ソフト系　C言語実習課題 1
ライフゲームの作成

V3.07

本課題は、3つのSetpに分けて実習します

- 実習を通じて、Stepごとに下記のことを学びます。
  - Step1: ライフゲームのルールを理解する
    - 課題の理解し、どのような関数が必要かを理解する
  - Step2: 課題を理解し、4つの必要な関数を作成する。
    - (0) main ➔ 2次元配列のマップの定義と各関数を呼び出す
    - (1) 初期化 ➔ 乱数を発生させて、マップを初期化する。
    - (2) マップの表示 ➔ 2次元配列の内容を表示する
    - (3) 世代を進める ➔ 自分の回りの生存数を調べて、世代を進める
    - (4) 生存数を調べる ➔ 指定されたセルの回り8セルの生存数を調べて返す
    - Step2-1: まず、「(1) 初期化」と、「 (2) マップの表示」を作成する
      - mainで、2次元配列mapを定義して、初期化とマップの表示を呼び出す
    - Step2-2: 次に、「(3) 世代を進める」と「(4) 生存数を調べる」を作成する
      - (4)生存数を調べるは、(3)世代を進めるから呼び出します
  - Step3: ファイルから初期パターンを読み込み初期化する
    - ファイルを開いてテキストデータを読み込む
  - Step4:文字列(テキスト)を入力して、乱数モードとファイルモードを切り替える
    - 入力した文字の最初の1文字が0～9までの数字なら乱数モード、それ以外はファイルモードで動作

ファイルの読み込みを学ぶ

Step1: ライフゲームとは

ライフゲームの詳細と実際の様子は、

「ライフゲーム」をGoogleで検索、Wikipediaで見てください。

https://ja.wikipedia.org/wiki/%E3%83%A9%E3%82%A4%E3%83%95%E3%82%B2%E3%83%BC%E3%83%A0

上記ページの中程の右に、「グライダー銃」というパターンの動作している画面があります。

このパターンは、後述のsample_data2.txt で初期化をすると表示されます。

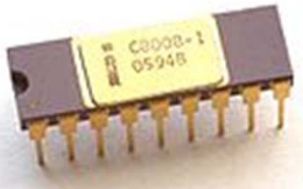
ライフゲームとマイクロコンピュータの歴史

- ライフゲームは1970年にイギリスの数学者ジョン・ホートン・コンウェイ が考案した生命の誕生、進化、絶滅のプロセスを簡易的なモデルで再現したシミュレーションゲームである。
- 単純なルールでその模様の変化を楽しめるため、パズルの要素を持っている。
- しかし、当時はコンピュータが高価なため、誰もが簡単に体験できるものではなかった。

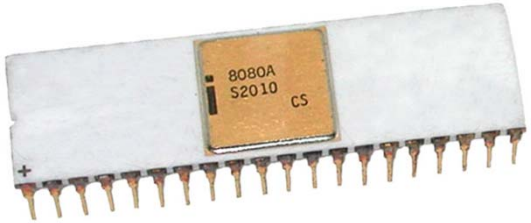
- 1971年に米国インテル社から世界初のマイクロプロセッサ「4004」が発売された。
- しかし、4004は4ビットのCPUで、それほど利用されることはなかった。

4004の概観

- 翌年(1972年)に、インテルは、世界初の8ビットCPUである「8008」を発売したが、
- アドレス空間が14ビット(16KB)しかなく、4004同様にそれほど利用されなかった。


8008の概観

- 1974年に、インテルからアドレス16ビット、データ8ビットが完全に独立した「8080」が発売された。
- 8080の発売で様々なボードコンピュータが登場し、ライフゲームが手軽に使えるようになった。
- この当時のゲームというと、キャラクタベースのライフゲームとスタートレックであった。



8080の概観

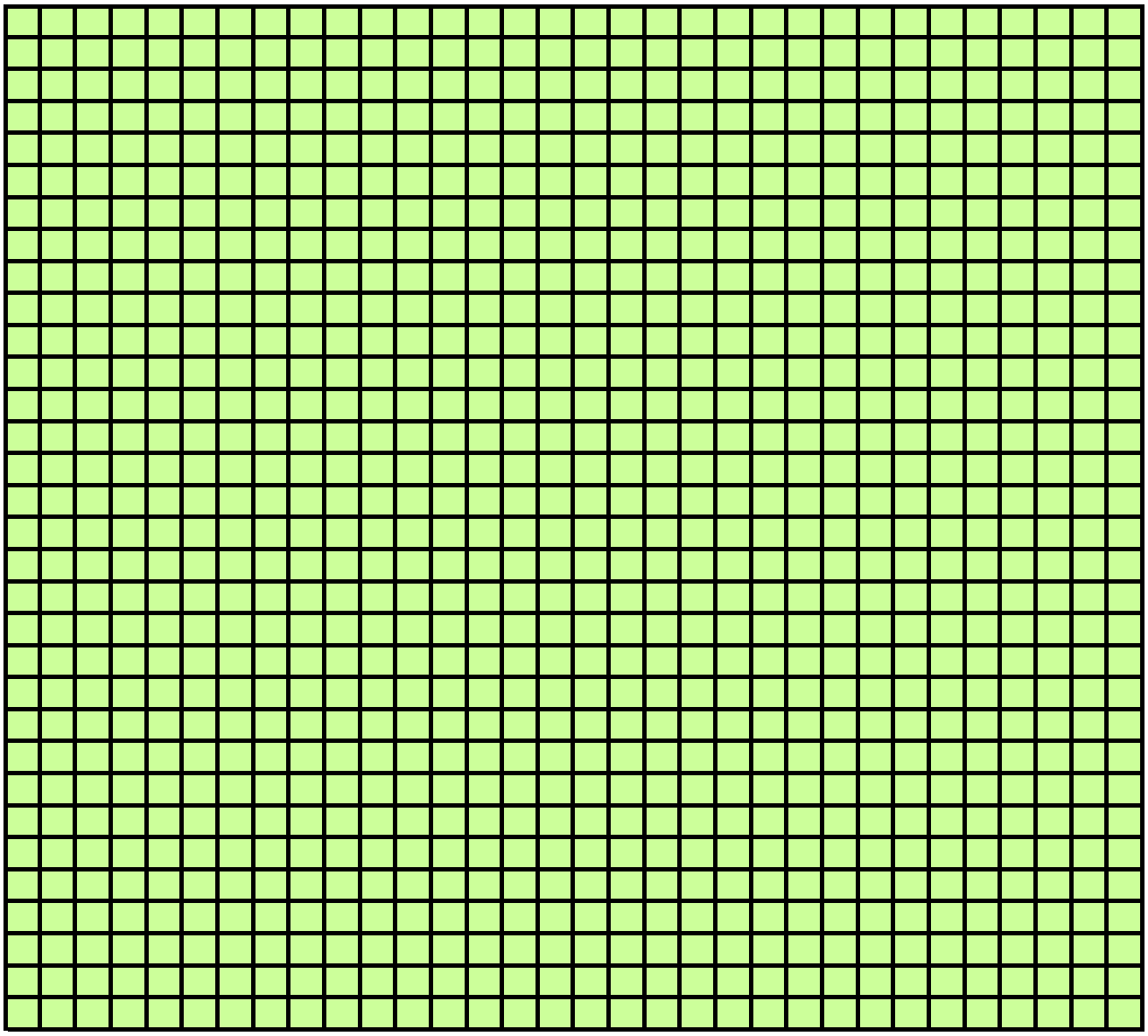
ライフゲームとは?

