

ソフト系 C言語実習課題 1

ライフゲームの作成

V3.07

2021 JTEC m.h

1

C言語研修課題の2題目は、ライフゲームの作成です。

タイトルは、ゲームとなっていますが、正確には簡易のシミュレーションです。

ここでは、2次元配列の操作、戻り値のある関数の作成、テキストファイルの読み込みを学びます。

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

ファイルの読み込みを学ぶ

– 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

2021 JTEC m.h

2

本課題は、次のようなStepで進めていきます。

Step1: ライフゲームのルールを理解する

プログラムを作成するには、課題の内容を理解し、どのように構成でどのような関数が必要かなどを整理することが重要です。実際の仕事でも、要件定義として仕様の作成します。

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だします。

初期化ができていれば、乱数で初期化されたmapの内容が表示されます。

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: 初期化を乱数でなく、ファイルから初期パターンを読み込み初期化します。

ファイルを開いてテキストデータを読み込みます。

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するようにします。

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。

まずは、「ライフゲーム」という言葉を初めて聞く人は、ライフゲームがどんなものかを知る必要があります。
「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動画があります。

これは、上からグライダーが永遠に発射されるというものです。

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。



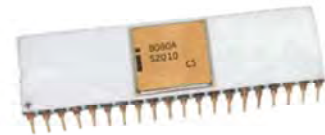
4004の概観

- 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

2021 JTEC m.h

4

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す。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ます。

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ありませんでした。

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」が発売されましたが、4004は4ビットのCPUで、それほど利用されることはありませんでした。さらに、翌年の1972年に、世界初の8ビットCPUである「8008」を発売しましたが、このCPUもアドレス空間が14ビット(16KB)しかなく、4004同様にそれほど利用されませんでした。

そして、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」が発売されました。この8080が世界のマイクロコンピュータの火付け役となりました。8080は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りました。

マイクロソフトの創業者である、ビル・ゲーツも、この8080で動作するBASICを開発してマイクロソフト社を設立しました。

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」でした。下記は、スタートレックゲームの画面です。ゲームといっても、モノクロの文字だけで表現するものでした。



「スタートレック」は、大宇宙空間を舞台にした本当のゲームでした。プログラムは、BASICで作成されて、当時として人気ゲームでした。

引用URL: <https://shamirgame.blog.fc2.com/blog-entry-165.html>

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(基盤の目を想定)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: 過疎過ぎて繁殖できず死滅する**

- 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: 過密過ぎて繁殖できず死滅する**

- 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>

2021 JTEC m.h

5

ライフゲームは、ゲームという名前が付いていますが、実際は、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、その進化の過程で表示される模様を楽しむものです。

そのルールはとてもシンプルで、ある領域(基盤の目を想定してください)の中に生物が存在し、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というものです。

