2021-1 C++프로그래밍 실습과제 05

(1) 각 문제에 대한 분석과 및 해결 방법

5.1. 교재 231~232쪽의 영역 채색 프로그램을 구현하고 테스트하라.

(1) 프로그램 5.14를 구현함

(2) 232쪽과 같은 자신만의 그림(image[][])을 만들어 잘 동작하는지 확인할 것

(3) 함수의 매개변수를 2차원 배열(22행과 같이)로 하는 경우의 문제점 또는 불편한 점과 개선할 수 있는 방법을 보고서에 자세히 적을 것. (힌트: 380~383쪽의 내용을 참고할 수 있음)

[문제분석 및 해결방법] : 기존의 코드를 참조하여 재귀적 방법을 이용하여 영역 채색 함수를 구현하고, 본인의 이름 이니셜인 LJH를 images[][] 배열에 표현하여 성공적으로 출력하였다.

5.2. 교재 237~241쪽의 지뢰찾기 게임을 구현하고 테스트하라.

(1) 책에 있는 코드를 구현하고 테스트 할 것.

(2) 243쪽 ~ 244쪽의 고찰 내용 생각해 볼 것. 특히 (4)의 내용에 대해서는 코드에서 사용된 부분을 찾고, 사용 이유를 보고서에 정리할 것.

- **참조형 매개변수와 참조자 변환**
- **인라인 함수와 디폴트 매개변수**
- **재귀호출**
- **정적 함수와 정적 변수**
- **나열형(enum) - bool 반환함수 및 문자처리 함수 toupper()**

[문제분석 및 해결방법] : 기존의 지뢰 찾기 게임 코드를 참조하여 게임을 구현하고, VS2019용 수정 내용을 참고하여 VS2019 환경에서 지뢰 찾기 게임을 구현하였다.

(2) 자신이 구현한 주요 코드

5.1. 교재 231~232쪽의 영역 채색 프로그램을 구현하고 테스트하라.

<LJH_05_1.cpp>

```
int main() {
        unsigned char image[HEIGHT][WIDTH] = {
                9, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 9, 0, 0, 0, 9,
                9, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 9, 0, 0, 0, 9,
                9, 0, 0, 0, 0, 0, 0, 0, 0, 9, 0, 9, 9, 9, 9, 9,
                9, 0, 0, 0, 0, 0, 9, 0, 0, 0, 9, 0, 9, 0, 0, 0, 9,
                9, 9, 9, 9, 9, 0, 9, 9, 9, 9, 0, 9, 0, 0, 0, 9
        };
        printImage(image, "<Original image>");
        blobColoring(image);
        printImage(image, "<Labelled image>");
        return 0;
}
```

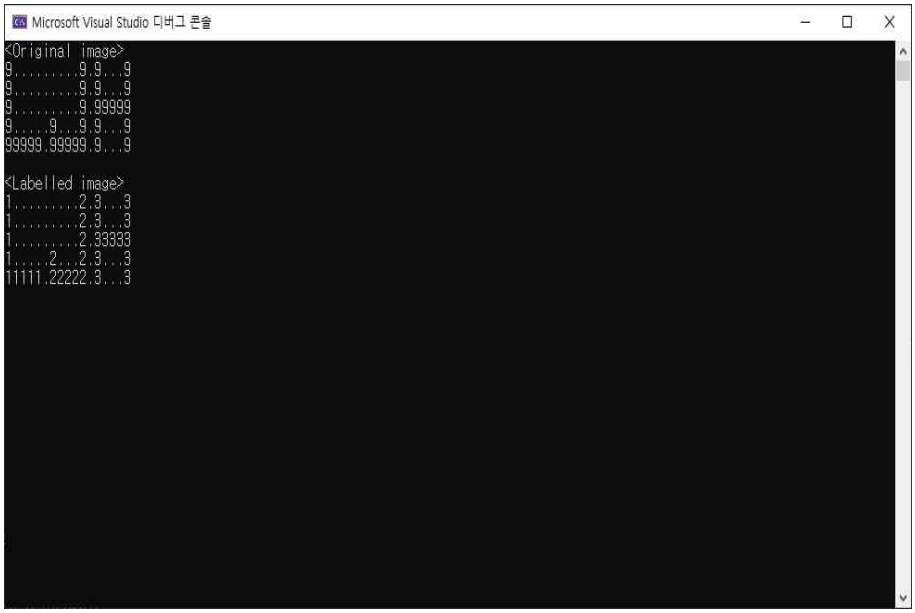
5.2. 교재 237~241쪽의 지뢰찾기 게임을 구현하고 테스트하라.

