

고급 프로그래밍 과제 #1: 생명 게임

과제 출제일: 5월 10일 (수)

제출 기한: 5월 27일 (토) 23시 59분까지

소개: 영국의 수학자 존 콘웨이(John H. Conway)는 사각형 칸으로 구성된 2차원 격자 안에서 생명이 탄생하고 소멸되는 과정을 모사하는 '생명 게임'(Game of Life)을 제안하였다 (그림 1). '게임'이라는 단어 때문에 참여자 간의 경쟁에 의하여 진행되는 일반적인 게임으로 오해할 수 있지만, 생명 게임은 사람의 개입 없이 간단한 규칙에 의하여 스스로 진행되는 일종의 시뮬레이션이다. 격자 안의 각 칸은 '삶' 혹은 '죽음'의 두 가지 상태만 가질 수 있고, 매 순간 자신의 상태 및 자신을 둘러싼 주위 8개 칸의 상태에 아래 규칙을 적용하여 다음 순간의 상태가 결정된다.

- (1) 자신이 현재 살아있고, 주위에 1개 이하의 칸만 살아있다면, 죽는다. (인구 부족)
- (2) 자신이 현재 살아있고, 주위에 4개 이상의 칸이 살아있다면, 죽는다. (인구 과밀)
- (3) 자신이 현재 살아있고, 주위에 2~3개의 칸이 살아있다면, 계속 산다.
- (4) 자신이 현재 죽어있고, 주위에 정확히 3개의 칸이 살아있다면, 살아난다.

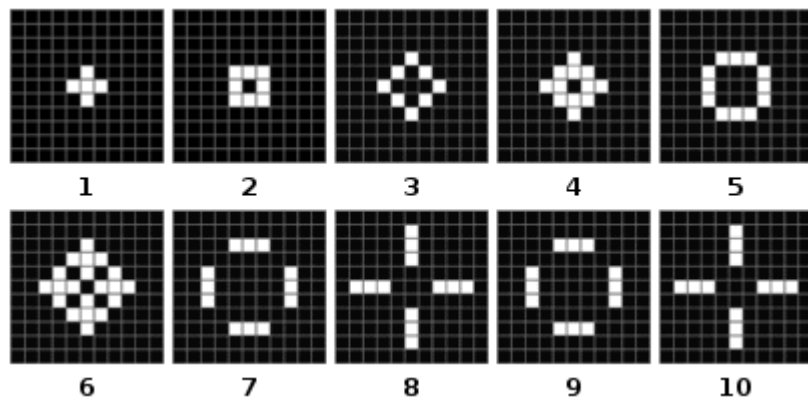


그림 1. 생명 게임의 순차적 진행 과정. 흰색 칸은 살아있는 상태, 검은색 칸은 죽은 상태를 표현한다.

초기에 살아있는 칸들이 이루는 형태에 따라서 이후 복잡다단한 양상으로 확장될 수도 있고, 동일한 형태가 반복적으로 나타날 수도 있으며, 어느 정도 변화를 거치다가 소멸되기도 한다. 표 1은 몇 가지 특징적인 초기 형태를 보이고 있다.

보트 (Boat, 형태 불변)	
------------------	--



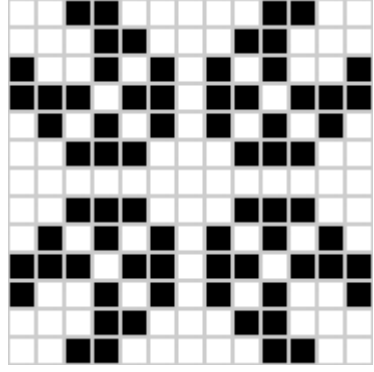

토드 (Toad, 2 단계 진동)	
글라이더 (Glider, 이동)	
펄사 (Pulsar, 3 단계 진동)	
다이하드 (Diehard, 한참 후 소멸)	

표 1. 생명 게임의 다양한 변화 양상을 유도하는 특징적인 초기 형태들. 그림 1과 달리 검은색 칸이 살아있는 상태, 흰색 칸이 죽은 상태에 해당된다.