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。唯一生命誕生の条件になります。

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。これは、現状維持ということで、何もしません。

過疎: 過疎過ぎて繁殖できず死滅する

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: 過密過ぎて繁殖できず死滅する

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。

生命の誕生、進化、絶滅のプロセスは、自然界においても成り立ちます。

肉食動物は草食動物を捕食しますが、食べ過ぎて草食動物が減ると肉食動物も減ります。

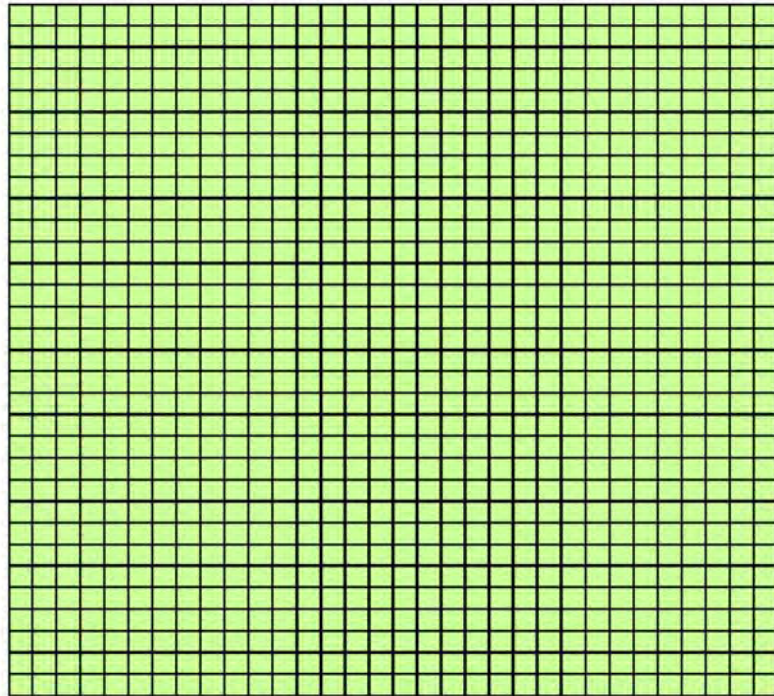
一方、草食動物は草を食べ、食べ過ぎて草がなくなると草食動物も減ります。

肉食動物 ⇔ 草食動物 ⇔ 草木 ⇔ 昆虫

ライフゲームは、こうした自然界にある連鎖をシミュレーションするものです。

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



2021 JTEC m.h

6

この図は、ライフゲームを行う2次元配列のテーブルで、**map**(地図)と呼びます。そして、1つ1つのマス目をセルと言います。

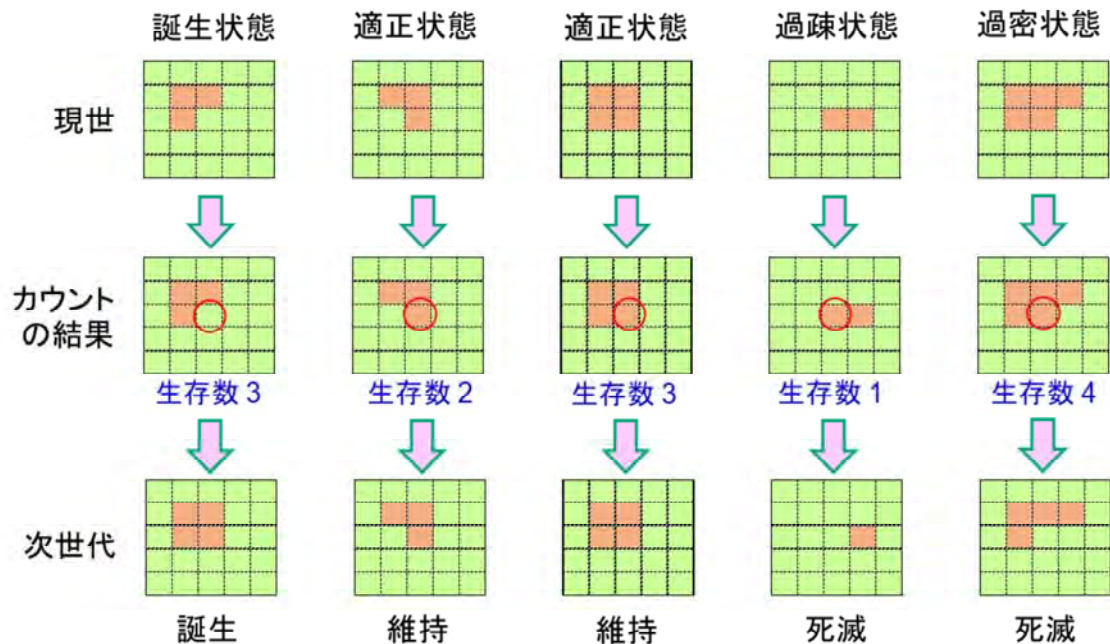
mapを大きくしたほうが進化の過程の模様を楽しめます。**map**を大きくすることは、配列を大きくすることです。ただ、今回は、黒い画面のコマンドプロンプトに文字として表示するので、縦方向はあまり大きくはできません。