<LJH_05_2.cpp>

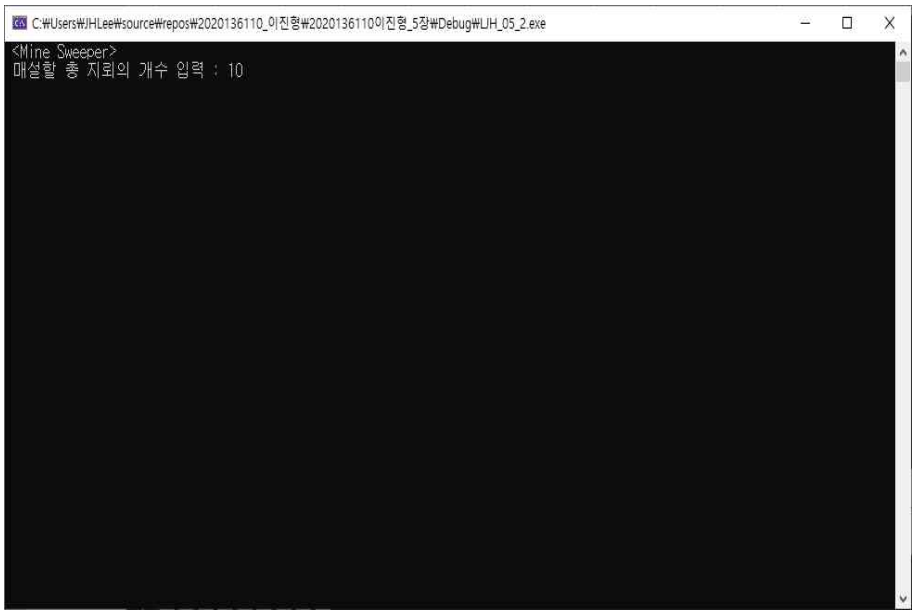
```
#include "MineSweeper.h"
int main() {
        int total;
        printf(" <Mine Sweeper>₩n");
        printf(" 매설할 총 지뢰의 개수 입력 : ");
        scanf("%d", &total);
        playMineSweeper(total);
        return 0;
}
```

