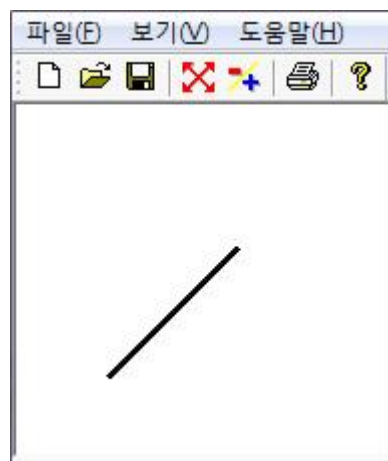
Buffer를 이용하여 (x1, y1)부터 (x2, y2)까지를 잇는 nWidth 두께의 nColor 색상의 선을 그린다.

예)

```
bool readFile(const char* filename){
    setWindow(0,0,5,5,0);
    //일반좌표계로 (0,0)부터 (5,5)까지를 윈도우 영역으로 설정한다.
}

static void drawBuffered(){
    DrawLine_I(1,1,3,3,1,RGB(10,10,10));
    //일반 좌표계로 (1,1)부터 (3,3)까지를 두께가 3인 검은색 (RGB(0,0,0) 선을 그린다.
    //원래 RGB(0,0,0)이 검은색이지만 이 경우 흰색으로 처리 되서 표시 되므로
    //육안으로 비슷해 보이게끔 RGB(10,10,10) 색을 이용해서 그렸다.)
}
```

의 결과는 [그림 12]와 같다.



[그림 12] DrawLine_I 예.

e. Buffer를 이용한 사각형 그리기

syntax void DrawBox_I(double x1, double y1, double x2, double y2, int nWidth, COLORREF nColor)

parameter

x1, y1	사각형 왼쪽 위 꼭지점의 좌표
x2, y2	사각형 오른쪽 아래 꼭지점의 좌표
nWidth	사각형 선의 두께 지정
nColor	사각형 외곽선의 색

기능

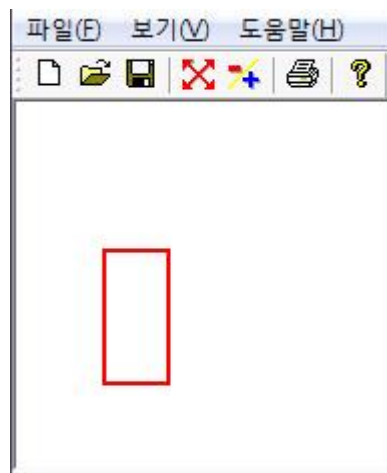
(x1, y1)를 왼쪽 위 꼭지점, (x2, y2)를 오른쪽 아래 꼭지점으로 갖는 nWidth 두께, nColor색의 사각형을 그린다.

예)

```
bool readFile(const char* filename){
    setWindow(0,0,5,5,0);
    //일반좌표계로 (0,0)부터 (5,5)까지를 윈도우 영역으로 설정한다.
}

static void drawBuffered(){
    DrawBox_I(1,1,2,3,2,RGB(255,0,0));
    //일반 좌표계로 (1,2)부터 (2,3)까지를 선의 두께가 2인 빨간색 사각형을 그린다.
}
```

의 결과는 [그림 13]으로 표시된다.



[그림 13] DrawBox_I 예.

e. Buffer를 이용해 사각형을 그리고 내부를 칠하기

syntax void DrawSolidBox_I(double x1, double y1, double x2, double y2, int nLineWidth, COLORREF nLineColor, COLORREF nFillColor)

parameter

x1, y1	사각형 왼쪽 위 꼭지점의 좌표
x2, y2	사각형 오른쪽 아래 꼭지점의 좌표
nWidth	사각형의 외곽선 두께
nLineColor	사각형의 외곽선의 색
nFillColor	사각형의 내부를 채울 색

기능

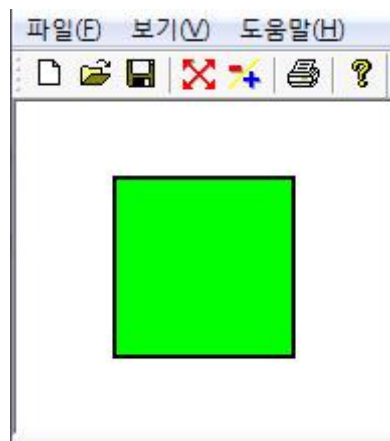
(x1, y1)를 왼쪽 위 꼭지점, (x2, y2)를 오른쪽 아래 꼭지점, 두께 nWidth에 nLineColor 색 테두리, nFillColor로 채워진 사각형을 그린다.

예)

```
bool readFile(const char* filename){
    setWindow(0,0,5,5,0);
    //일반좌표계로 (0,0)부터 (5,5)까지를 윈도우 영역으로 설정한다.
}

static void drawBuffered(){
    DrawSolidBox_I(1,1,4,4,2,RGB(10,10,10),RGB(0,255,0));
    //일반 좌표계로 (1,1)부터 (4,4)까지의 사각형을
    //두께2 검은색 테두리에 초록색으로 채워진 사각형을 그린다.
}
```

의 결과는 [그림 14]와 같다.



[그림 14] DrawSolidBox 예.

f. Buffer를 이용해 사각형을 그리고 내부를 hatch로 채우기

syntax void DrawHatchBox_I(double x1, double y1, double x2, double y2, int nLineWidth, COLORREF nLineColor, int nType, COLORREF nFillColor)

parameter

x1, y1	사각형 왼쪽 위 꼭지점의 좌표
x2, y2	사각형 오른쪽 아래 꼭지점의 좌표
nWidth	사각형의 외곽선 두께
nLineColor	사각형의 외곽선의 색
nType	사각형의 내부를 채울 hatch를 지정한다. HS_BDIAGONAL : '\' HS_CROSS : '+' HS_DIAGCROSS : 'X' HS_FDIAGONAL : '/' HS_HORIZONTAL : '-' HS_VERTICAL : ' '
"nFillColor	Hatch에 사용될 색.