この図では、縦38行、横 38文字の大きさを説明していますが、実際のプログラムでは、横方向を2倍の76文字にしてやってみてください。

横は、縦の2倍が、黒いコンソール画面では正方形に近い形になります。

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、



2021 JTEC m.h

7

この図は、ライフゲームのルールに従った、実際の変化の様子を表したものです。

map全体だと大きすぎるので、ここでは、5×5の25マスの中心位置のセルを調べたとき状況で説明します。

「現世」

現在の状態です。このマップの真ん中を中心にして、隣接する周りの8個の生存数を調べるというものです。

「カウントの結果」

中心に隣接する周り8セルを数えています。mapの下に表示している数字が生存数の数です。

「次世代」

周り8セルを数えた結果、ルールに従って次世代に反映させます。

この図で、一番左側の「誕生状態」は、自分が死んでいて、周りの生存数が3です。この状態が、唯一新しい生命が誕生する条件です。次世代で、自分の位置に生命が誕生します。

「適正状態」の例は、自分が生存していて、周り8セルの生存数が2と3です。この状態は、現状維持で、次世代には反映されません。

「過疎状態」は、自分が生存していて、周り8セルの生存数が1でした。この状態は、過疎状態で、次世代で死滅してしまいます。

「過密状態」は、自分が生存していて、周り8セルの生存数が4でした。この状態は、過密状態で、次世代で死滅してしまいます。

Step2: 4つの必要な関数を作成する

どのようなことをするかが、ぼんやりと分かったと思います。

ここまでは、生命の誕生と死滅は理解できたが、実際プログラムでどのようにするかまでは、見えてきていないと思います。

それで、**Step2**では、ライフゲームを実現するために、提示された要求仕様(課題と条件)から、全体をどのような構成にして、どのような関数が必要かを整理します。

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。

2021 JTEC m.h

9

ここで、もう一度、課題を確認します。

課題: 下記の条件でライフゲームを作成する

条件: 下記のとおり

- 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。

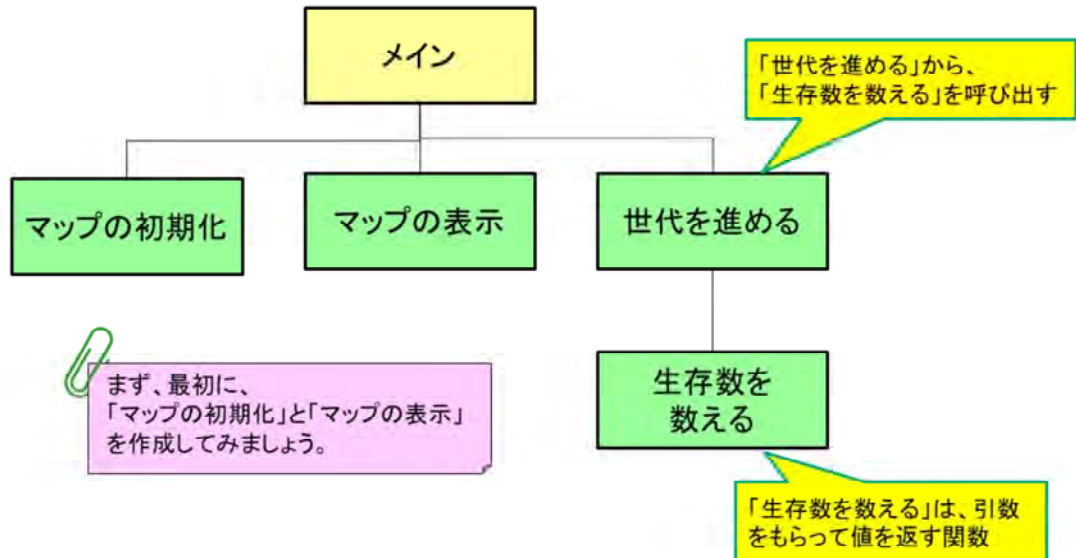
ちなみに、23/3ルールとは:

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



2021 JTEC m.h

10

要件定義(課題と条件)から、全体のモジュール構成を考えます。

今回は、図のようなモジュール(ここでは関数)の構成になります。

「メインは」、`main()`関数です。ここから、各関数を呼び出すことになります。

「マップの初期化」は、`main`から呼ばれて、`map`を初期化します。

最初のStep2では、乱数を使って初期化をします。

Step3では、ファイルからパターンデータを読み込み初期化をします。

「マップの表示」も、`main`から呼ばれて、2次元配列の`map`の内容を表示します。

今回は、生存している場合は、`'@'`を、死んでいる場合は、`'.'`を表示するようにします。

まず、Step2-1として、「マップの初期化」と「マップの表示」を作成してみます。

初期化された`map`が画面に表示されます。

次に、Step2-2として、「世代を進める」と「生存数を調べる」を作成します。

「生存数を調べる」は、「世代を進める」関数から呼ばれます。

作成する関数の中で「生存数を調べる」は、生存数を数えて値を戻り値として返す関数になります。

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 );
    }

    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
    }
}
```

もし、printf("¥033[2J")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。

2021 JTEC m.h

11

全体のモジュール構成と、作成する関数がわかったところで、もう少し具体的な例で示します。

```
#define WORLD_H 38
#define WORLD_W 76
int main(void){
    char map[WORLD_H][WORLD_W];
    printf("¥033[2J");          // 画面を消す ← 正しく実行できない場合は最終ページを参照してください。
    初期化(map);
    ループ{
        printf("¥033[2;1H");    //カーソル位置を、高さ2行目、横1文字目に移動 ← 正しく動作しない場合は同上
        マップを表示(map);
        世代を進める(map );
    }
    return 0;
}

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
    このとき、1と0の比率は、7:3程度がよいので、乱数の結果を10で余りを求め、if文で比率を変える
    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する。
    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。。

2021 JTEC m.h

12

mapの内容は、「0」か「1」の2値が入ります。そのため、データ型はchar型で十分です。

mapは、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)。

画面制御は、下記のエスケープシーケンスを使用する。

```
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
```

表示開始位置を2行目からにするのは、最上段に表示されている入力した文字を消さないためです。

乱数の場合、入力した数字で乱数の種を初期化しています。

そのため、同じ数字を入力すると、再度同じパターンで実行されます。

もし、上記のprintf()で画面制御が正しく動作しない場合、

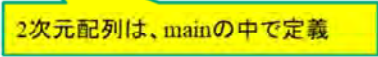
本テキストの最後にある「Windowsによる画面制御」を参考にしてください。

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

// 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
        */
    }
}
```



2021 JTEC m.h

13

それでは、初期化関数から順番に、ヒントを出していきます。

最初の初期化関数は、乱数を使って2次元配列mapの中を「1」か「0」にしていくものです。

for文の2重ループで構成される、典型的な2次元配列のアクセスになります。

乱数を使って、配列の中に1か0を入れていくわけですが、`rand() % 2` というように、2で割った余りにすると、1と0の比率が1:1になってしまいます。

そこで、10で割った余り(0~9)を求め、if文で判定することで、1と0の比率を変えることができます。

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
*/
    }
}
```

2021 JTEC m.h

14

mapの表示関数も典型的な、for文の2重構造になります。

初期化関数同様、外側のfor文が縦のループ、内側のfor文が横のループとなります。

そして、表示するmapの値が、「1」なら"@"を、「0」(else)なら"."を出力します。

九九の表でも経験しましたが、内側のforループが終わったら改行をします。

2次元のデータを表示するとき、内側のループが終わったら改行するのは、定型パターンです。

初期化関数と、表示関数ができたら、Step2-1のコーディングが終了となります。この段階でコンパイルをして実行してみましょう。

画面に、"@と"."の混じったmapの内容が表示されればOKです。

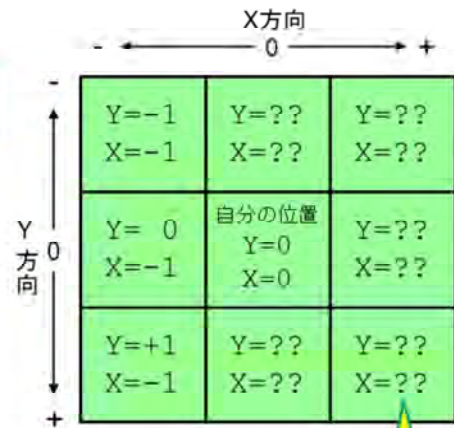
自分の周り8個の生存数を数える考え方

■ 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



■ 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ーするので、
 - if (座標 < WORLD_X -1)