- 生命の誕生、進化、絶滅のプロセスを簡易的なモデルで再現したシミュレーションで、ルールはシンプル。
- ある領域(碁盤の目を想定)の中に生物が存在するが、下記のルールで、生物の誕生、生存、過疎、過密が繰り返される。
  - **誕生: 次の世代に新たに生命が誕生する**
    - 死んでいるセルに隣接する生きたセルがちょうど3つあれば、次の世代が誕生する。
  - **生存: 現状が維持される**
    - 生きているセルに隣接する生きたセルが2つか3つならば、次の世代でも生存する。
  - **過疎: 過疎過ぎて繁栄できず死滅する**
    - 生きているセルに隣接する生きたセルが1つ以下ならば、過疎により死滅する。
  - **過密: 過密過ぎて繁栄できず死滅する**
    - 生きているセルに隣接する生きたセルが4つ以上ならば、過密により死滅する。
- 自然界においても、上記の関係は成り立つ
  - 肉食動物 ⇔ 草食動物 ⇔ 草木 ⇔ 昆虫

参照URL: https://ja.wikipedia.org/wiki/%E3%83%A9%E3%82%A4%E3%83%95%E3%82%B2%E3%83%BC%E3%83%A0

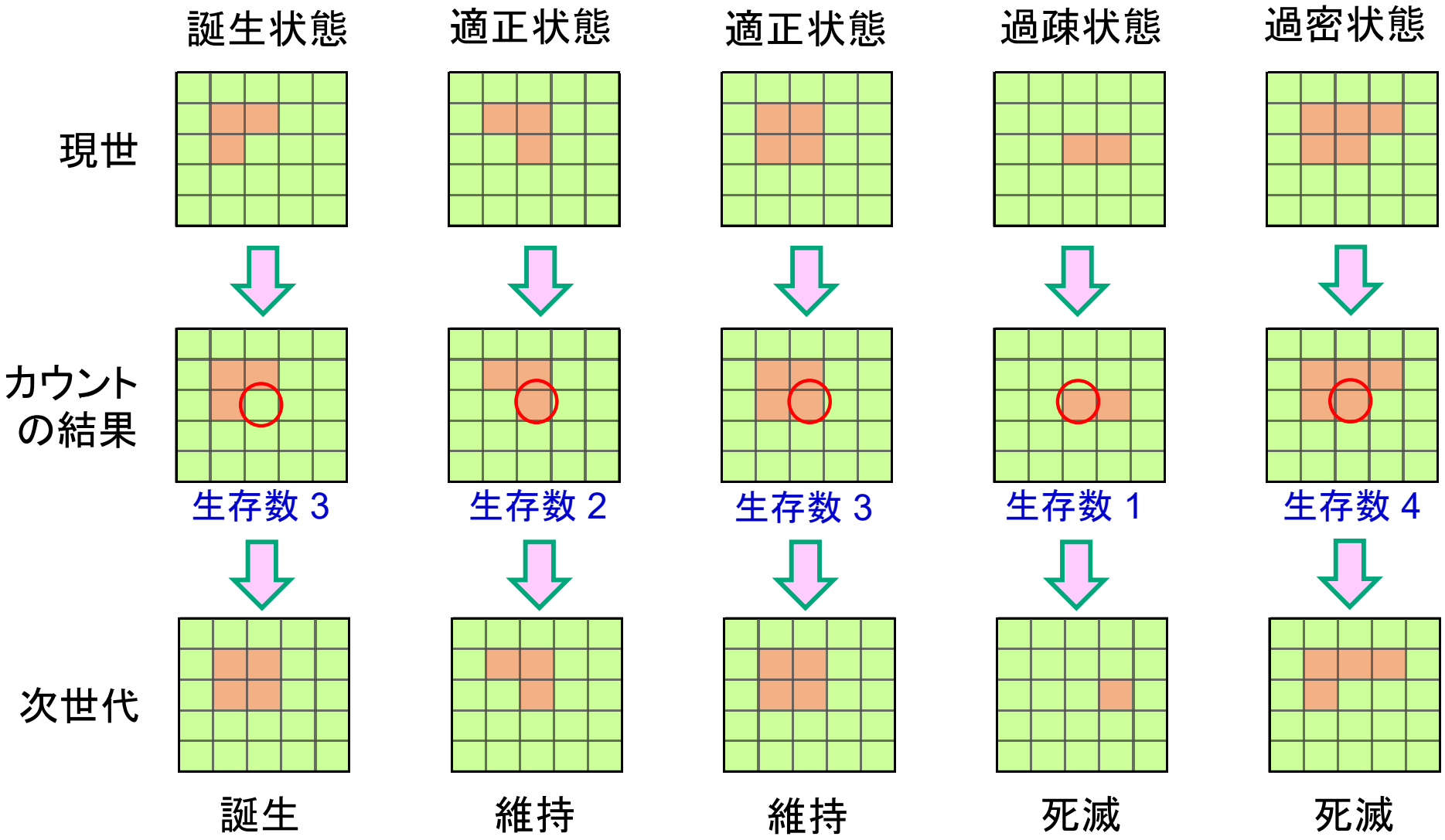
38×38の2次元配列(マップ)