기능

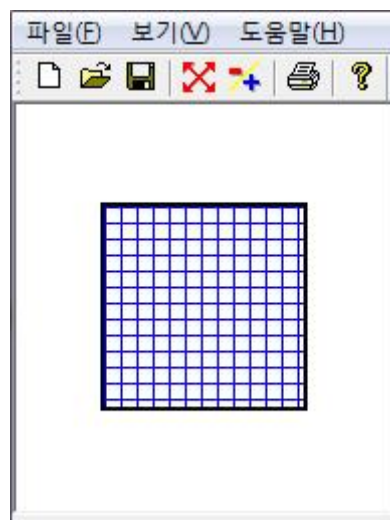
(x1, y1)부터 (x2, y2)까지의 사각형을 두께 nWidth에 nLineColor 색 테두리로 그리고 그 내부를 nFillColor색 nType hatch로 채운다.

예)

```
bool readFile(const char* filename){
    setWindow(0,0,5,5,0);
    //일반좌표계로 (0,0)부터 (5,5)까지를 윈도우 영역으로 설정한다.
}

static void drawBuffered(){
    DrawHatchBox_I(1,1,4,4,2,RGB(10,10,10),HS_CROSS,RGB(0,0,255));
    //일반 좌표계로 (1,1)부터 (4,4)까지의 사각형을
    // 선의 두께가 2, 검은색 테두리로 그리고 파란색 "+"hatch로 채운다.
}
```

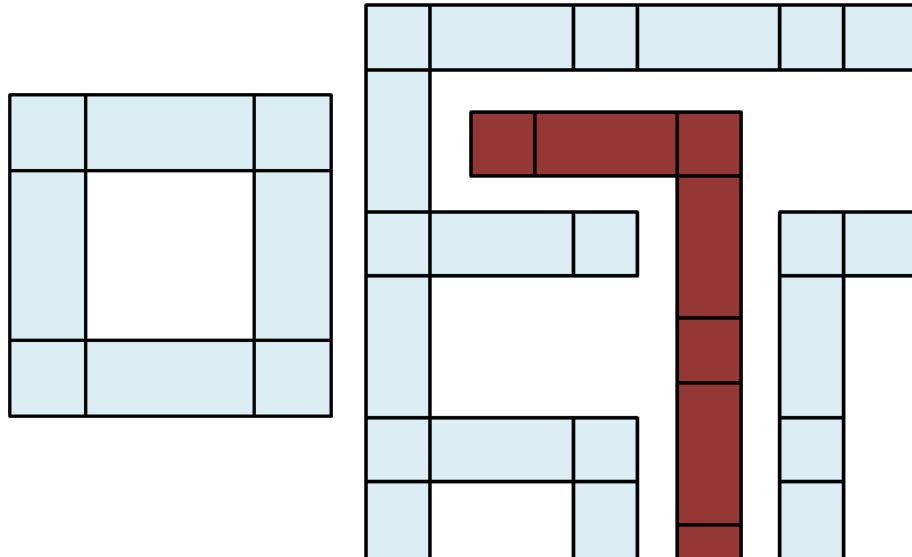
이 결과는 [그림 15]와 같다.



[그림 15] DrawHatchBox 예.

이외에도 DrawDirect 방식에서 쓰이는 그리기 함수나 다각형을 그리는 함수 등이 있으니 필요한 사람은 첨부한 pdf파일을 참고하기 바란다. 위의 주어진 함수들을 응용하여 자유롭게 미로텍스트 입력파일(~.maz)를 그림으로 변환해보자.

힌트를 주자면 [그림 16]과 같이 미로의 한 칸의 각 모서리를 정사각형, 각 변을 정사각형으로 그리는 방식을 취하거나 학생 개개인이 원하는 방식대로 미로를 그려도 무방하다. 미로 칸 내부는 후에 길을 그릴 수 있도록 넉넉히 남겨둔다. 길은 [그림 16]과 같이 사각형으로 그릴수도 있고 선을 사용해서 그릴수도 있다. 자유롭게 선택한다.



[그림 16] 주어진 함수를 이용한 간단한 미로그리기 예.
(좌) :미로 한칸, (우) 길이 그려진 미로의 일부분.

Step3. 메모리 해제

불러온 파일의 정보가 저장되어 있는 상태에서 새로 다른 파일을 불러올 경우, 기존의 메모리를 해제시켜주어야 한다. User Code\usercode.cpp의 함수 freeMemory()에 자신이 만든 자료구조를 할당한 메모리들을 해제시키는 코드를 삽입한다. 이 함수에서 메모리 해제를 완전하게 하지 않으면, 메모리 부족 등 문제가 발생할 수 있다.

4. 숙제 및 보고서 작성

4-1 예비보고서

1. DFS, BFS 알고리즘에 대해 조사하고 간략히 요약한다.
2. 미로문제에서 DFS, BFS를 수행하기 위한 자료구조를 설계하고 그 자료구조의 공간 복잡도를 보인다.
3. 설계한 자료구조에서 DFS, BFS를 어떻게 수행할지 간략히 보인다.
4. 작성한 예비 보고서는 실험 시작 전 제출하여야 한다.