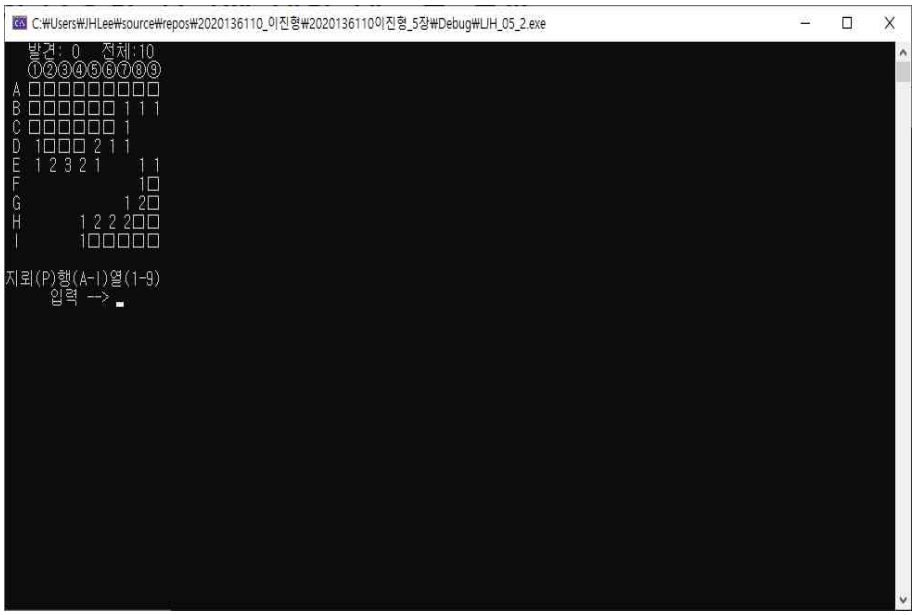
(3) 다양한 입력에 대한 테스트 결과

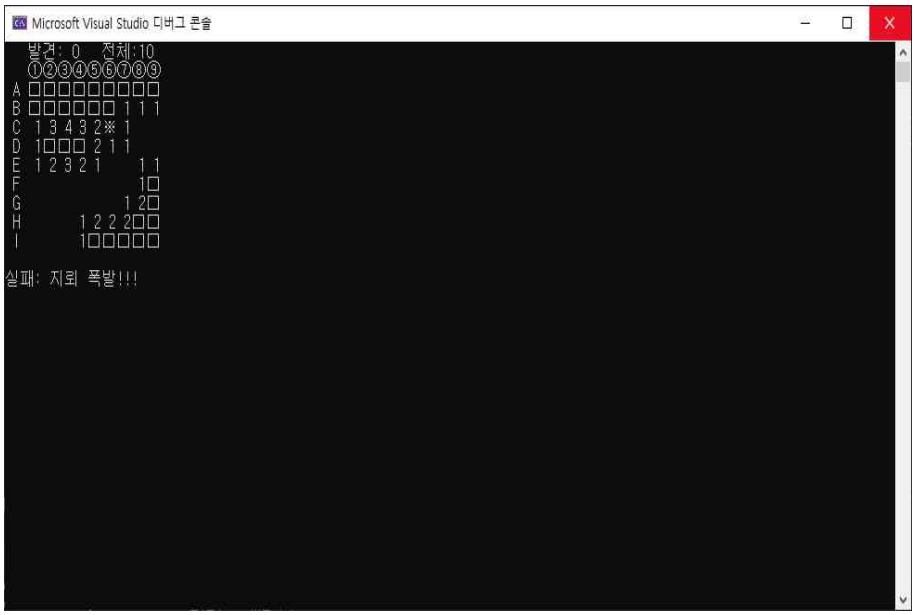
5.1. 교재 231~232쪽의 영역 채색 프로그램을 구현하고 테스트하라.



5.2. 교재 237~241쪽의 지뢰찾기 게임을 구현하고 테스트하라.




C:₩Users₩JHLee₩source₩repos₩2020136110_이진형₩20201361l0이진형_5장₩Debug₩LJH_05_2.exe
발견: 0 전체:10
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
A □□□□□□□□□□
B □□□□□□□ 1 1 1
C □□□□□□□ 1
D 1□□□ 2 1 1
E 1 2 3 2 1 1 1
F 1□
G 1 2□
H 1 2 2 2□□
I 1□□□□□□
지뢰(P)행(A-I)열(1-9)
입력 -->


Microsoft Visual Studio 디버그 콘솔
발견: 0 전체:10
① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨
A □□□□□□□□□□
B □□□□□□□ 1 1 1
C 1 3 4 3 2※ 1
D 1□□□ 2 1 1
E 1 2 3 2 1 1 1
F 1□
G 1 2□
H 1 2 2 2□□
I 1□□□□□□
실패: 지뢰 폭발!!!

(4) 코드에 대한 설명 및 해당 문제에 대한 고찰

5.1-(3). 교재 231~232쪽의 영역 채색 프로그램을 구현하고 테스트하라.
함수의 매개변수를 2차원 배열(22행과 같이)로 하는 경우의 문제점 또는 불편한 점과 개선할 수 있는 방법을 보고서에 자세히 적을 것. (힌트: 380~383쪽의 내용을 참고할 수 있음)

```
#define WIDTH  17
#define HEIGHT 5
void label(unsigned char img[HEIGHT][WIDTH], int x, int y, int color);
void blobColoring(unsigned char img[HEIGHT][WIDTH]);
void printImage(unsigned char img[HEIGHT][WIDTH], const char * msg);
```

함수의 매개변수를 위와 같이 2차원 배열로 두면 컴파일 시점에 매개 변수가 설정한 고정된 배열의 길이를 가지게 되므로 런타임 시점에서 배열의 길이를 수정할 수 없으며, 이에 따라서 고정된 길이 이상의 값을 배열에 넣어 사용할 수 없다. 이는 프로그램의 확장 가능한 설계와 유지 보수성을 저해할 수 있는 사용 방식이다.
이를 개선하기 위한 방법으로 동적 할당을 사용할 수 있다.

```
int ** alloc2DInt(int rows, int cols) {
        int ** mat =new int *[rows];
        for(int i =0; i < rows; i ++){
                mat[i] =new int[cols];
        }
        return mat;
}
void free2DInt(int **mat, int rows, int cols =0){
        for(int i =0; i < rows; i ++){
                delete [] mat[i];
        }
        delete [] mat;
}
```