この領域をマップ(地図)といい、1つ1つのマス目はセルという



実際の状態変化

下記は、ライフゲームのルールに従った、実際の変化の様子である。
例えば、5×5の25マスの中心位置のセルを調べたとき状況では、



Step2: 4つの必要な関数を作成する

課題と条件

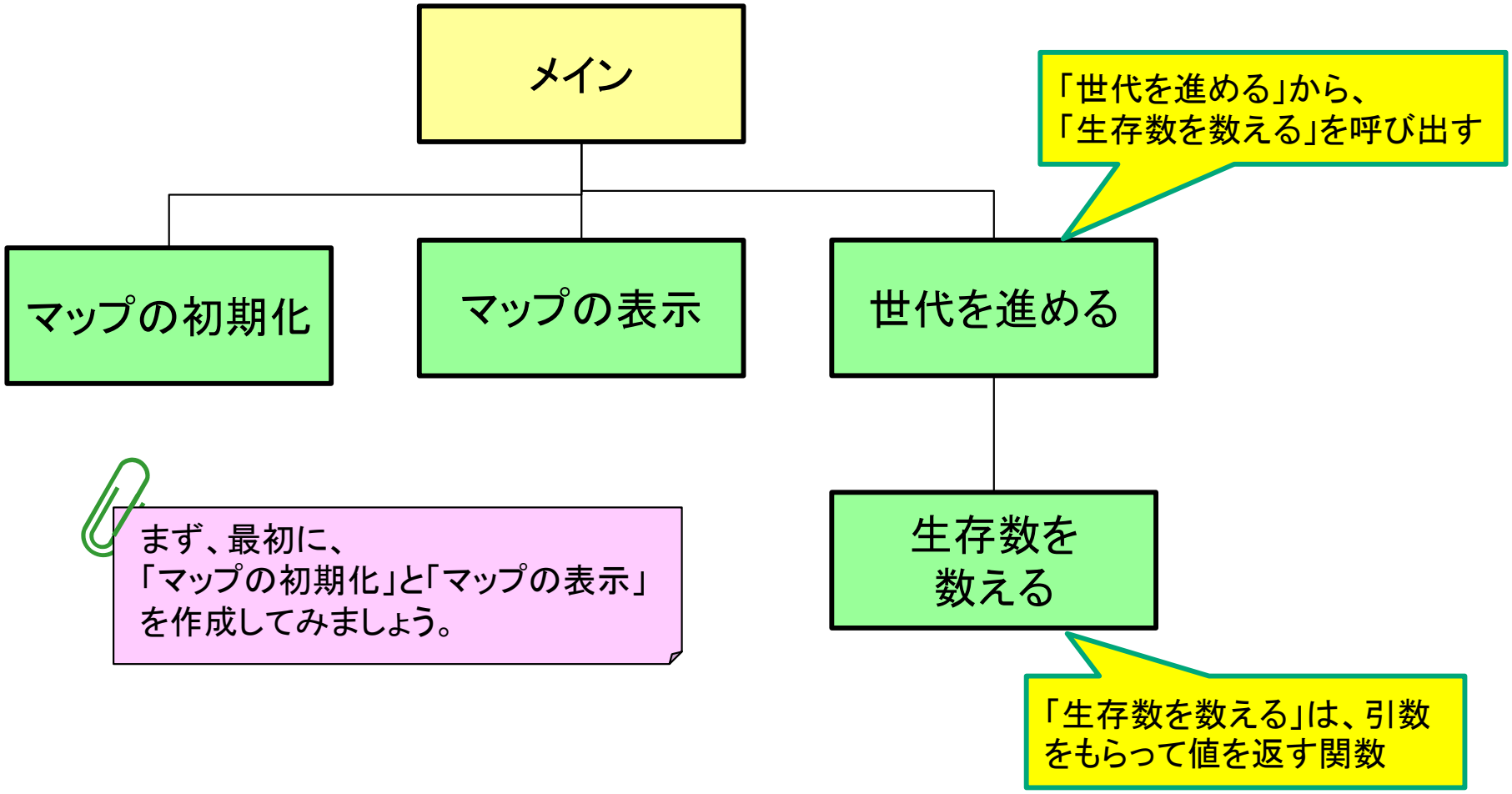
- 課題: 下記の条件でライフゲームを作成する
- 条件:
  - 領域(マップ)は、main関数の中で定義する
  - 領域(マップ)の高さと幅は、定数で指定できるようにする。
    - 初期値は、縦38×横76とする。
  - 世代を進める
  - 23/3の標準ルールで作成する。
    - 誕生: 死んでいるセルに隣接する生きたセルがちょうど3つあれば、次の世代が誕生する。
    - 生存: 生きているセルに隣接する生きたセルが2つか3つならば、次の世代でも生存する。
    - 過疎: 生きているセルに隣接する生きたセルが1つ以下ならば、過疎により死滅する。
    - 過密: 生きているセルに隣接する生きたセルが4つ以上ならば、過密により死滅する。

Wikipediaのライフゲームの項目を参考にする

23/3ルールとは:
周囲に3つの隣人がいれば生命が誕生し、1以下か4以上の隣人であれば死滅し、2つあるいは3つの隣人であれば現状維持するというルールを、標準のライフゲームのルール 23/3と表します。

プログラム構造

- ライフゲームのプログラム構造の例
  - メイン関数の他に、下記の4個の関数で構成される



4つの必要な関数を作成

```
int main(void){
        char map[WORLD_H][WORLD_W];
        printf("¥033[2J");              // 画面を消す
        初期化(map);
        for(gen=1; gen<1000; gen++){
                printf("¥033[2;1H");   //カーソル位置を、高さ2行目、横1文字目に移動
                printf(“世代=%d¥n”,gen);
                マップを表示( map);
                世代を進める(map ):
        }
}
```

もし、printf(“¥033[..)を実行して、画面に [2Jとか、[2;1H と言う文字が表示されたらこの方法は使えません。
本テキストの最後にある「Windowsによる画面制御」を見てください。

初期化 {
　初期のマップを設定する。Step2では乱数(rand関数)を使って初期化する(1と0の比率は、7:3程度がよい)。
　Step3では、ファイルからデータを読み込んで初期化する。
}

マップを表示{
　2次元配列のマップを表示。1は”@”、0(else)は”.”で表示すると良い
}

世代を進める {
　①マップのコピー　← 世代を進めるために、tempマップに現在のマップをコピー。
　②tempマップを調べて、結果をmapに反映して、1世代進める
}

指定されたセルの周囲の生きているセルを返す{
　return 生きているセルの数
}

プログラム作成のヒント

- mapには、「0」か「1」の2値が入るので、下記のようにchar型の2次元配列を定義します

  char map[縦][横];

- ただし、配列の大きさは、縦、横とも定数で定義し、後で大きさを変更できるようにする。

  #define WORLD_H 38 // 縦の大きさ
  #define WORLD_W 76 // 横の大きさ

  例: char map[WORLD_H][WORLD_W]

- 配列の中(セル)は、0(死んでいる)と1(生きている)の値を持つようにする。
- 配列の定義は、グローバル変数にはしないこと(mainの中で定義)。
- 画面の制御(画面クリアとカーソル移動)
  - 前ページの例にあるように、下記のprintf( )で画面制御を実行した場合、

    printf("¥033[2J"); // 画面を消す
    printf("¥033[2;1H"); //カーソル位置を、高さ2行目、横1文字目に移動

  もし、画面に [2Jとか、[2;1H と言う文字が表示される場合は、この方法は使えません。
  本テキストの最後にある「Windowsによる画面制御」の方法を使ってください。。