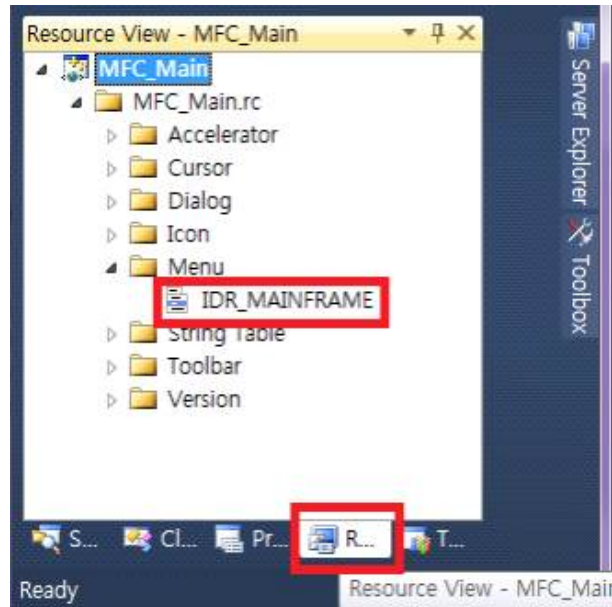
4-2. 숙제

3주차 실험에서는 2주차에서 작성한 프로그램을 바탕으로, ~.maz 파일을 읽어 들여 미

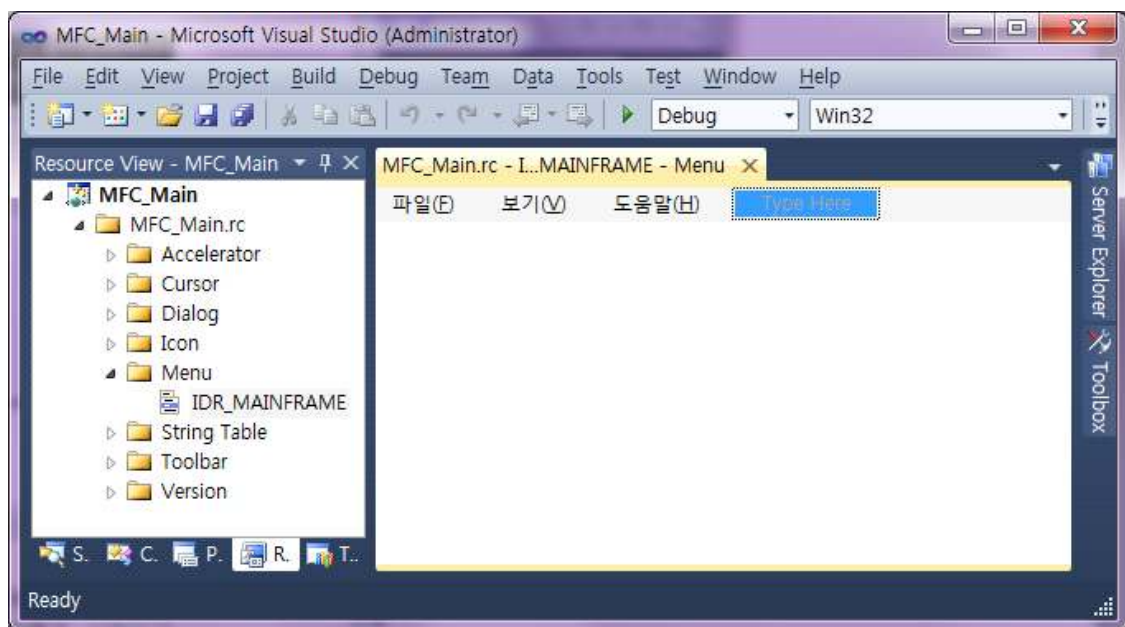
로의 경로를 찾는 프로그램을 설계할 것이다. 이 때, BFS, DFS 중 두 가지 방법을 통해 경로를 탐색하게 되는데, 이를 위해 BFS, DFS 버튼을 toolbar에 추가하자.

toolbar에 버튼을 추가하는 방법은 다음과 같다.

Step 1. 프로젝트 워크스페이스 창에서 [Resource View]탭을 선택하고, Menu폴더 밑의 IDR_MAINFRAME을 선택한다. 그러면 [그림 18]과 같은 메뉴 편집창이 생긴다.

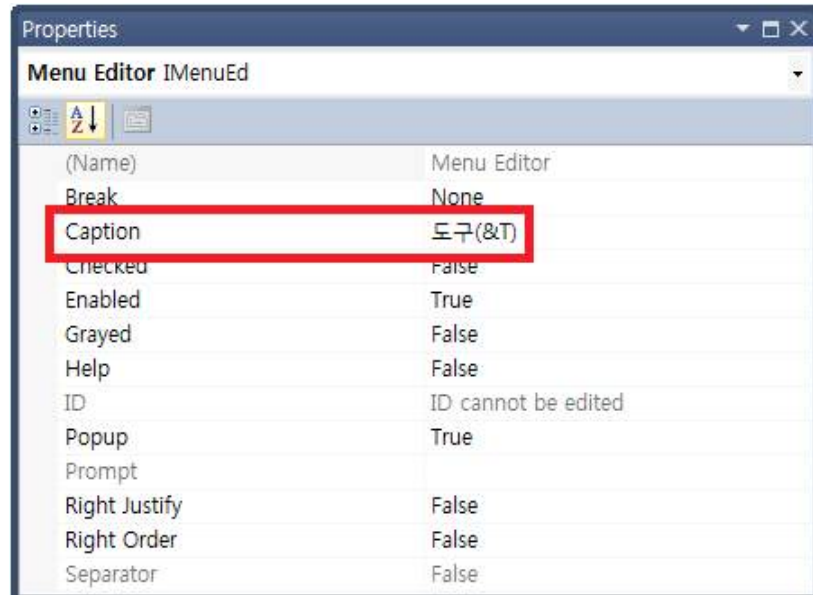


[그림 17] 리소스 뷰에서
메뉴 편집 화면 불러오기.