??の座標位置は、自分で考える

2021 JTEC m.h

15

Step2-1で、初期化関数と表示関数ができたら、いよいよ、世代を進める関数を作成します。世代を進める関数は、指定したセルの周り8個の生存数(セル)を数える関数を呼び出します。

まずは、生存数を数える関数から作成します。生存数は指定された座標位置を中心にして、周り8個のセルを数えます。

初期化関数で、mapの中には「1」と「0」しか入っていません。「1」は生存、「0」は死んでいることを表しているので、単純に回り8個のセルの内容を足していけば良いことになります。

そのために、周り8個の座標を求めます。ノートに右側の緑の図のように、9マスの枠を書いてください。

そして、各マスに座標を書いていきます。例として左側の1列の座標を書いていきます、同様に残りの6セルの座標を書いてください。座標を書き終わったら、左側の例のように、実際にコーディングをします。

8セルを数えるので、8行の加算するプログラムになります。

ライフゲームで、一番バグが出るのが、この生存数を調べるところです。慎重にコーディングをしてください。また、どここのセルを数えているのかのコメントと書いておきます。

ここまでで、生存数を数える関数の半分が終了です。

この生存数を数える関数は、引数で与えられた座標位置を中心として周り8セルを数えています。そのとき引数で与えられた座標の変数に対して、「-1」とか「+1」をしています。このとき、指定された座標が「0」なのに -1 をしたら、座標位置がマイナスとなり、配列の領域以外をアクセスすることになります。

一方、+1 するところは、指定された座標が「最大値-1」だった場合、+1をした場合、同様に配列の領域を超えてアクセスすることになります。C言語の配列は、0から始まるので、最大値は定義した大きさ-1になります。そのため、定義時に使用した値にするとの領域を超えてしまいます。

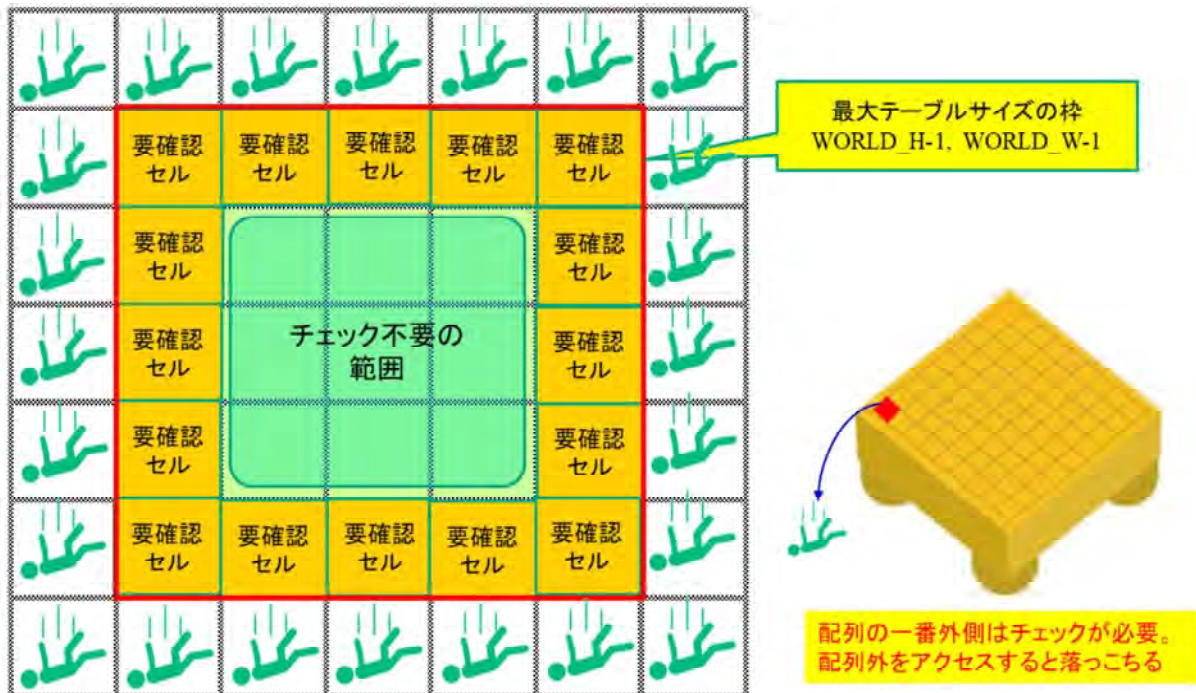
ということで、「-1 する座標は、引数で与えられた座標位置が0より大きくなければなりません」。