初期化関数のヒント

- ライフゲームは、2次元配列を使います
- そのため、どちらも、2次元配列のアクセスの構造になります
  - 2次元配列は、forの2重ループで構成されます

```
char map[WORLD_H][WORLD_W];                          // 2次元配列の定義

                                  2次元配列は、mainの中で定義


// 初期化ルーチン
for (ypos = 0; ypos < WORLD_H; ypos++){                    //縦のループ
      for (xpos = 0; xpos < WORLD_W ; xpos++) {            //横のループ
/*
          乱数を使って、map[ypos][xpos] に、0か1を入れる
          参考例: 乱数を10で割った余りを求め、
                  if( )文で、7より小さければ、0 を、それ以外であれば 1 を入れる
*/
   }
}
```

表示関数のヒント

```
// mapの表示ルーチン

for (ypos = 0; ypos < WORLD_H; ypos++){          //縦のループ
     for (xpos = 0; xpos < WORLD_W; xpos++) {  //横のループ
/*
        map[ypos][xpos]の値が、0なら、“.”を表示
                                0以外なら、“@”を表示
*/
   }
}
```

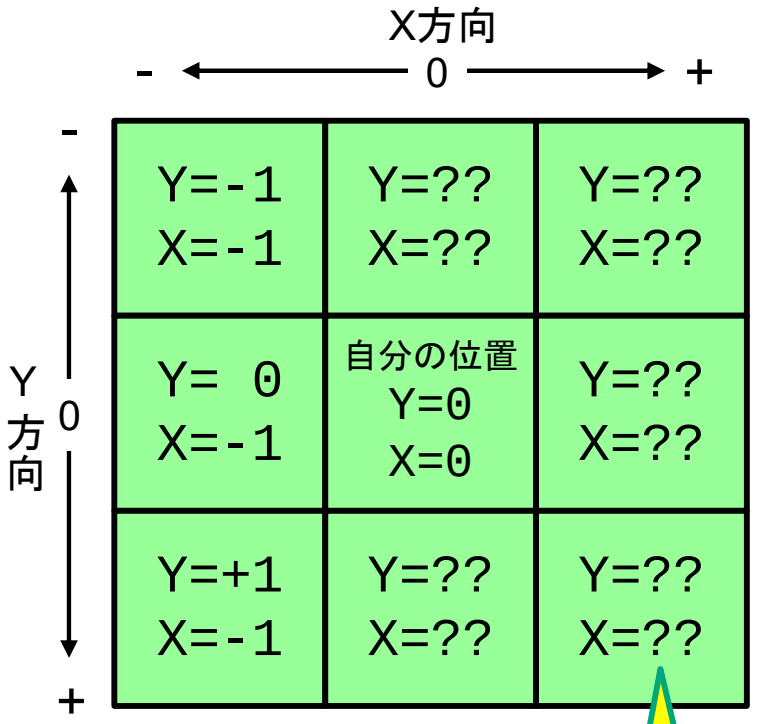
If( )文で判定

自分の周り8個の生存数を数える考え方

- まず、自分の周り8個を数える式を書く

```
int sum=0;      // 0クリア

sum += temp[ypos-1][xpos-1];    左上
sum += temp[ypos  ][xpos-1]; // 左
sum += temp[ypos+1][xpos-1]; // 左下
sum += temp[ypos??][xpos??]; // 下
sum += temp[ypos??][xpos??]; // 右下
sum += temp[ypos??][xpos??]; // 右
sum += temp[ypos??][xpos??]; // 右上
sum += temp[ypos??][xpos??]; // 上
```