(C++의 new 키워드를 이용한 2차원 배열 동적 할당)

```
void **malloc_double_pointer(int type_size, int width, int height)
{
    void **grid =malloc(sizeof(void *) * width);
    if (grid ==NULL)
    {
        return NULL;
    }
    for (int i =0; i < width; i ++)
    {
        grid[i] =malloc(type_size * height);
        if (grid[i] ==NULL)
        {
            return NULL;
        }
    }
    return grid;
}
void free_double_pointer(void **grid, int width)
{
    for (int x =0; x < width; x ++)
    {
        free(grid[x]);
    }
    free(grid);
}
```

(malloc, free 함수를 이용한 2차원 배열 동적 할당)
위와 같은 방식으로 이중 포인터를 이용하여 2차원 배열을 동적 할당하면 포인터만 함수에 매개변수로 넘겨주면 2중 포인터에 첨자 연산자(Subscript Operator)를 이용하여 기존의 2차원 배열과 똑같은 방식으로 요소에 접근할 수 있다.

```
#define MATRIX_COL 256
int to_2d_y(int index) {
    return index / MATRIX_COL;
}
int to_2d_x(int index) {
    return index % MATRIX_COL;
}
int to_2d_index(x, y) {
    return y * MATRIX_COL + x;
}
// array[to_2d_index(x, y)]
```

기존의 2차원 배열을 똑같은 방식으로 2개의 첨자 연산자를 통해 사용할 수는 없지만, 위의 코드와 같이 1차원 배열을 동적 할당한 뒤 임의의 색인 방식을 만들어서 단일 포인터 1차원 배열을 2차원 배열과 유사하게 사용할 수 있다. 이 방식은 복잡한 동적 할당 없이 사용할 수 있는 장점과 배열의 차원이 늘어날 때마다 복잡해지는 동적 할당/해제 코드를 작성하는 대신 색인 생성 함수만을 조정하면 되는 장점이 있다.

5.2-(2). 교재 237~241쪽의 지뢰찾기 게임을 구현하고 테스트하라.
243쪽 ~ 244쪽의 고찰 내용 생각해 볼 것. 특히 (4)의 내용에 대해서는 코드에서 사용된 부분을 찾고, 사용 이유를

보고서에 정리할 것.

- **참조형 매개변수와 참조자 변환**

```
inline int & mask(int x, int y) {
  return MineMapMask[y][x];
}
inline int & label(int x, int y) {
  return MineMapLabel[y][x];
}
static void mark(int x, int y) {
  if (isValid(x, y) && mask(x, y) == Hide) {
    mask(x, y) = Flag;
  }
}
static void init(int total =9) {
  srand((unsigned int)time(NULL));
  for (int y =0; y < ny; y ++)
    for (int x =0; x < nx; x ++) {
      mask(x, y) = Hide;
      label(x, y) =0;
    }
  nBomb = total;
  for (int i =0; i < nBomb; i ++) {
    int x, y;
    do {
      x = rand() % nx;
      y = rand() % ny;
    } while (label(x, y) != Empty);
    label(x, y) = Bomb;
  }
  for (int y =0; y < ny; y ++) {
    for (int x =0; x < nx; x ++) {
      if (label(x, y) == Empty) {
        label(x, y) = countNbrBombs(x, y);
      }
    }
  }
}
static bool getPos(int & x, int & y) {
  printf("₩n지뢰(P)행(A-I)열(1-9)₩n        입력 --> ");
  bool isBomb =false;
  y = toupper(getch()) -'A';
  if (y =='P'-'A') {
    isBomb =true;
    y = toupper(getche()) -'A';
  }
  x = getch() -'1';
  return isBomb;
}
```

포인터에 비해 &나 * 연산자 없이 변수를 참조하는 것이 가능하고(Call by reference), 포인터를 이용하다 잘못된 메모리를 수정하는 것을 방지할 수 있다.