一方、「+1する座標は、+1をしても最大値-1 よりも小さくなければなりません」。

次ページは、その様子を説明したものです。

範囲外をアクセスすると落ちこちる

5x5の25マスの場合



2021 JTEC m.h

16

この図で、赤い枠で囲まれた範囲が、実際の2次元配列です。一番外側の以外は、座標を-1しても、+1しても、配列を超えることはありません、

しかし、一番外側(図の黄色の部分)は、-1をしたり+1をすると配列の領域を超えてしまいます。

配列の領域を超えるということは、碁盤が落ちることを意味します。

配列の領域を超えてのアクセスは、メモリ破壊にもつながり、組み込み系のシステムでは、その時点でシステムがクラッシュしてしまうことがあります。

そのため、「座標を-1するところは0より大きい」、「座標を+1するところは、最大値-1より小さい」という条件を加える必要があります。

-1や+1をしない座標については、チェックは必要ありません。

その条件を入れた例が、次ページです。

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+1]; // 右下  
    sum += temp[ypos+1][xpos+1]; // 右  
    sum += temp[ypos+1][xpos]; // 右上  
    sum += temp[ypos+1][xpos-1]; // 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

2021 JTEC m.h

17

これが、生存数を調べる関数の完成形です。

先にコーディングをした、周り8セルを数える行の前に、if文を1行ずつ入ります。

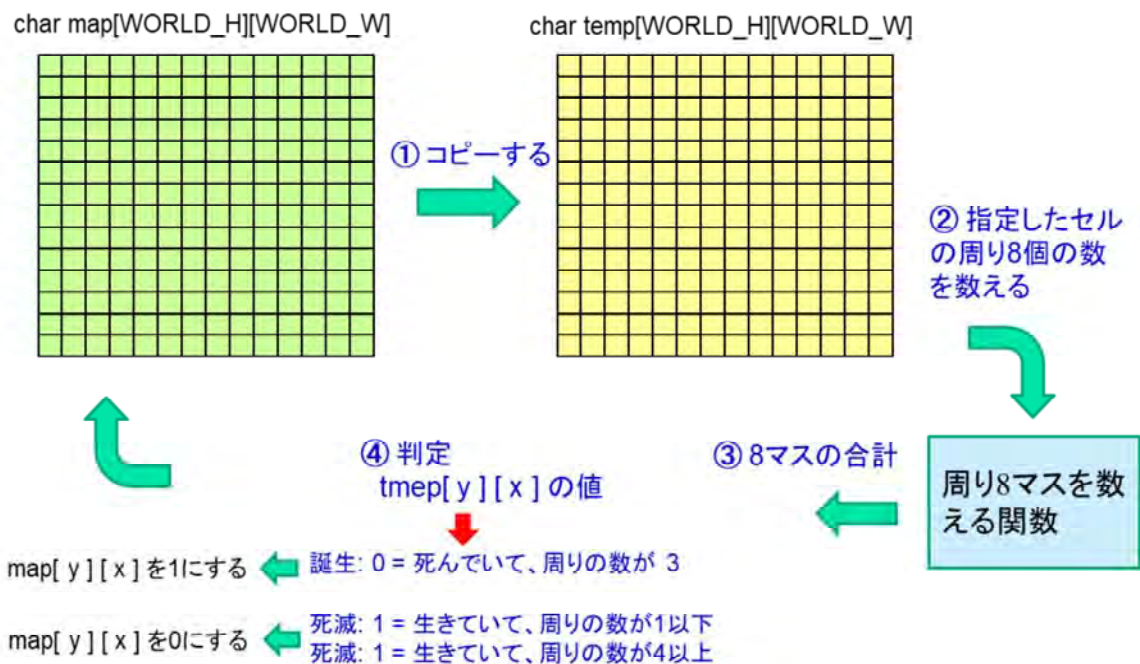
ご覧のように、「-1 する座標は、0より大きい」、「+1する座標は最大値-1より小さい」という条件が入っています。