??部分の±は、自分で考える

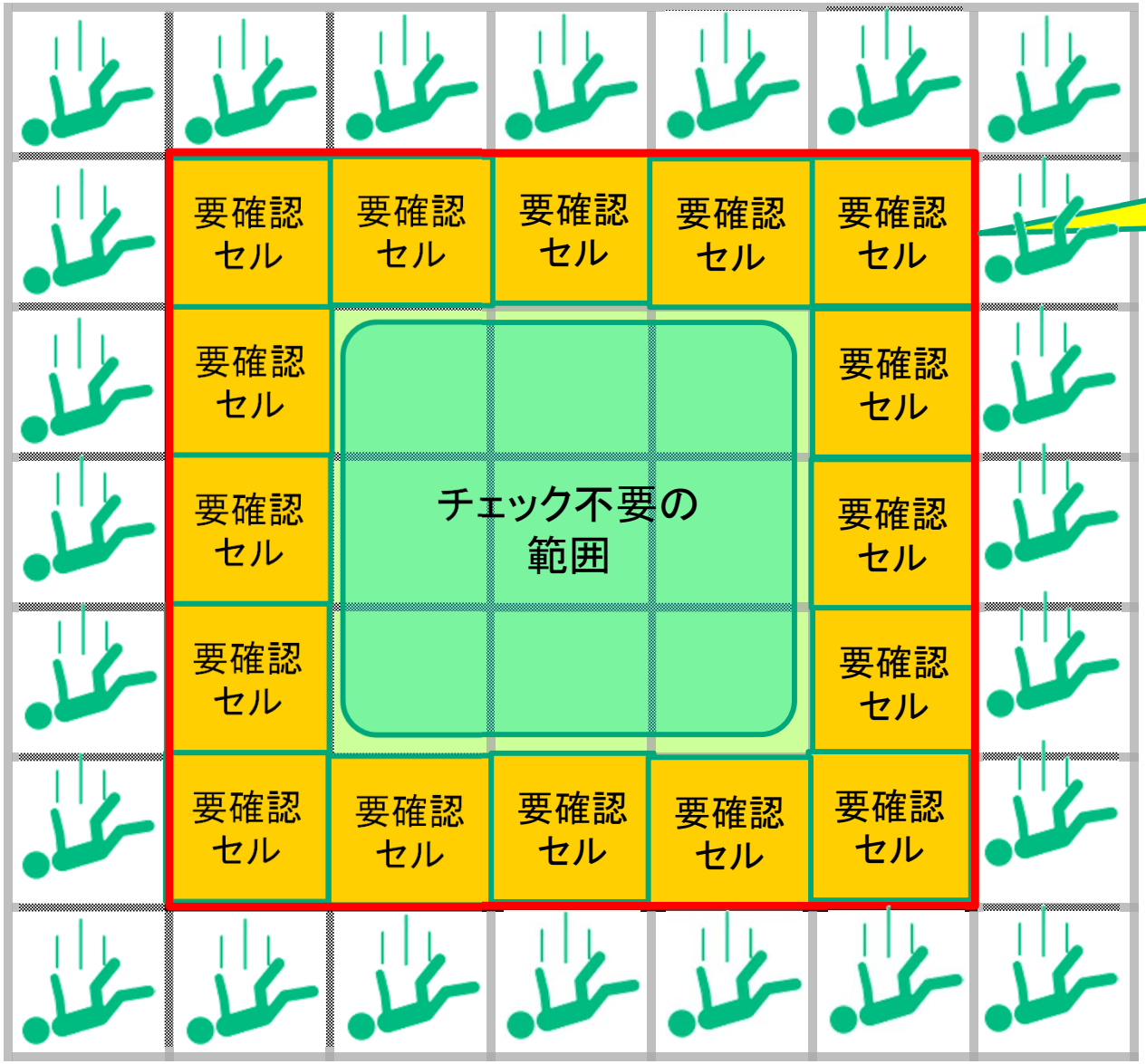


??の座標位置は、自分で考える

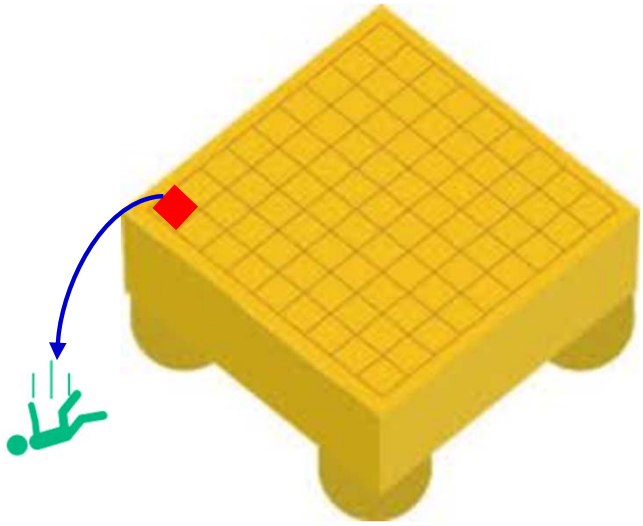
- 次に、座標の指定が配列の大きさを超えてアクセスはできないので、
  - マイナスする座標は、自分の位置が0より大きくなければならない
    - 自分の位置が0の場合、-1をすると、座標はマイナスの値になるので、
    - if (座標 > 0)
  - プラスする座標は、自分の位置が最列の最大値-1 よりも小さくなければならない
    - 自分の位置が最大値の場合、+1をすると座標をオーバするので、
    - if (座標 < WORLD_X -1)

範囲外をアクセスすると落っこちる

5x5の25マスの場合



最大テーブルサイズの枠
WORLD_H-1, WORLD_W-1



配列の一番外側はチェックが必要。
配列外をアクセスすると落っこちる

実際の参考プログラム例

```
int lifeCount(char temp[WORLD_H][WORLD_W], int ypos, int xpos){

   int sum=0;

   if(ypos>0 && xpos>0)
      sum += temp[ypos-1][xpos-1]; // 左上
   if(xpos>0)
      sum += temp[ypos  ][xpos-1]; // 左
   if(ypos<WORLD_H-1 && xpos>0)
      sum += temp[ypos+1][xpos-1]; // 左下

      sum += temp[ypos??][xpos??]; // 下
      sum += temp[ypos??][xpos??]; // 右下
      sum += temp[ypos??][xpos??]; // 右
      sum += temp[ypos??][xpos??]; // 右上
      sum += temp[ypos??][xpos??]; // 上


   return sum;
}
```