[그림 18] 메뉴 편집 화면.

Step 2. 메뉴 편집창의 메뉴의 '도움말' 옆 박스를 오른쪽 클릭 → [Insert New] → 더블 클릭을 하면, [그림 19]에서 보이는 것 같이 메뉴 속성창이 뜨는데 여기의 'Caption' 에디트 박스 안에 메뉴의 이름을 입력하고 엔터를 치면 새 메뉴가 생성된다. (이름 옆에 '(&T)' 이라고 쓰면 자동으로 alt+T 단축키가 생성됨). '도구' 메뉴를 생성하고 드래그 하여 '보기'메뉴와 '도움말' 메뉴 사이로 옮기자.

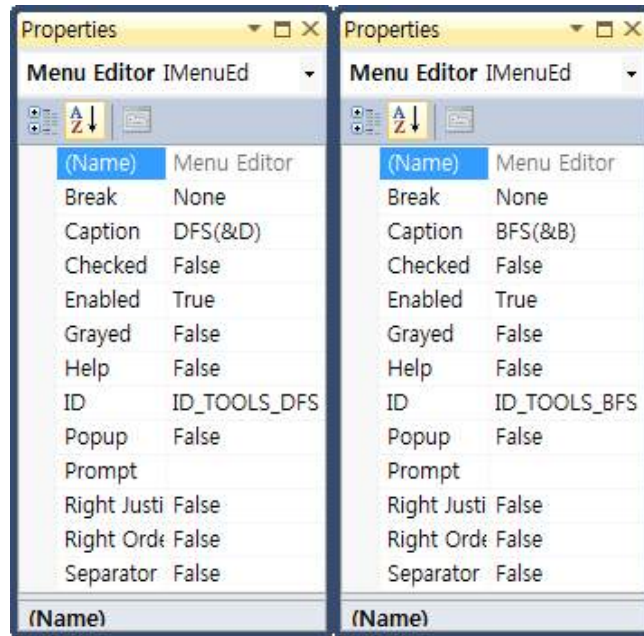


[그림 19] '도구' 주메뉴 추가하기.

- ☞ **Break** : 메뉴가 길어지면 이단으로 구성할 것인지 등의 모양을 결정한다.
- ☞ **Checked** : 메뉴에 체크 표시가 생긴다.
- ☞ **Enabled** : 메뉴를 클릭할 수 있다.
- ☞ **Grayed** : 메뉴의 색이 회색으로 바뀐다. Grayed + Inactive 되면 죽어있는 메뉴이다.
- ☞ **Help** : Help창과 연동할 수 있다.
- ☞ **Pop-Up** : 서브메뉴를 가지고 있는 메뉴이다. ('도구' 메뉴와 같은 메뉴를 말한다.)
- ☞ **Separator** : 메뉴 내에 분리 선을 둔다.

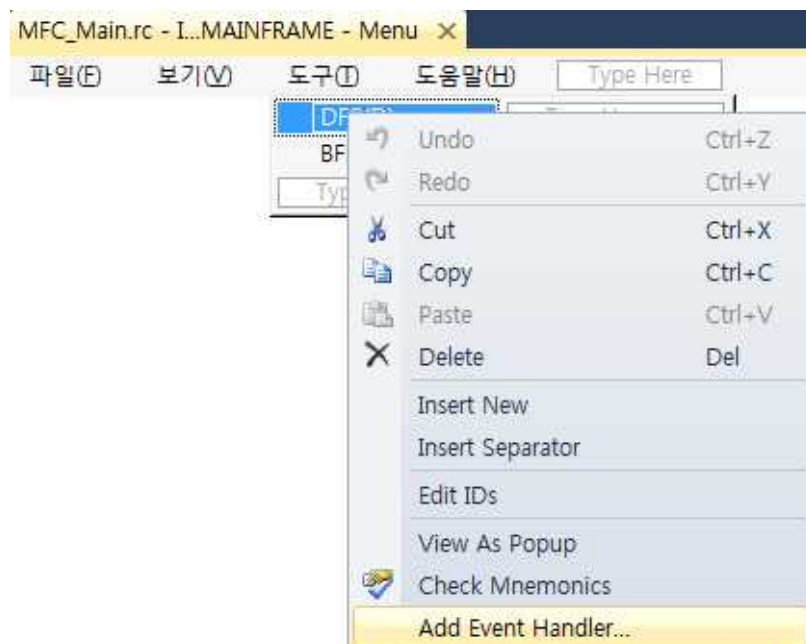
Step 3. Step 2의 메뉴 만드는 것과 유사하게 '도구' 메뉴를 누르고 그 밑의 빈 공간을 오른쪽 클릭 → [Insert New] →더블 클릭해서 서브 메뉴 작성 한다.