左側の列は、例を書いています。残りの6セルについても同様にします。

何度も言いますが、ライフゲームが正しく動作しない場合、ほとんどこの生存数を数えるところが間違っています。慎重にコーディングをしてください。

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



2021 JTEC m.h

18

生存数を調べる関数ができたら、いよいよ、世代を進める関数になります。

世代を進める関数では、ローカル変数として、縦、横の座標の変数の他に、mapとまったく同じ大きさの2次元配列を定義します。ここでは、tempという名前を説明をします。

そして、引数で受け取った2次元配列(map)を、ローカル変数で定義した配列(temp)に、そのままコピーします。

C言語では、配列変数同士の代入はできないため、初期化関数や表示関数と同じように、for文の2重ループでコピーします。

コピーが終わったら、いよいよ世代を進めます。世代を進めるということは、コピーされたtempの配列を調べて、結果をmapに反映していくことです。

mapをtempにコピーするときに使用した、for文の2重ループをそのままコピーして利用できます。

そして、tempと縦、横の座標位置を引数にして、生存数を調べる関数を呼び出すと、生存数が戻り値として返されます。

生存数が返されたら、現在のtempの座標位置の値を調べ、下記のルールの通りに処理をします。

(1)「現在の座標位置は死んでいて、周りの生存数が3の場合」、生命誕生として、mapの同じ座標位置「1」を入れます。

(2)「現在の座標位置は生きていて、周りの生存数が、1以下、あるいは4以上だった場合」、死滅としてmapの同じ座標位置に「0」を入れます。

これ以外の条件は、現状維持ですので、何もしません。

これで、ライフゲームに必要な関数はすべて完成しました。

あとは、main関数で、for文などでループをして、表示関数と世代を進める関数を呼ぶようにします。ループ回数は500回か1000程度が良いと思います。

ただ、ここで問題があります。単に表示関数呼ぶだけで、マップの表示がどんどんスクロールされてしまいます。そこで、マップを表示する前に、画面のカーソル位置を先頭に戻してやる必要があります。

そうした、画面制御には、ANISIで定められた「エスケープシーケンス」という国際規格があります。これは、エスケープ文字「0x1B」に複数の文字を避けて指定するというものです。

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[OK"); //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進数としても良い。
例: "\033[30m" は: "\x1b[30m" と同じ

2021 JTEC m.h

19

このリストは、エスケープシーケンスの一部です。実際のエスケープシーケンスについては、別途ネットなどで調べてください。

エスケープシーケンスは、全体を文字列として指定します。最初の文字は、エスケープ文字「0x1B」で、そのあとに「[」の文字が続き、以後複数の文字で構成されます。ただし、文字列にエスケープ文字を直接書くことはできません。C言語のエスケープ文字「¥」記号を使って入れます。

「エスケープ文字」と「エスケープシーケンス」言う言葉が出てきましたが、両者を混同しないでください。文字列の中に、「0x1B」という文字コードを入れるには、16進数で入れる場合と8進数で入れる場合の二通りがあります。

