**알고리즘 및 실습**

**Term-Project 보고서**



|  |  |
| --- | --- |
| **과목명** | 알고리즘 및 실습 |
| **분반** | 1 |
| **교수님** | 민 준 기 교수님 |
| **조** | 7 |
| **조원** | 2012136132 최강석  2012136139 최형근  2012136139 허준석 |
| **제출일** | 2016. 06. 07 |











|  |
| --- |
| **프로젝트 목표** |
| 3x3 범위의 스캔을 사용할 수 있는 쥐가 최소한의 에너지를 사용하여 미로를 탈출하도록 한다. |

|  |
| --- |
| **개발 환경** |
| 운영체제 : Window 7 – 64bit  개발 툴 : Visual Studio 2010  사용언어 : C++ |

|  |
| --- |
| **프로젝트 진행도** |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **월** | **3월** | | | | **4월** | | | | | **5월** | | | | | **6월** | | | **주차** | **1** | **2** | **3** | **4** | | **1** | **2** | **3** | **4** | | **1** | **2** | **3** | **4** | | **1** | | | **프로그램 구상** |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | | | **스캔**  **구상** |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | | | **프로그램**  **작성** |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | | | **보고서 및**  **PPT 작성** |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | | |

|  |
| --- |
| **소스 코드 : Maze.h** |
| #pragma once  #include <iostream>  #include <fstream>  #include <Windows.h>  #include <conio.h>  #include <cstdlib>  #include <ctime>  using namespace std;  #define ROAD 0 // 길  #define WALL 1 // 벽  #define PASS 2 // 지나온 길  #define BLOCK 3 // 길 막아 버리기  #define MOUSE 4 // 현재 위치  #define NOWAY 7 // 막다른 길  #define START 8 // 시작지점  #define END 9 // 도착지점  #define MAX\_STACK\_SIZE 10000  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 좌표  struct Point2D{  int x; // x 좌표  int y; // y 좌표  Point2D(int xx=0, int yy=0)  {  x = xx;  y = yy;  }  };  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // CMaze 클래스  class CMaze  {  public:  int row; // 미로의 행  int col; // 미로의 열  int\*\* maze; // 미로 배열  int\*\* map; // 쥐의 이동 현황  Point2D stack[MAX\_STACK\_SIZE]; // 최적화된 경로 배열  Point2D e; // stack 배열 초기화 변수  int m\_top; // stack 배열 인덱스  Point2D m\_start; // 미로의 입구 좌표  Point2D m\_exit; // 미로의 출구 좌표  Point2D now; // 현재 위치  Point2D pre; // 직전 위치  char tmp; // 파일의 내용을 읽을 때 사용할 변수  ifstream file; // 파일을 읽을 때 사용할 변수  int energy; // 에너지  double mana; // 마나  int usescan; // 스캔을 사용한 횟수  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 생성자  CMaze(void)  {  // 변수 초기화  char name[30];  row = 1;  col = 0;  m\_start.x = 0;  m\_start.y = 1;  mana = 0.0;  usescan = 0;  e.x = -1;  e.y = -1;  m\_top = 0;  // 최적화 경로 배열 초기화  for(int i = 0; i < MAX\_STACK\_SIZE; i++)  {  stack[i].x = e.x;  stack[i].y = e.y;  }  // 파일 이름을 입력 받는다.  cout << "텍스트 파일 이름을 입력하세요 : ";  cin >> name;  // 화면을 지운다.  system("cls");  // 파일을 연다.  file.open(name);  // 만약 파일이 없을 경우 에러 메시지를 준다.  if(!file)  {  cerr << "지도를 찾을 수 없습니다.!" << endl;  exit(1);  }  // 미로 사이즈를 탐색한다.  SearchSize();  // 미로 사이즈 만큼 배열을 동적 할당한다.  Init(col, row);  // 초기 에너지는 row\*col\*2  energy = row\*col\*2;  // 미로를 2차원 배열 maze와 map에 저장한다.  