- **인라인 함수와 디폴트 매개변수**

```
inline int & mask(int x, int y) {
  return MineMapMask[y][x];
}
inline int & label(int x, int y) {
  return MineMapLabel[y][x];
}
inline bool  isValid(int x, int y) {
  return (x >=0 && x < nx && y >=0 && y < ny);
}
inline bool  isBomb(int x, int y) {
  return isValid(x, y) && label(x, y) == Bomb;
}
inline bool  isEmpty(int x, int y) {
  return isValid(x, y) && label(x, y) == Empty;
}
static void init(int total =9) {
  srand((unsigned int)time(NULL));
  for (int y =0; y < ny; y ++)
    for (int x =0; x < nx; x ++) {
      mask(x, y) = Hide;
      label(x, y) =0;
    }
  nBomb = total;
  for (int i =0; i < nBomb; i ++) {
    int x, y;
    do {
      x = rand() % nx;
      y = rand() % ny;
    } while (label(x, y) != Empty);
    label(x, y) = Bomb;
  }
  for (int y =0; y < ny; y ++) {
    for (int x =0; x < nx; x ++) {
      if (label(x, y) == Empty) {
        label(x, y) = countNbrBombs(x, y);
      }
    }
  }
}
```

간단한 처리 함수를 인라인 함수로 선언하면 컴파일 시점에 함수가 아닌 내부 코드로 인라인 함수 내부 구문이 바뀌어 컴파일되기 때문에 함수 호출로 발생하는 오버 헤드없이 빠른 속도로 해당 부분을 실행할 수 있다.

기본 매개변수 값을 지정하면 외부에서 해당 함수를 호출 시 자주 사용하는 상수 등을 매개변수로 주지 않아도 되는 장점이 있고, 필요에 따라서 매개변수에 값을 넘길 수도 있기 때문에, 해당 함수의 편리하고 유용한 사용이 가능해진다.

- **재귀호출**

```
static void dig(int x, int y) {
  if (isValid(x, y) && mask(x, y) != Open) {
    mask(x, y) = Open;
    if (label(x, y) ==0) {
      dig(x -1, y -1);
      dig(x -1, y);
      dig(x -1, y +1);
      dig(x, y -1);
      dig(x, y +1);
      dig(x +1, y -1);
      dig(x +1, y);
      dig(x +1, y +1);
    }
  }
}
```

재귀적으로 표현이 가능한 알고리즘에서 재귀 함수를 이용하여 해당 알고리즘을 표현하면 코드가 자연스러워지고 가독성이 높아질 가능성이 크고, 변수 사용을 줄여주는 이점도 있다.

- **정적 함수와 정적 변수**

```
static int  MineMapMask[DIM][DIM];
static int  MineMapLabel[DIM][DIM];
static int  nx = DIM, ny = DIM;
static int  nBomb = DIM;
static void dig(int x, int y);
static void mark(int x, int y);
static int getBombCount();
static void print();
static int countNbrBombs(int x, int y);
static void init(int total =9);
static bool getPos(int & x, int & y);
static int checkDone();
```

명시적으로 해당 파일에서만 사용하는 함수/변수임을 나타내어 외부 링킹 등을 방지하고, 함수/변수등을 내부에서 사용되도록 은닉할 수 있다.

- **나열형(enum) - bool 반환함수 및 문자처리 함수 toupper()**

```
enum LabelType { Empty =0, Bomb =9 };
enum MaskType { Hide =0, Open, Flag };
static bool getPos(int & x, int & y) {
  printf("₩n지뢰(P)행(A-I)열(1-9)₩n       입력 --> ");
  bool isBomb =false;
  y = toupper(getch()) -'A';
  if (y =='P'-'A') {
    isBomb =true;
    y = toupper(getche()) -'A';
  }
  x = getch() -'1';
  return isBomb;
}
```

나열형: 매크로 상수나, 일반적인 상수 정의를 이용한 상수 정의 방법보다 가독성 있고 비슷한 상수들을 묶어서 정의할 수 있다.

bool 반환 함수: 조건문과 결합하여 복잡한 조건을 검사하는데 편리하게 사용할 수 있다.

toupper(): **'%소문자%' + ('A' - 'a')**와 같은 방법 대신 손쉽게 소문자를 대문자로 변환할 수 있다. 가독성 면에서도 뛰어나다.

(5) 이번 과제에 대한 느낀점

이번 과제를 해결하기 위해 교재의 코드를 분석하고 조사하면서 모르고 있었던 세부적인 내용을 많이 알 수 있어서 좋았다.