ID	Caption
ID_TOOLS_DFS	DFS
ID_TOOLS_BFS	BFS



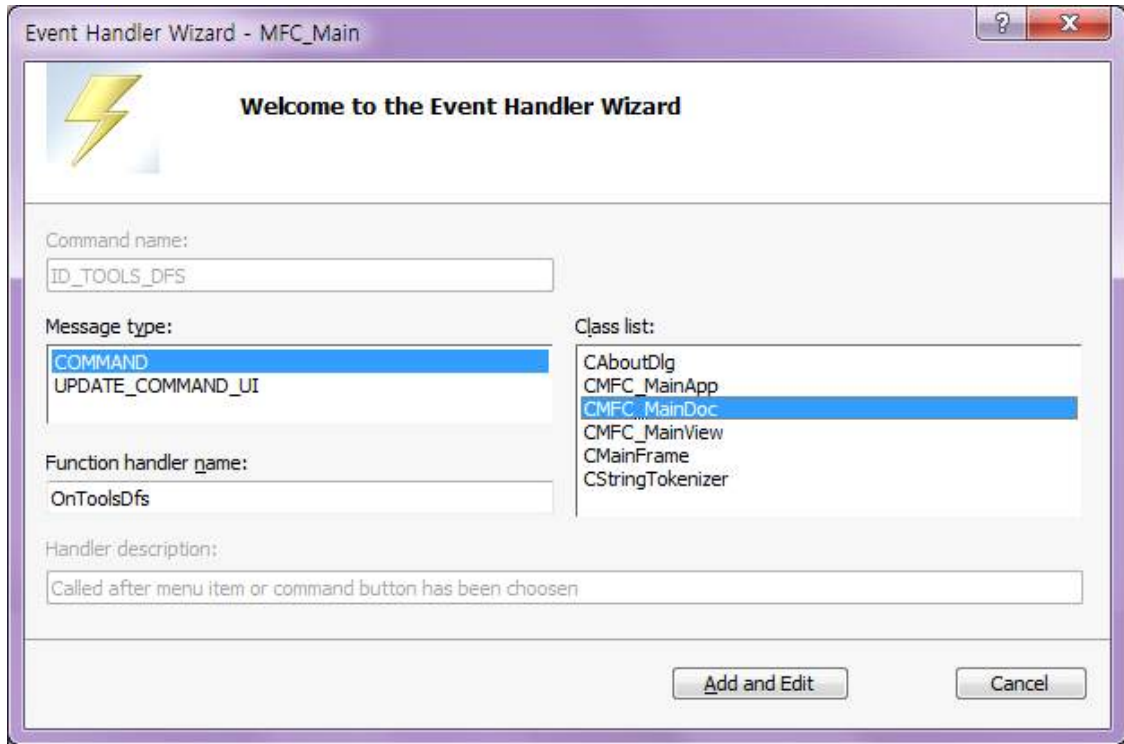
[그림 20] 서브 메뉴 만들기.

Step 4. 메뉴를 다 만들면 [그림 21] 과 같다. 서브메뉴에서 오른쪽 마우스 버튼을 눌러 [Add Event Handler...]를 선택하여 Event Handler Wizard를 실행시킨다.



[그림 21] Event Handler 추가하기.

Step 5. Add Event Handler 버튼을 클릭하면 [그림 22]과 같은 Event Handler Wizard 창이 뜨는데 이때 'Message type' 리스트 안의 'COMMAND'(메뉴클릭)과 Class list 안의 'CMFC_MainDoc'를 선택한 후 하단의 'Add and Edit' 버튼을 클릭한다.



[그림 22] Event Handler Wizard.

Step 6. Add and Edit 버튼을 클릭하면 [그림 23]와 같이 MFC_MainDoc.cpp 파일에 보조메뉴를 선택했을 때 실행될 함수가 생성됨을 확인할 수 있다.



[그림 23] 추가된 DFS 메뉴 Event Handler.

이 함수 안에는 다음과 소스를 추가하자.

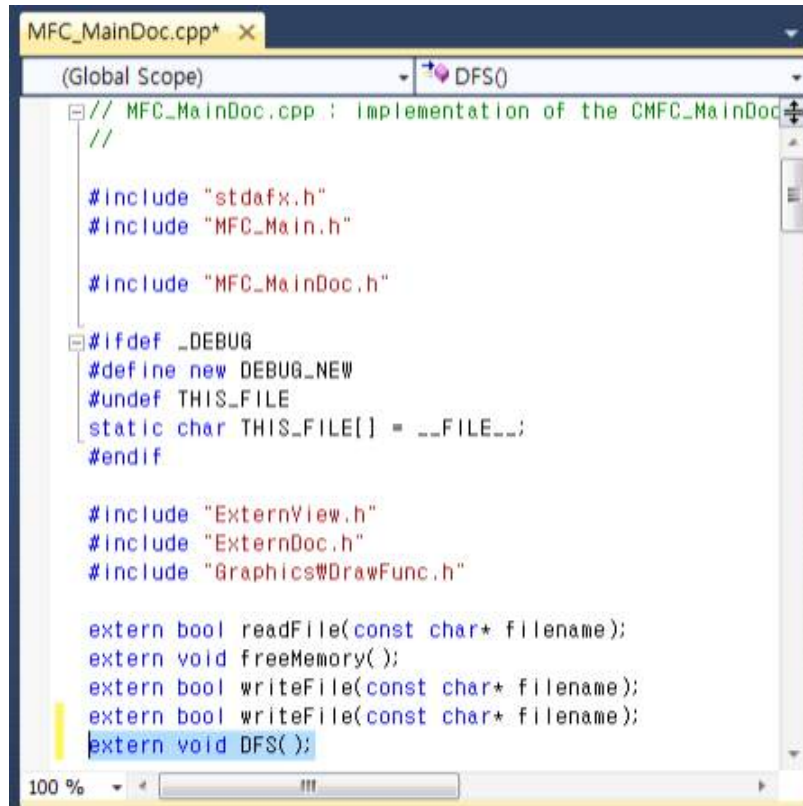
```
// execute menu handler function
DFS();

//if you want to draw, do not erase following code
#ifdef GRAPHICS_BUFFERED
    g_pView->m_bRedraw = true;
#endif
```

```
g_pView->Invalidate();
```

해당 소스 안의 DFS(); 함수는 User Code 디렉토리의 user code.cpp에 작성할 것이므로, MFC_Main.DOC.cpp의 가장 상단부에