StoreInArray();  // 미로를 콘솔창에 출력한다.  PrintMaze();  // 행과 열을 콘솔창에 출력한다.  PrintSize(row, col);  // 에너지 현황을 콘솔 창에 출력한다.  PrintEnergy(energy);  // 마나와 스캔 현황을 콘솔창에 출력한다.  PrintMana(mana, usescan);  getchar();  getchar();  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // 소멸자  ~CMaze(void)  {  // 메모리 동적 해제  Reset();  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // IsEmpty : stack 배열이 비었는가?  bool IsEmpty()  {  return m\_top == 0;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // IsFull : stack 배열이 꽉 찼는가?  bool IsFull()  {  return m\_top == MAX\_STACK\_SIZE;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Push : 해당 좌표를 stack 배열에 삽입한다.  void Push(Point2D &p)  {  // 배열이 꽉 찼으면 에러메세지를 준다.  if(IsFull())  {  printf("Error : Stack Full Error\n");  return;  }  m\_top++; // 인덱스 1 증가  stack[m\_top] = p; // 해당 좌표를 삽입    }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Pop : stack에서 좌표 하나를 삭제한다.  void Pop()  {  // 배열이 비어 있으면 에러메세지를 준다.  if(IsEmpty())  {  printf("Error : Stack Empty Error\n");  return ;  }  stack[m\_top] = e; // 좌표 하나를 삭제한다.  m\_top--; // 인덱스 1 감소  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Peek : 가장 최근에 삽입한 좌표를 반환한다.  Point2D Peek()  {  // 배열이 비어있으면 에러메세지를 준다.  if(IsEmpty())  {  printf("Error : Stack Empty Error\n");  return 0;  }  return stack[m\_top]; // 좌표 반환  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // IsNoWay :현재 좌표가 막다른 길인가?  bool IsNoWay(int x, int y)  {  // 4개의 방향 중에서 1개를 제외하고 나머지가 벽인 경우 true  if((maze[x][y+1] != WALL) && (maze[x][y-1] == WALL) && (maze[x+1][y] == WALL) && (maze[x-1][y] == WALL))  return true;  if((maze[x][y+1] == WALL) && (maze[x][y-1] != WALL) && (maze[x+1][y] == WALL) && (maze[x-1][y] == WALL))  return true;  if((maze[x][y+1] == WALL) && (maze[x][y-1] == WALL) && (maze[x+1][y] != WALL) && (maze[x-1][y] == WALL))  return true;  if((maze[x][y+1] == WALL) && (maze[x][y-1] == WALL) && (maze[x+1][y] == WALL) && (maze[x-1][y] != WALL))  return true;  return false; // 아니면 false  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // IsCross : 현재 좌표가 교차로인가? (갈림길인가?)  bool IsCross(int x, int y)  {  // 4개의 방향 중에서 3개가 길인 경우 true  if((maze[x][y+1] == WALL) && (maze[x][y-1] == ROAD) && (maze[x+1][y] == ROAD) && (maze[x-1][y] == ROAD))  return true;  if((maze[x][y+1] == ROAD) && (maze[x][y-1] == WALL) && (maze[x+1][y] == ROAD) && (maze[x-1][y] == ROAD))  return true;  if((maze[x][y+1] == ROAD) && (maze[x][y-1] == ROAD) && (maze[x+1][y] == WALL) && (maze[x-1][y] == ROAD))  return true;  if((maze[x][y+1] == ROAD) && (maze[x][y-1] == ROAD) && (maze[x+1][y] == ROAD) && (maze[x-1][y] == WALL))  return true;  // 4개의 방향 모두가 길인 경우 true  if((maze[x][y+1] == ROAD) && (maze[x][y-1] == ROAD) && (maze[x+1][y] == ROAD) && (maze[x-1][y] == ROAD))  return true;  return false; // 아니면 false  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // NearExit : 근처에 출구가 있는가?  bool NearExit(int x, int y)  {  // 4개의 방향 중에서 출구가 있으면 true  if(maze[x][y+1] == END)  return true;  if(maze[x+1][y] == END)  return true;  if(maze[x][y-1] == END)  return true;  if(maze[x-1][y] == END)  return true;  return false; // 없으면 false  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // CanGo : 현재 좌표가 계속 진행할 수 있는 위치인가?  bool CanGo(int x, int y)  {  // 4개의 방향 중에서 1개라도 갈 수 있는 길이면 true  if((map[x][y+1] == ROAD) || (map[x][y-1] == ROAD) || (map[x+1][y] == ROAD) || (map[x-1][y] == ROAD))  return true;  return false; // 아니면 false  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Init : 메모리 동적 할당  void Init(int w, int h)  {  col = w;  row = h;    maze = new int\*[row]; // 행에 대해서 동적 할당  for(int i=0; i<h; i++)  maze[i] = new int[col]; // 열에 대해서 동적 할당  map = new int\*[row]; // 행에 대해서 동적 할당  for(int i=0; i<h; i++)  map[i] = new int[col]; // 열에 대해서 동적 할당  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Reset : 메모리 동적 해제  void Reset()  {  if(maze != NULL && map != NULL)  {  for(int i=0; i<row; i++)  delete[] maze[i]; // 열에 대해서 동적 해제  delete[] maze; // 행에 대해서 동적 해제  for(int i=0; i<row; i++)  delete[] map[i]; // 열에 대해서 동적 해제  delete[] map; // 행에 대해서 동적 해제  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // SearchSize : 미로 크기를 탐색한다. (행, 열 크기 구하기)  void SearchSize()  {  file.clear();  while(!file.eof())  {  file.get(tmp); // 파일의 글자 하나를 읽어온다.  // 읽어온 글자가 1 또는 0이면  if(tmp == '1' || tmp == '0'){  if(row == 1) // (열의 수가 계속 증가하는 것을 방지) col++; // 열의 수 1 증가  }  // 읽어온 글자가 \n이면  if(tmp == '\n'){  row++; // 행의 수 1 증가  }  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // StoreInArray : 미로를 2차원 배열 maze와 map에 저장  void StoreInArray()  {  file.clear();  file.seekg(0);  file.get(tmp); // 글자 하나를 읽어온다.  for(int i = 0; i < row; i++)  {  for(int j = 0; j < col; j++)  {  if(tmp == '0') // 글자가 0이면  maze[i][j] = ROAD; // 길 else if(tmp == '1') // 글자가 1이면  maze[i][j] = WALL; // 벽  else if(tmp == ' ') // 글자가 공백이면  j--; // j--  do  {  file.get(tmp);  }while(tmp == '\n' && !file.eof());  }  }  //\*\*\* 출구 좌표를 찾는다.  // (입구와 출구는 테두리 벽 중 유일하게 길이다.)  for(int i = 0; i < col; i++){  // 맨 위 테두리 중에서 길이 있으면  if(maze[0][i] == ROAD){  maze[0][i] = END; // 그것은 출구다.  m\_exit.x = 0; // x 좌표 저장  m\_exit.y = i; // y 좌표 저장  }  // 맨 아래 테두리 중에서 길이 있으면  else if(maze[row-1][i] == ROAD){  maze[row-1][i] = END; // 그것은 출구다.  m\_exit.x = row-1; // x 좌표 저장 m\_exit.y = i; // y 좌표 저장  }  }  for(int i = 0; i < row; i++){  // 맨 왼쪽 테두리 중에서 길이 있으면  if(maze[i][0] == ROAD){  maze[i][0] = END; // 그것은 출구다.  m\_exit.x = i; // x 좌표 저장  m\_exit.y = 0; // y 좌표 저장  }  // 맨 오른쪽 테두리 중에서 길이 있으면  else if(maze[i][col-1] == ROAD){  maze[i][col-1] = END; // 그것은 출구다.  m\_exit.x = i; // x 좌표 저장  m\_exit.y = col-1; // y 좌표 저장  }  }  maze[0][1] = START; // 입구는 무조건 (0,1)  // maze배열을 map배열에 복사  for(int