マイナスする座標は、0より大きい必要がある

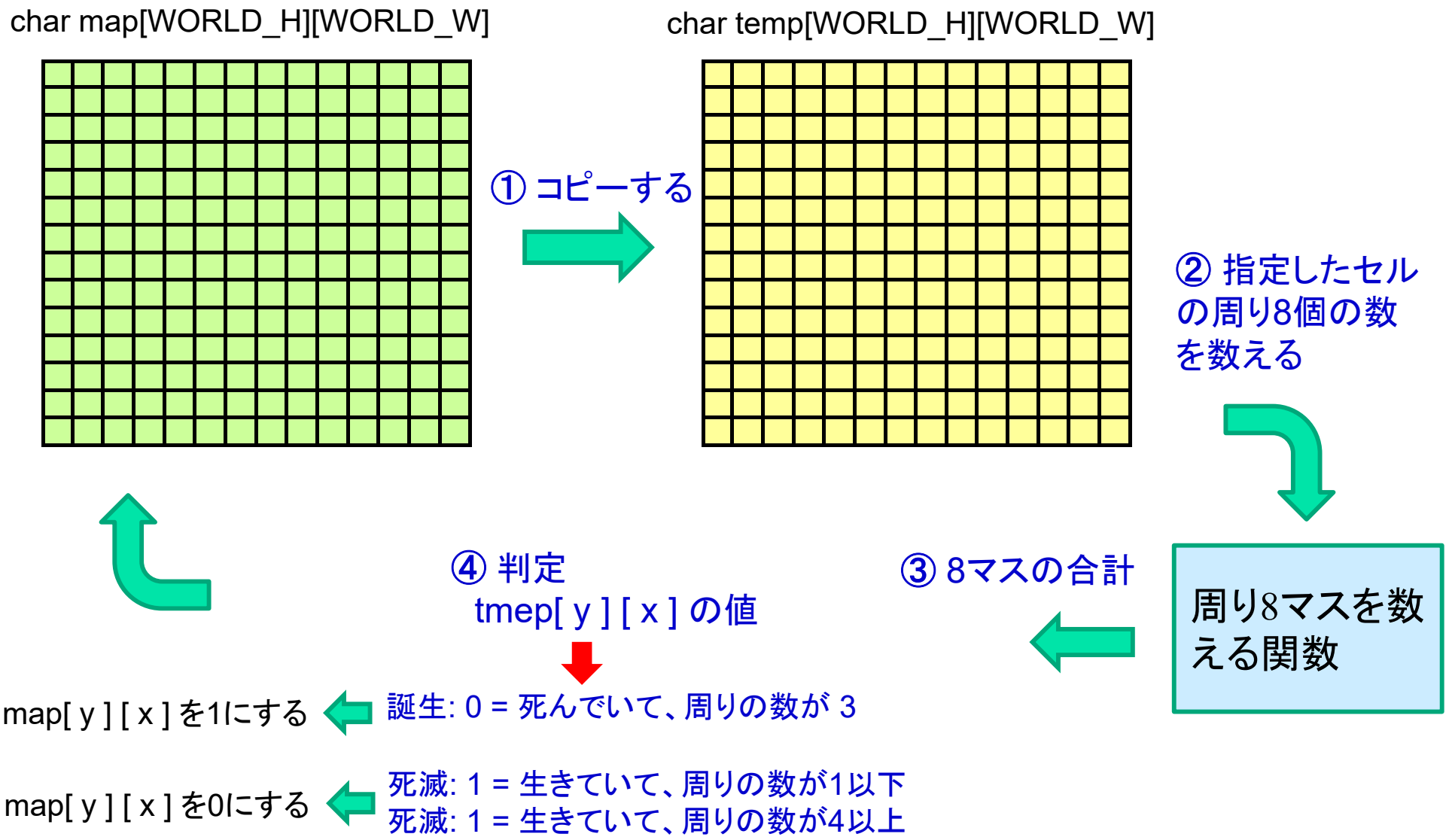
プラスする座標は、最大値-1より小さい必要がある

残りの5カ所は、自分で考えてif文を追加する

周り8個の生存数

世代を進める関数の作成のヒント

コピーしたtempの内容を調べて、結果をmapに反映する



画面の制御 - エスケープシーケンスとは?

- Unixなどの世界では、画面の制御にエスケープシーケンスというものを使用している。
- Windows10から、コマンドプロンプトでエスケープシーケンスがサポートされた。
- エスケープシーケンスとは、0x1bに続く文字で色々と定められている。
- 下記は、その一例です。

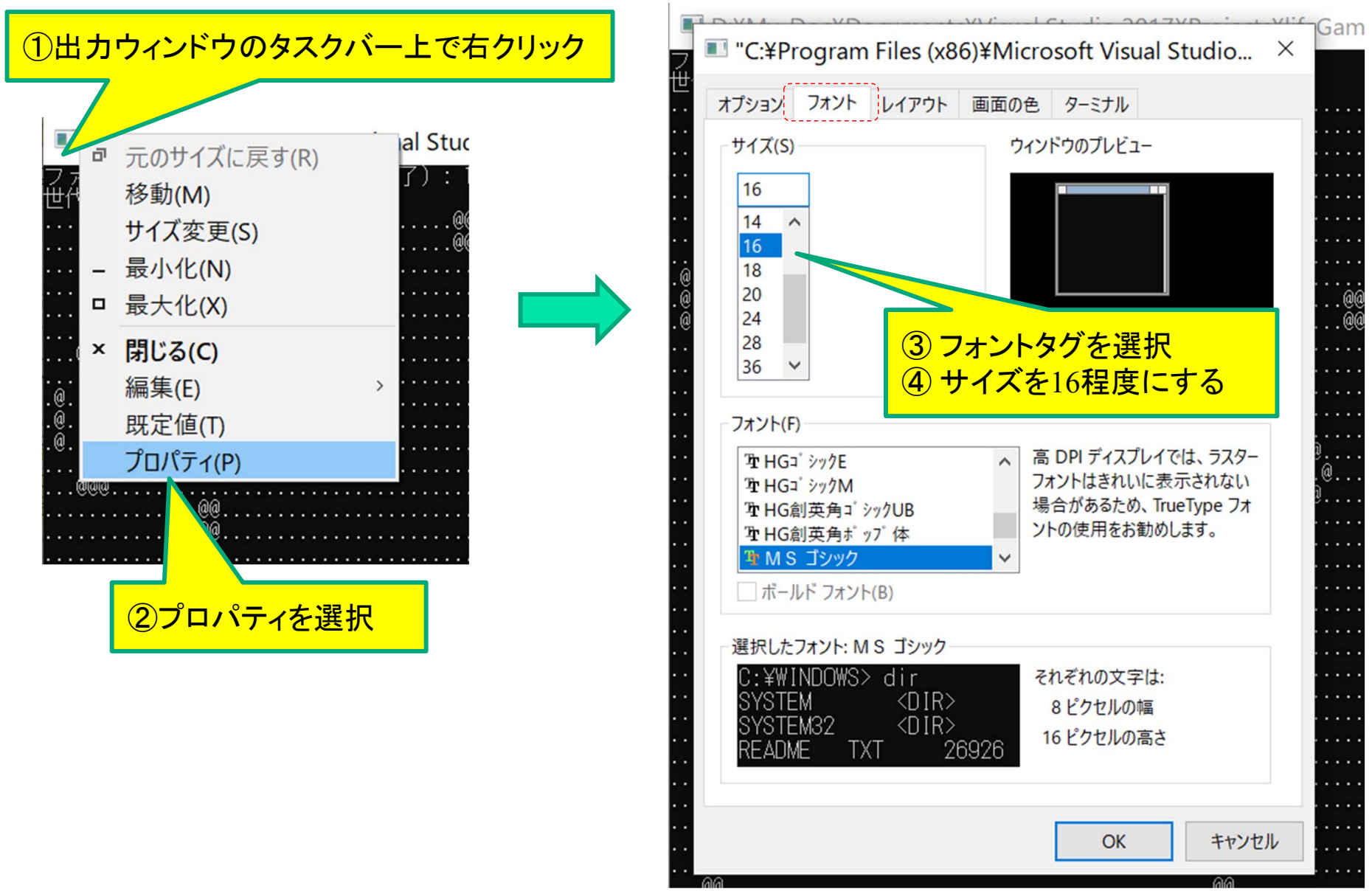
```
printf("¥033[2J");       //画面クリア
printf("¥033[0K");       //カーソル位置からその行の右端までをクリア
printf("¥033[1K"");      //カーソル位置からその行の左端までをクリア
printf("¥033[2K"");      //カーソル位置の行をクリア

printf("¥033[%d;%dH",10,20); //カーソル位置を、高さ10行目、横20文字目に移動
printf("¥033[%dC" ,10);  //カーソルを10文字だけ右に移動
printf("¥033[%dD" ,10);  //カーソルを10行文字だけ左に移動
printf("¥033[%dB" ,10);  //カーソルを10行だけ下に移動
printf("¥033[%dA" ,10);  //カーソルを10行だけ上に移動

printf("¥033[30m");      //黒の文字に変更
printf("¥033[31m");      //赤の文字に変更
printf("¥033[32m");      //緑の文字に変更
printf("¥033[33m");      //黄色の文字に変更
printf("¥033[34m");      //青の文字に変更
printf("¥033[35m");      //マゼンダの文字に変更
printf("¥033[36m");      //シアンの文字に変更
printf("¥033[37m");      //白の文字に変更
printf("¥033[39m");      //文字の色を通常の色に戻す
Printf("¥033[0m");       //リセット(設定を未設定の状態に戻す)
```