구현: 본 과제는 임의 크기의 2차원 격자에서 생명 게임을 진행하고 그 과정을 화면에 출력하는 C++ 프로그램의 작성을 목표로 한다. 프로그램의 main() 함수는 출제자가 제공하고, 수강생은 생명 게임의 핵심 기능을 담당하는 LifeGame 클래스가 의도대로 사용될 수 있도록 선언하고 정의하여야 한다. 구체적으로 LifeGame 클래스는 최소한 다음의 멤버 변수/함수 및 friend 함수를 가져야 한다.

(1) 멤버 변수

■ int width;

▷ 격자의 너비.

■ int height;

▷ 격자의 높이.

■ bool* current_cells;

▷ 격자에 속한 각 칸의 현재 상태('살'은 true, '죽음'은 false)를 저장하는 배열의 첫 번째 원소를 가리키는 포인터. 임의 크기의 격자를 생성하기 위하여 배열은 동적 할당하도록 한다. 격자의 i번째 행, j번째 열에 해당하는 칸의 상태는 $*(current_cells + i*width + j)$ 로 접근 가능하다.

■ bool* next_cells;

▷ 격자에 속한 각 칸의 다음 상태('살'은 true, '죽음'은 false)를 저장하는 배열의 첫 번째 원소를 가리키는 포인터로서, current_cells와 유사함. (주의 2 참고.)

(2) 멤버 함수

■ LifeGame();

▷ 기본 생성자 (default constructor). 멤버 변수의 값을 초기화한다.

■ ~LifeGame();

▷ 소멸자 (destructor). 동적 할당된 메모리를 해제한다.

■ void initialize(int w, int h);

▷ 주어진 너비(w)와 높이(h)의 격자를 저장하기 위한 bool 타입 배열을 동적 할당하여 멤버 변수 current_cells가 가리키도록 하고, 모든 칸의 상태를 false로 초기화한다. 멤버 변수 width와 height도 각각 w와 h의 값을 할당해주어야 한다. (주의: 이전에 이미 동적 할당이 이루어졌을 경우, 기존 메모리를 해제한 후 다시 할당하여야 함.)

■ int getWidth() const;

▷ 멤버 변수 width의 값을 반환한다.

■ int getHeight() const;

▷ 멤버 변수 height의 값을 반환한다.

■ void setState(int i, int j, bool s);

▷ 격자의 (i, j) 위치에 해당하는 칸의 상태를 주어진 인자 s로 설정한다.

■ bool getState(int i, int j) const;

▷ 격자의 (i, j) 위치에 해당하는 칸의 상태를 반환한다.

■ void clear();

▷ 모든 칸의 상태를 false로 설정한다.

■ void update();

▷ 모든 칸의 상태를 생명 게임 규칙에 맞게 다음 상태로 전환한다.

(3) friend 함수

■ ostream& operator <<(ostream& os, const LifeGame& game);

▷ cout을 이용한 출력이 가능하도록 << 연산자를 오버로딩한다.

예) 너비 5, 높이 5의 격자에 표 1의 보트(boat) 형태가 저장되어 있을 경우, cout << "The current game state is: \n" << game; 는 다음과 같은 결과를 출력하여야 한다.

The current game state is:

```
. . . . .  
. 0 0 . .  
. 0 . 0 .  
. . 0 . .  
. . . . .
```

(칸과 칸 사이는 공백 문자 한 개 삽입, 죽은 칸은 '.', 살아있는 칸은 'o'를 출력.)

결과: 함께 첨부하는 실행파일 참고.

주의 1: 이상적인 생명 게임은 무한한 크기의 격자를 가정하고 있지만, 컴퓨터 프로그램으로 구현 시에는 제한된 크기의 격자를 사용할 수밖에 없다. 따라서 일부 칸은 격자의 경계 영역에 놓여 이를 둘러싼 주위 칸의 개수가 8보다 적어진다. 본 과제에서는 격자의 왼쪽 경계와 오른쪽 경계, 그리고 위쪽 경계와 아래쪽 경계가 서로 맞닿아 있다고 가정하여 이 문제를 해결하도록 한다. 예를 들어, 너비 5, 높이 5의 격자에서 (0,0)의 위치에 놓인 칸은 (1, 0), (0, 1), (1, 1)뿐 아니라 (4, 0), (4, 1), (0, 4), (1, 4), (4, 4)에 해당하는 칸에 의하여 둘러싸여 있는 것으로 처리한다 (그림 2 오른쪽).