i = 0; i < row; i++)  for(int j = 0; j < col; j++)  {  if(maze[i][j] == ROAD || maze[i][j] == START || maze[i][j] == END)  map[i][j] = ROAD;  else if(maze[i][j] == WALL)  map[i][j] = WALL;  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // PrintMaze : 미로를 콘솔창에 출력한다.  void PrintMaze()  {  int i, j; // 인덱스  for(i = 0; i < row; i++)  {  for(j = 0; j < col; j++)  {  if(maze[i][j] == WALL) // 해당 좌표가 벽이면  {  Color(2); // 초록색  cout << "■"; // 벽 }  else if(maze[i][j] == ROAD) // 해당 좌표가 길이면  {  cout << "　"; // 길  }  else if(maze[i][j] == START) // 해당 좌표가 시작지점이면  {  Color(14); // 노란색  cout << "ⓢ"; // 시작지점  }  else if(maze[i][j] == END) //해당 좌표가 도착지점이면 {  Color(12); // 빨간색  cout << "¶"; // 도착지점  }  }  cout << endl;  }    Go(col\*2+2, 2);  Color(12); // 빨간색  cout << "2016년도 1학기 알고리즘 및 실습 Term Project";  Color(15); // 하얀색  Go(col\*2+2, 30);  cout << "<제작>";  Go(col\*2+2, 31);  cout << "2012136132 최강석";  Go(col\*2+2, 32);  cout << "2012136139 최형근";  Go(col\*2+2, 33);  cout << "2012136142 허준석";  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // PrintSize : 행과 열 크기를 콘솔창에 출력한다.  void PrintSize(int row, int col)  {  Color(14); // 노란색  Go(col\*2+2, 7);  cout << "[행] : " << row;  Go(col\*2+2, 8);  cout << "[열] : " << col;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // PrintEnergy : 에너지를 콘솔창에 출력한다.  void PrintEnergy(int e)  {  Color(10); // 연두색  Go(col\*2+2, 10);  cout << "[에너지] : " << e;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // PrintMana : 마나와 스캔 횟수를 콘솔창에 출력한다.  void PrintMana(double m, int s)  {  Color(11); // 하늘색  Go(col\*2+2, 12);  printf("　[마나] : %.1f", m);  Color(7); // 회색  Go(col\*2+2, 14);  cout << "　 [스캔] : " << s;  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // DrawMouse : 해당 좌표에 쥐를 그린다.  void DrawMouse(Point2D p)  {  Go(p.y\*2, p.x);  Color(7); // 회색  cout << "ⓜ";  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // EraseMouse : 해당 좌표에 있는 쥐를 지운다.  void EraseMouse(Point2D p)  {  Go(p.y\*2, p.x);  Color(7); // 회색  cout << "　";  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // OptimalMouse : 최적화된 경로를 출력한다.  void OptimalMouse(Point2D p)  {  Go(p.y\*2, p.x);  Color(14); // 노란색  cout << "▦";  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // SearchExit : 미로 찾기를 수행한다.  void SearchExit()  {  now = m\_start; // 현재 좌표 = 시작지점  DrawMouse(now); // 현재 좌표에 쥐를 그린다.  Push(now); // 현재 좌표를 stack 배열에 삽입한다.  map[now.x][now.y] = PASS; // 현재 좌표는 왔던 길이다.  PrintEnergy(energy); // 에너지 출력  PrintMana(mana, usescan); // 마나&스캔 출력  while(1)  {  Point2D right = now;  Point2D left = now;  Point2D up = now;  Point2D down = now;  right.y = now.y + 1; // 현재 좌표의 오른쪽  left.y = now.y - 1; // 현재 좌표의 왼쪽  up.x = now.x - 1; // 현재 좌표의 위쪽  down.x = now.x + 1; // 현재 좌표의 아래쪽  pre = now; // 이전 위치 = 현재 위치로 설정  EraseMouse(pre); // 이전 위치의 쥐를 지운다.  // 미로 탐색 순서 : 오른쪽 – 아래쪽 – 왼쪽 – 위쪽  if(map[right.x][right.y] == ROAD) // 오른쪽이 길이면  {  // 오른쪽으로 이동  now.x = right.x;  now.y = right.y;  }  else if(map[down.x][down.y] == ROAD) // 아래쪽이 길이면  {  // 아래쪽으로 이동  now.x = down.x;  now.y = down.y;  }  else if(map[left.x][left.y] == ROAD) // 왼쪽이 길이면  {  // 왼쪽으로 이동  now.x = left.x;  now.y = left.y;  }  else if(map[up.x][up.y] == ROAD) // 위쪽이 길이면  {  // 위쪽으로 이동  now.x = up.x;  now.y = up.y;  }  DrawMouse(now); // 현재 좌표에 쥐를 그린다.  Push(now); // 현재 좌표를 stack 배열에 삽입한다.  energy--; // 쥐가 이동했으므로 에너지 1 감소  PrintEnergy(energy); // 에너지 출력  mana = mana + 0.1; // 쥐가 이동했으므로 마나 0.1 증가  if(mana >= 0.9) // 마나가 1이 되면  {  mana = 0.0; // 마나를 0으로 초기화 하고  Scan(ROAD); // 스캔 발동  }  PrintMana(mana, usescan); // 마나&스캔 출력  map[now.x][now.y] = PASS; // 현재 좌표는 왔던 길이다.  if(IsNoWay(now.x, now.y)) // 만약 현재 좌표가 막다른 길이면  {  GoBack(); // 되돌아가는 함수 호출  }  Sleep(100); // 딜레이  if(NearExit(now.x, now.y)) // 근처에 도착지점이 있으면  break; // 반복문 탈출!!  }  PrintOptimal(stack); // 최적화 경로를 출력한다.  PrintResult(); // 최종 결과를 출력한다.  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // GoBack : 막다른 길에 다다랐을 때 다시 돌아간다.  void GoBack()  {  while(1)  {  EraseMouse(now); // 현재 좌표의 쥐를 지운다.  map[now.x][now.y] = WALL; // 현재 좌표를 보이지 않는 벽으로 막는다.  Pop(); // 배열에서 좌표를 삭제한다.  // 만약 현재 위치가 계속 진행 가능한 위치이면  if(IsCross(now.x, now.y) && CanGo(now.x, now.y))  {  map[now.x][now.y] = PASS; // 현재 좌표를 왔던 길로 설정  Push(now); // 다시 현재 좌표를 stack 배열에 삽입  if(CanGo(now.x, now.y))  break; // 반복문 탈출!!  }  now = Peek(); // 현재 좌표 = 가장 최근에 삽입된 좌표  DrawMouse(now); // 현재 좌표에 쥐를 그린다.  energy--; // 에너지 1 감소  PrintEnergy(energy); // 에너지 출력  mana = mana + 0.1; // 마나 0.1 증가  if(mana >= 0.9) // 마나가 1이 되면  {  mana = 0.0; // 마나를 0으로 초기화하고  Scan(PASS); // 스캔 발동  }  PrintMana(mana, usescan); // 마나&스캔 출력  Sleep(100); // 딜레이  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // PrintOptimal : 최적화 경로를 출력한다.  void PrintOptimal(Point2D p[])  {  for(int i = 1; i < m\_top; i++)  OptimalMouse(p[i]);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // PrintResult : 최종 결과를 출력한다. (에너지 사용량 + 스캔 사용 횟수)  void PrintResult()  {  Color(10);  Go(col\*2+2, 18);  cout << "\*\*\*\*\*\*\*\*\*\* 결과 \*\*\*\*\*\*\*\*\*\*";  Go(col\*2+2, 19);  cout << "\* \*";  Go(col\*2+2, 20);  printf("\* 총 에너지 사용량 : %3d \*", row\*col\*2 - energy);  Go(col\*2+2, 21);  printf("\* 총 　스캔 사용량 : %3d \*", usescan);  Go(col\*2+2, 22);  cout << "\* \*";  Go(col\*2+2, 23);  cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*";  Go(col\*2+2, 25);  Color(13);  cout << "- Enter를 두 번 누르면 종료합니다. -";  Go(0, row+1);  getchar();  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Scan : 스캔  void Scan(int x)  {  usescan++; // 스캔 사용 횟수 1 증가  Point2D right = now;  Point2D left = now;  Point2D up = now;  Point2D down = now;  right.y = now.y + 1; // 오른쪽  left.y = now.y - 1; // 왼쪽  up.x = now.x - 1; // 위쪽  down.x = now.x + 1; // 아래쪽  // 스캔 발동 순서 : 오른쪽 – 아래쪽 – 왼쪽 – 위쪽  if(map[right.x][right.y] == x)  {  // 오른쪽 스캔  Scanning(right.x, right.y + 1 );  }  else if(map[down.x][down.y] == x)  {  // 아래쪽 스캔  Scanning(down.x + 1, down.y);  }  else if(map[left.x][left.y] == x)  {  // 왼쪽 스캔  Scanning(left.x, left.y - 1);  }  else if(map[up.x][up.y] == x)  {  // 위쪽 스캔  Scanning(up.x - 1, up.y);  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Scanning : 콘솔 창에 스캔 발동 효과 출력 + 막다른 길 벽으로 막기  void Scanning(int x, int y)  {  for(int i = 0; i < 9; i++)  {  DrawScan(x, y); // 스캔 효과 그리기  DrawScan(x, y-1); // 스캔 효과 그리기  DrawScan(x+1, y-1); // 스캔 효과 그리기  DrawScan(x+1, y); // 스캔 효과 그리기  DrawScan(x+1, y+1); // 스캔 효과 그리기  DrawScan(x, y+1); // 스캔 효과 그리기  DrawScan(x-1, y+1); // 스캔 효과 그리기  DrawScan(x-1, y); // 스캔 효과 그리기  DrawScan(x-1, y-1); // 스캔 효과 그리기  Sleep(50); // 딜레이  DrawBack(x, y); // 복원 or 막다른 길 막기  DrawBack(x, y-1); // 복원 or 막다른 길 막기  DrawBack(x+1, y-1); // 복원 or 막다른 길 막기  DrawBack(x+1, y); // 복원 or 막다른 길 막기  DrawBack(x+1, y+1); // 복원 or 막다른 길 막기  DrawBack(x, y+1); // 복원 or 막다른 길 막기  DrawBack(x-1, y+1); // 복원 or 막다른 길 막기  DrawBack(x-1, y); // 복원 or 막다른 길 막기  DrawBack(x-1, y-1); // 복원 or 막다른 길 막기  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // DrawScan : 해당 좌표에 스캔 효과 그리기  void DrawScan(int x, int y)  {  Go(y\*2, x);  Color(11);  cout << "⊙ËA";  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // DrawBack : 스캔 효과를 지우고 복원 or 막다른 길 벽으로 막기  void DrawBack(int x, int y)  {  Go(y\*2, x);  if(maze[x][y] == WALL) // 해당 좌표가 벽이었으면  {  Color(2);  cout << "■"; // 다시 벽을 그린다.  }  else if(maze[x][y] == ROAD) // 해당 좌표가 길이었으면  {  if(IsNoWay(x, y)) // 막다른 길이면  {  maze[x][y] = WALL; // 벽으로 막기  map[x][y] = WALL; // 벽으로 막기  }  else // 막다른 길이 아니면  cout << "　"; // 다시 길을 그린다.  }  else if(maze[x][y] == START) // 해당 좌표가 시작지점이었으면  {  Color(14);  cout << "ⓢ"; // 다시 시작지점을 그린다.  }  else if(maze[x][y] == END) // 해당 좌표가 도착지점이었으면  {  Color(12);  cout << "¶"; // 다시 도착지점을 그린다.  }  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Go : 콘솔창에서 해당 좌표로 이동한다. (WinAPI)  void Go(int x, int y)  {  COORD pos = {x, y};  SetConsoleCursorPosition(GetStdHandle(STD\_OUTPUT\_HANDLE), pos);  }  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  // Color : 콘솔창의 글자 색을 바꾼다. (WinAPI)  void Color(unsigned short color)  {  HANDLE hCon = GetStdHandle(STD\_OUTPUT\_HANDLE);  SetConsoleTextAttribute(hCon,color);  }  }; |

|  |
| --- |
| **소스 코드 : main.cpp** |
| #include "Maze.h"  void main()  {  CMaze maze;  maze.SearchExit();  getchar();  getchar();  } |

|  |
| --- |
| **결과 화면** |
| **<최초 실행 시>**  **C:\Users\HyeongGeun\Desktop\3.PNG**  **<올바른 파일명을 입력했을 시 미로 출력>**  **C:\Users\HyeongGeun\Desktop\2.PNG**  **<올바르지 않은 파일명을 입력했을 시 에러>**  **C:\Users\HyeongGeun\Desktop\3.PNG**  **<미로 탐색 완료 후 최적화 경로 및 결과 출력>**  **C:\Users\HyeongGeun\Desktop\4.PNG** |