Teratermなどでテストをする場合は、
ESCキーを押した後に、「[」「2」「J」と押します。
以下同じです。

¥033　は、C言語における8進数の定数表現
¥x1b　として、16進定数としても良い。

例: "¥033[30m" は : "¥x1b[30m"　と同じ

出力ウィンドウのフォントサイズを変更



step 3：初期データをファイルから読込む

step3では、初期値の内容を
テキストファイルから読んで設定する。

ファイルの操作

- ファイルの操作は下記の4つの操作が必要になります
  - (1) FILE型でファイルポインタを定義する(オープンした時の情報が入る)
  - (2) ファイルを開く(オープン) ➔ 成功するとファイルポインタに情報が入る
  - (3) ファイルポインタに対して、ファイルの操作(読み込み/書き込みなど)を行う
  - (4) ファイルを閉じる(クローズ) ➔ ファイルポインタを閉じる

- ファイルポインタとはオープンされたファイルの情報が入っている
  - FILE型のポインタとして定義 ➔ 例: FILE *fp;
  - FILE型には下記の情報が含まれているが、現在のVisual Studioでは見えない。

```
struct _iobuf {
        char *_ptr;
        int   _cnt;
        char *_base;
        int   _flag;
        int   _file;   ← システムのファイル番号(0,1,2は標準入出力のため3から始まる)
        int   _charbuf;
        int   _bufsiz;
        char *_tmpfname;
};
typedef struct _iobuf FILE;
```

ファイルのオープン

- ファイルのオープンは、fopen( ) あるいは fopen_s( ) を使う
  - fopen()の使用例

```
FILE *fp;     // ファイルポインタの定義

fp = fopen( fineName , mode);  // ファイルのオープン
```

オープンに成功すると、fpにファイル情報のポインタが入る
オープンに失敗すると、fpにNULLが返される

  - C言語では、if文の中に入れるとオープンの成功/失敗が判断できる

```
if ( ( fp = fopen("sample_data1.txt",  "r") ) != NULL)
            // オープン成功
else
            // オープン失敗
```

fopenのmode指定

- ファイル名がパスリスト(ドライブ名やフォルダ名がある)場合
  - 区切り記号は、”¥¥” にする。

```
char *filename = “C:¥¥Users¥¥user¥¥source¥¥repos¥¥test¥¥test¥¥source.c”;
fp = fopen ( filename, “r”);
```

- fopenのモードとは、ファイルをどのように開くかを文字列で指定
  - 基本は、次の3のモード

モード	機能　と　開始位置	ファイルがないとき
″r″	読み取り(ファイルの先頭から)	エラー
″w″	書き込み(ファイルの先頭から)	新規作成
″a″	追加書き込み(ファイルの最後から)	新規作成

  - 基本モードに対して、次のファイル形式と動作を指定

モード	ファイルと種類と動作
″b″	バイナリ形式(″b″がないとテキスト形式になる)
″+″	更新モードの指定

  - 実際の指定は、上記の組合わせで指定する
    - “rb”　バイナリデータの読み込み
    - “w+”　テキストデータの追加書き込み

ファイルのRead/Write操作

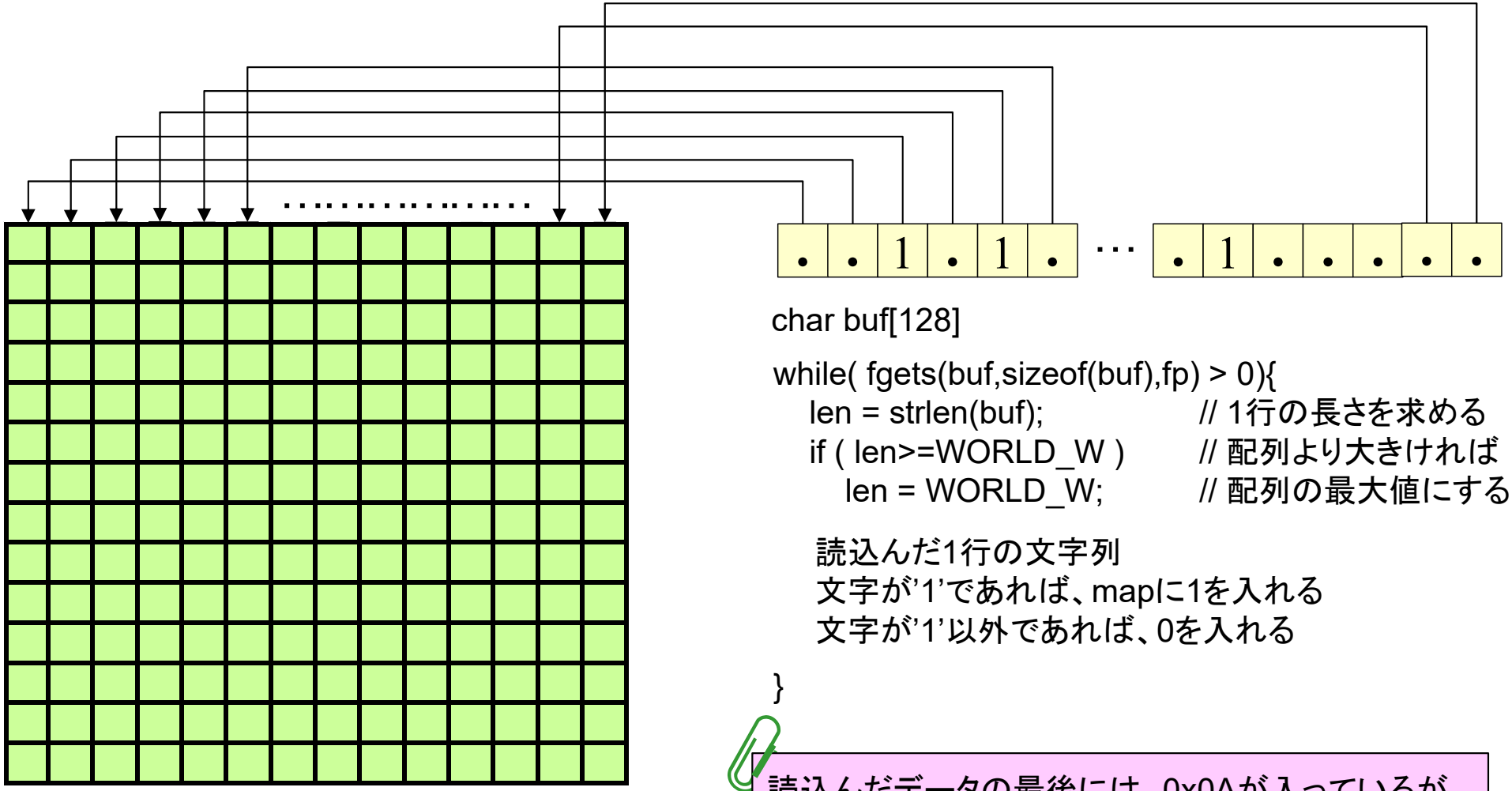
- 代表的なファイルの読み込み関数
  - fgetc()　　１文字の読み込み
  - fgets()　　１行の読み込み(¥n)まで読む(0x0Aも読込まれている)
  - fread()　　バイナリの読み込み
  - fscanf()　　書式指定の読み込み