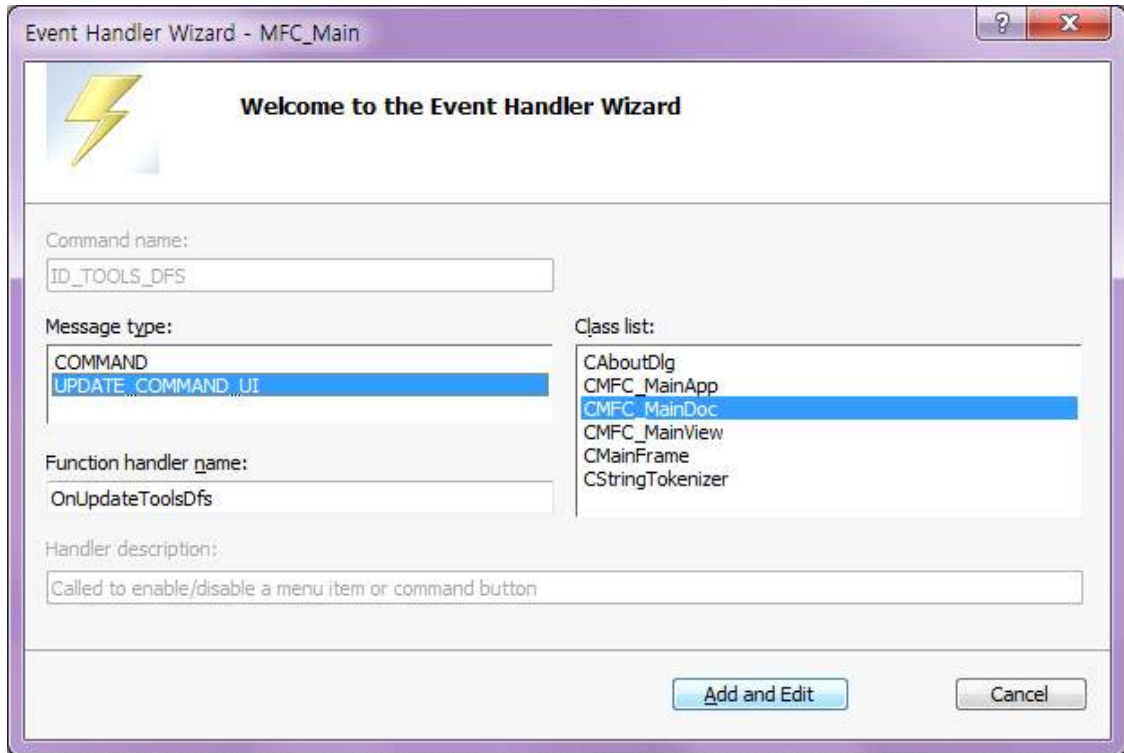
[그림 24] 처럼 extern bool DFS(); 로 DFS 함수를 선언해주자.



[그림 24] DFS 함수 선언.

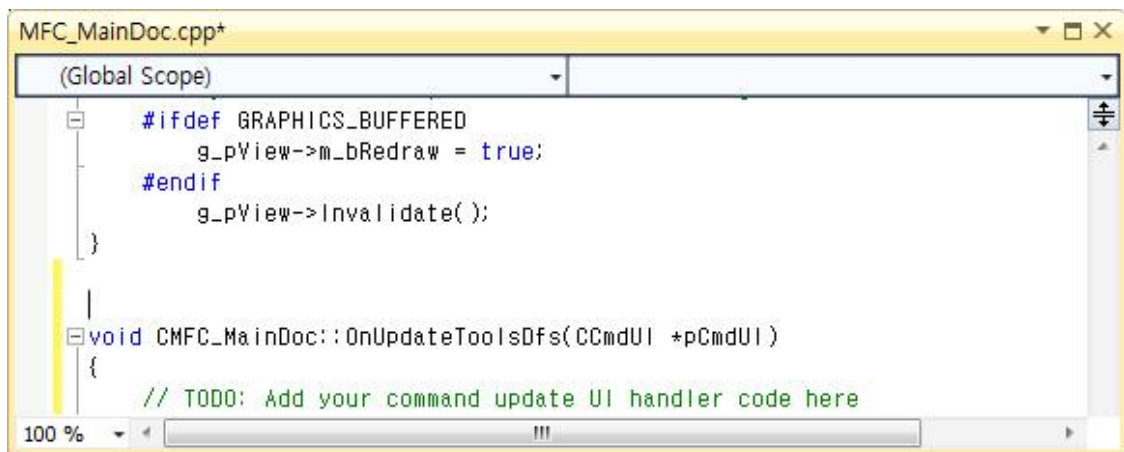
Step 7. Step 5, Step 6과 같은 방법으로 BFS, SP 메뉴에 대한 Event Handler도 추가한다.

Step 8. 이제 data가 로드 되지 않았을 때 각 메뉴들을 비활성화 시키기 위해 몇 가지 절차를 취하자. step.6와 동일한 과정을 거쳐서 'DFS' 서브메뉴에 [그림 25]처럼 Event Handler Wizard를 실행시킨 후, 이번에는 'Message type' 리스트에서 'UPDATE_COMMAND_UI', 'Class list'에서는 'CMFC_MainDOC'를 선택하고 하단의 Add and Edit 버튼을 클릭한다.



[그림 25] 데이터를 열지 않았을 때 메뉴 비활성화 시키기.

Step 9. [그림 26]처럼 MFC_MainDoc.cpp 파일에 메소드가 생성되었다.