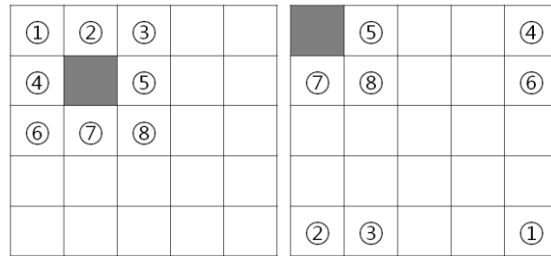


그림 2. (왼쪽) 격자 내부에 속한 칸의 이웃 8개 칸. (오른쪽) 경계에 놓인 칸의 이웃 8개 칸.

주의 2: update() 함수를 구현할 때, 모든 칸은 자신을 둘러싼 칸들의 '현재' 상태에 따라 다음 상태를 결정하여야 한다. 흔히 저지르기 쉬운 실수 중 하나는, 각 칸을 순차적으로 방문하면서 곧바로 상태 값을 갱신하는 것이다. 이 경우 나중에 방문한 칸의 이웃 중에는 이미 '다음' 상태로 갱신된 칸이 있을 수 있기 때문에 올바른 결과를 얻을 수 없게 된다 (그림 3). 이와 같은 '순서 의존성'을 해결하는 간단한 방법은 먼저 모든 칸의 '현재' 상태를 다른 메모리 영역에 보관하여 두는 것이다. 각 칸의 다음 상태를 결정할 때는 이와 같이 보관된 값을 참조하도록 함으로써 갱신 순서와 상관없이 올바른 결과를 얻을 수 있다.

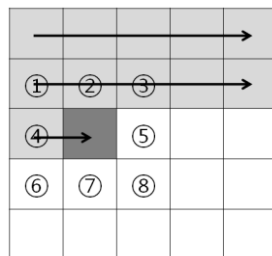


그림 3. 곧바로 상태 값을 갱신하면 이웃 8개 칸 중에서 이미 방문한 칸(열은 회색 영역의 ①②③④)은 다음 상태, 아직 방문하지 않은 칸(흰색 영역의 ⑤⑥⑦⑧)은 현재 상태를 갖게 되는 문제가 발생한다.

제출: 주석 문을 적절히 작성한 코드(HW1_학번.cpp)와 실행 파일(HW1_학번.exe), 그리고 보고서 파일(HW1_학번.docx/hwp/pdf, 보고서는 자유 형식이지만 자신이 실행한 실행 예와 고찰 부분이 반드시 포함되어 있도록 합니다. 보고서도 평가 대상이니 최대한 잘 작성하길 바랍니다.)을 zip으로 묶어 유-캠퍼스를 통해 제출합니다. (제출 파일의 이름은 반드시 HW1_학번.zip으로 합니다.) 코드 내부에는 자신의 학번과 이름이 주석 문 형식으로 코드 맨 상단에 위치해야 합니다. 작성한 코드는 반드시 Visual Studio 2015 혹은 그 이상의 최신 버전에서 빌드 및 실행 가능해야 합니다.

감점 규정: 최대 하루까지의 연장만을 허용합니다. 본래 마감일보다 하루 늦게 제출할 경우 50%의 감점이 있고, 그 이후에는 제출한다 하여도 0점 처리됩니다. 코드에 C언어 입출력 함수인 printf(), scanf() 함수나 동적할당 함수인 malloc(), free() 함수 등을 사용할 경우 0점 처리됩니다. 수업 시간에 배운 코딩 convention을 지키지 않을 경우 감점될 수 있습니다. 출제자가 제공한 코드의 틀(각종 함수 포함)은 절대로 수정하여서는 안됩니다.

[점수를 제대로 받기 위해 제출하기 전 반드시 체크할 사항]

1. 자신이 작성한 코드가 Visual Studio 2015/2017에서 실행 가능한가?
2. 동적 메모리 할당과 반환을 위하여 new와 delete를 이용하였는가?
3. 첨부한 예제 실행파일과 동일한 결과를 출력하는가?
4. 제출할 파일 이름을 지시된 대로 만들었는가?
5. 코드에 주석 문은 적당히 작성하였는가?
6. 수업 시간에 배운 코딩 convention을 지켰는가? (예, 변수 이름 표기법, 전역 변수 사용 자제, 메모리 할당 여부 확인, 메모리 반환, public과 private 영역의 구분 등)