- 代表的なファイルの書き込み関数
  - fputc()　　1文字の書き込み
  - fputs()　　1行の書き込み
  - fwrite()　　バイナリの書き込み
  - fprintf()　　書式指定で書き込み

Windowsのテキストファイルは、改行コードがCR(0x0D)、LF(0x0A)の2文字になっています。
fgets() で読込んだ場合、¥nのLF(0x0A)までを取り込まれ、CR(0x0D)は無視して読み飛ばされます。
一方、scanfの%sで読込んだ場合は、CR(0x0D)の前までを取り込み、CR(0x0D)は読込まれません。
そして、OSの内部にLF(0x0A)が残っているので注意が必要です。

初期データをファイルから読込み初期化する

ファイルから初期データを読込み、mapを初期化する




```
char buf[128]

while( fgets(buf,sizeof(buf),fp) > 0){
    len = strlen(buf);              // 1行の長さを求める
    if ( len>=WORLD_W )             // 配列より大きければ
        len = WORLD_W;              // 配列の最大値にする

    読込んだ1行の文字列
    文字が'1'であれば、mapに1を入れる
    文字が'1'以外であれば、0を入れる

}
```

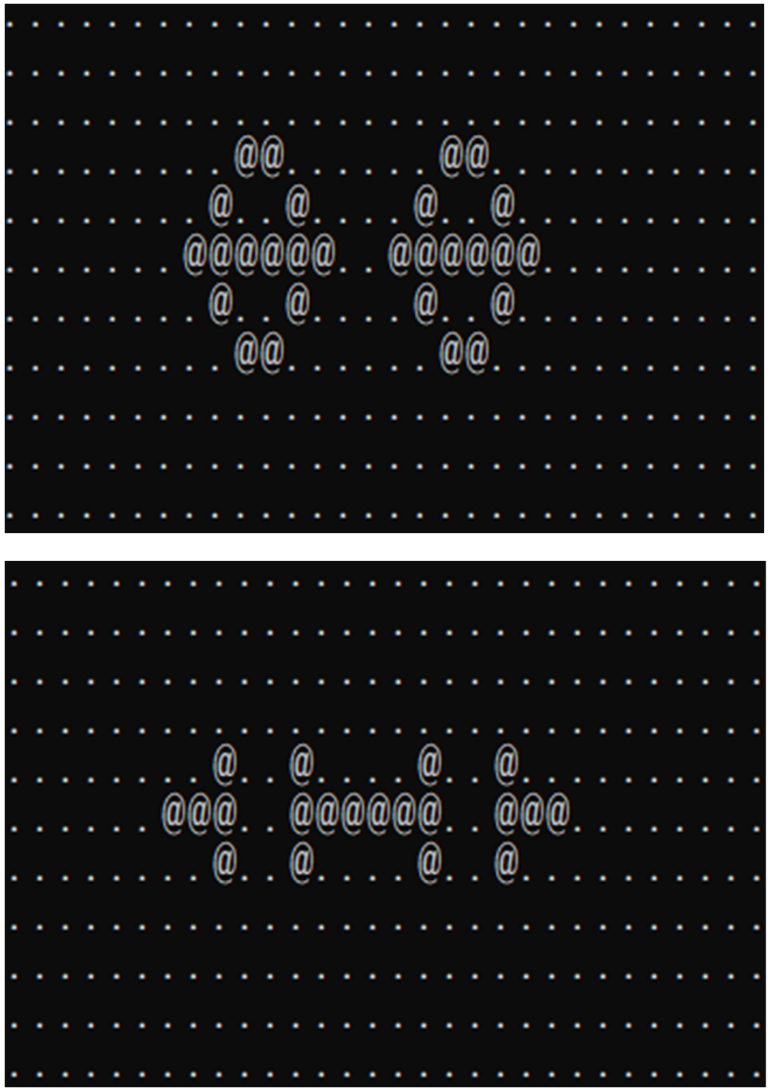
読込んだデータの最後には、0x0Aが入っているが、'1'ではないので、0として処理をする

初期値のパターン(sample_data1.txt)

初期値のデータパターン

```
........................................
........................................
........................................
........................................
........................................
........................................
................11111111................
................1.1111.1................
................11111111................
........................................
........................................
........................................
........................................
........................................
........................................
```

実行結果



ファイルのパターンデータはマップの配列より小さいため、
初期化の前に、配列の内容を0クリアしておく必要がある。

上記の2つのパターンが繰り返される

初期値のパターン(sample_data2.txt)

初期値のデータパターン

```
.....................................
.........................1...........
.......................1.1...........
.............11.....11............11.
............1...1...11............11.
.11........1.....1...11..............
.11........1...1.11...1.1............
...........1.....1.....1.............
............1...1....................
.............11......................
.....................................
.....................................
```

このパターンは、Glider Gunといって、
永久にグライダー?を放出する。

実行結果

Step4:2つの初期化を切り替える

Setp4: 乱数モードとファイルモードを選択

- scanfで、文字列を入力
  - 最初の文字が0～9('0'～'9')なら乱数モードで動作
    - 10進文字列を、atoi( )関数を使って数値に変換し、乱数の種を設定
  - 最初の1文字が、数字以外であれば、ファイルモードで動作
    - 入力された文字列をファイル名としてファイルをオープンする

```
char inbuf[128];          // scanfで読込むバッファの定義

printf("ファイル名か乱数の種の入力(0で終了): ");
scanf("%s", inbuf);

if(inbufの最初の文字のが0～9であれば)  ← 文字の比較になります。
   result = 乱数による初期化(map,inbuf):
else
   result = ファイルから読込んで初期化(map,inbuf);

if ( result != 0 )
   初期化失敗
```

参考資料:Windowsによる画面制御

参考資料: Windowsによる画面制御例

```
#include <Windows.h>

    system("cls");        // 画面のクリア
```

方法1

```
COORD coord;
HANDLE hStdout;
hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
```

画面制御の初期化
(mainの最初に実行)

```
 coord.X=0;   // Xの位置(横)
 coord.Y=1;   // Yの位置(縦)
 SetConsoleCursorPosition(hStdout,coord); // カーソルの移動
```

カーソルの移動
(mapの表示前に実行)

方法2

```
COORD coord;        // 構造体の宣言

coord.X=0;   // Xの位置(横)
coord.Y=1;   // Yの位置(縦)
SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
```