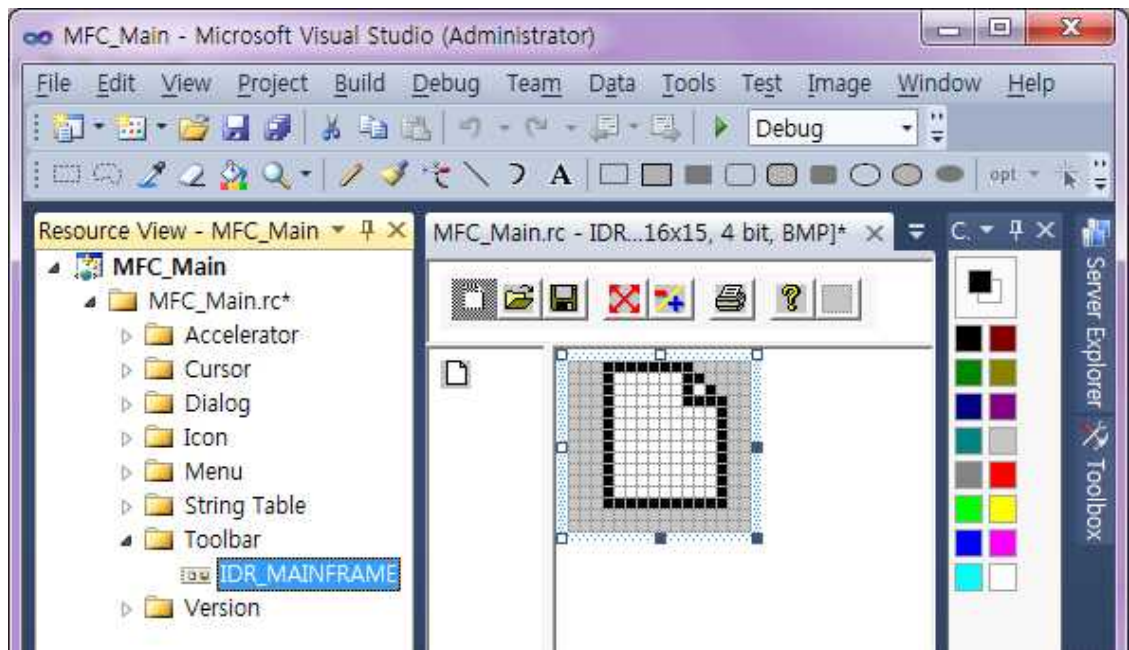
[그림 26] 생성된 OnUpdateToolsDfs 메소드.

이 메소드안에 다음과 같은 코드를 써 넣는다.

```
// if user data is loaded, enable "DFS" menu
pCmdUI->Enable(m_bDataLoaded);
```

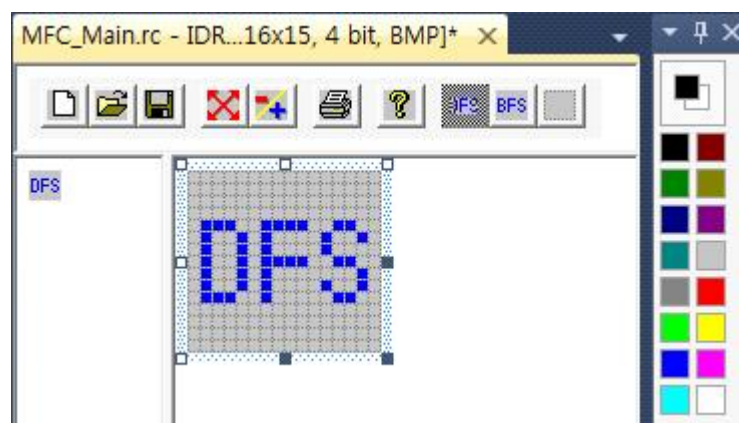
Step 10. BFS에 대해서도 Step 8, 9를 반복하여 데이터를 불러오지 않았을 때 BFS메뉴가 비활성화 되도록 하자.

Step 11. 이제 툴바에 DFS, BFS 버튼을 추가해보자. 프로젝트 워크스페이스 창에서 [Resource View]탭을 선택하고, Toolbar 디렉토리의 IDR_MAINFRAME를 선택한다. 그러면 [그림 27]과 같은 툴바 편집창이 뜬다.



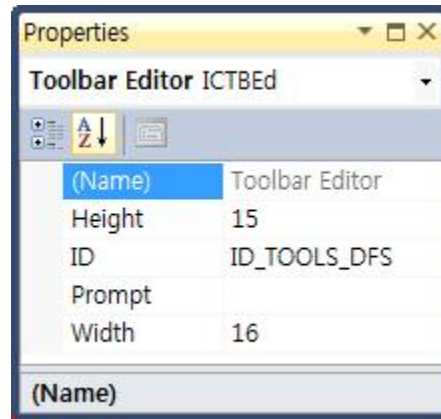
[그림 27] Toolbar 편집창.

Step 12. 툴바의 맨 오른쪽 빈 버튼을 누르면 아래 그림같이 아이콘을 편집할 수 있다. 그림판에 그림을 그리는 것과 비슷하게 그리면 된다.



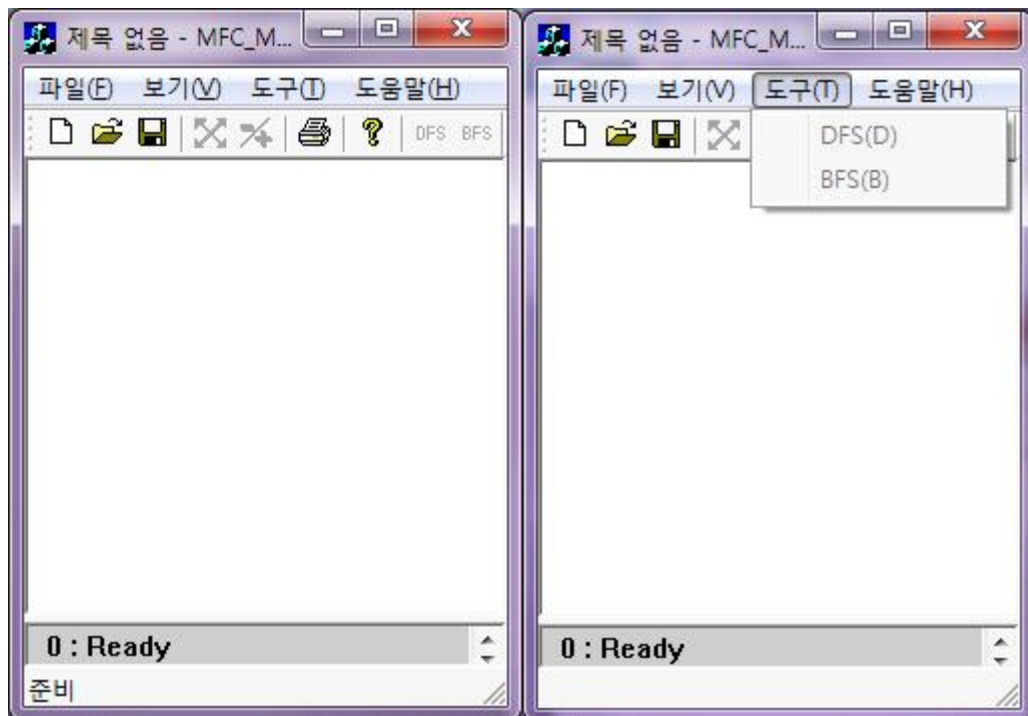
[그림 28] Toolbar 버튼 그리기.

Step 13. 그려진 툴바내의 버튼을 더블 클릭하면 아래 그림과 같이 ID를 추가할 수 있다. 'DFS'에 대해서 'ID_TOOLS_DFS'를 입력하고 엔터를 친다. 'BFS'에도 동일한 작업을 수행한다.

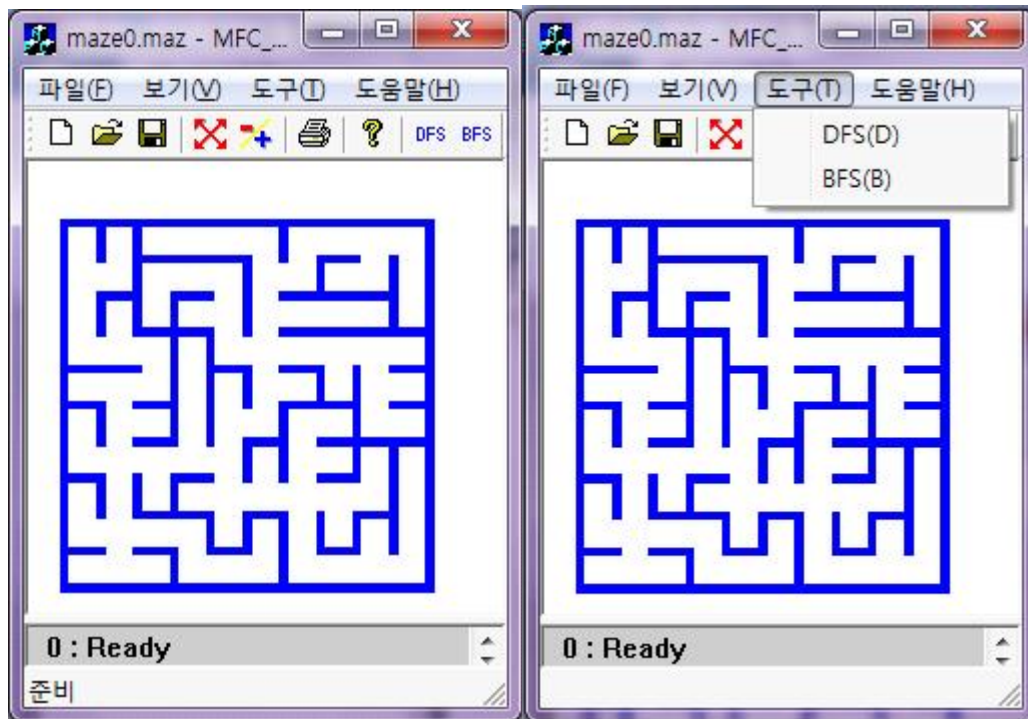


[그림 29] 버튼에 ID 추가하기.

Step 14. 모든 과정을 성공적으로 수행하면 다음과 같은 화면이 만들어진다.



[그림 30] 데이터를 불러오기 전 비활성화 된 툴바 버튼(좌), 서버 메뉴(우).



[그림 31] 데이터를 불러온 후 활성화 된
툴바 버튼(좌), 서브메뉴(우)

4-3 결과 보고서

아래 보인 사항을 작성하여 조교가 공지한 기간까지 제출하시오.

1. 실험시간에 작성한 프로그램에서 자료구조와 구성한 자료구조를 화면에 그리는 방법들을 설명한다. 완성한 자료구조를 이용한 그래픽 전환 작업의 시간 및 공간복잡도를 보이고 실험 전에 생각한 방법과 어떻게 다른지 아울러 기술한다.
2. 본 실험 및 숙제를 통해 습득한 내용을 한 내용을 기술한다.

PRJ-2(2주차/3주) 예비보고서

전공:

학년:

학번:

이름

PRJ-2(1주차/3주) 결과보고서

전공:

학년:

학번:

이름