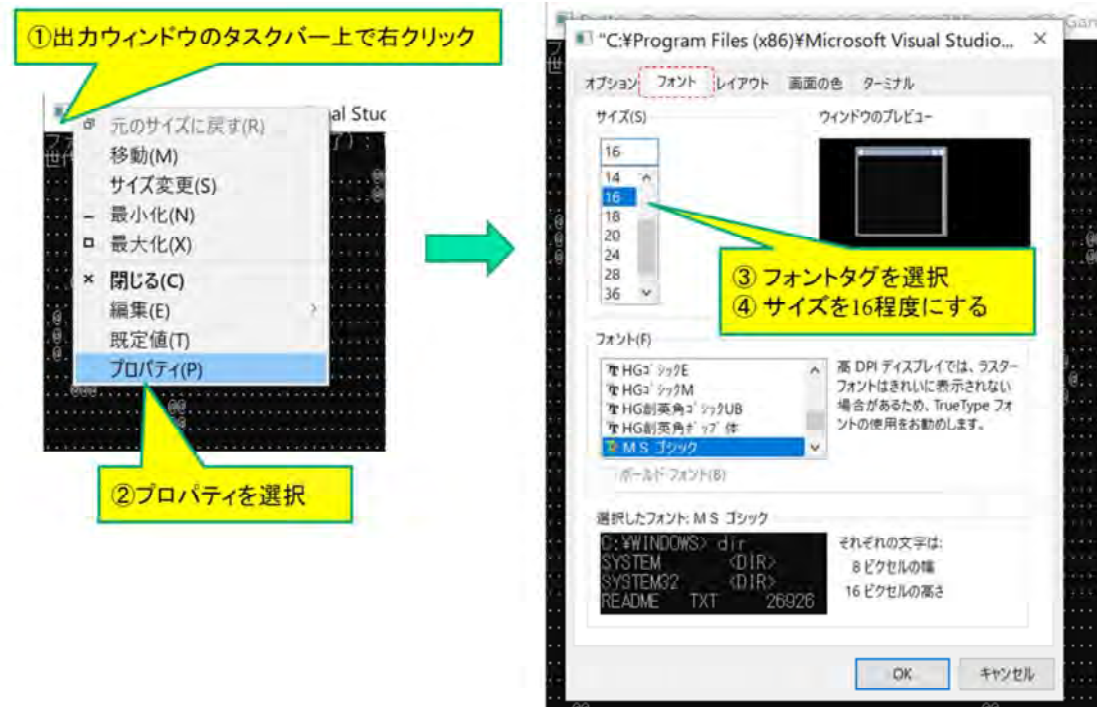
16進数で指定する場合は、「¥x1B」と書きます。← ¥記号の後は「x」1文字だけです、C言語の16進数定数指定「0x」にはしないでください。0は不要です。

一方、8進数で指定する場合は、¥033と書きます。C言語では、0から始まる数字は、8進数として評価します。ネット上で、¥033 としている例が多いです。033 は、16進数の0x1B、10進数の27になります。

実は、国際規格であるエスケープシーケンスは、長い間Windowsではサポートされていませんでした。しかし、Windows10から標準サポートとなり使用できるようになりました。

もし、エスケープシーケンスを出力して、画面に、「[2;1H」のような文字が表示される場合は、エスケープシーケンスがサポートされていないか、あるいはコマンドプロンプトの設定がレガシーモードになっている可能性があります。

出力ウィンドウのフォントサイズを変更



2021 JTEC m.h

20

作成したライフゲームを実行したとき、出力ウィンドウ(黒い画面)のフォントサイズが大きかったり、ウィンドウサイズが小さいとマップの表示がスクロールしてしまいます。

もし、画面がスクロールするようであれば、下記の手順でフォントサイズを小さくしてみてください。

- (1) ライフゲームの先頭にブレークポイントを設定して実行を止める。
- (2) 出力ウィンドウのタスクバー上で、マウスの右クリックをして、[プロパティ]を選択します。
- (3) プロパティ画面が表示されるので、[フォント]タグを選択
- (4) フォントサイズを16程度に設定し、[OK]ボタンをクリックします。

これで、設定は終了です。

[f5]を押して、処理を続行するか、再度実行します。

step 3 : 初期データをファイルから読み込む

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。

Step3は、ファイルの操作です。プログラムを作成して、ファイルを操作はしばしば登場します。

ここでは、ファイルの操作についての説明です。ファイルを操作するには、

- (1) ファイルを開く
- (2) 適正な関数を使って、読み書きをする
- (3) ファイルを閉じる

という3つの操作が必要になります。

ファイルのオープンには、`fopen()`関数を使いますが、Visual Studio 2017以降は、`fopen()`を使うと、関数名の後ろに、“_s”の付いた関数を使えとエラーメッセージが表示されます。

これは、セキュリティ強化(誤ってメモリを破壊することを防ぐ)のために、マイクロソフトが独自拡張をしたものです。C言語の国際規格の C11 (ISO/IEC 9899:2011)から規格になりましたが、マイクロソフトのルールです

したがって、“_s”の付いた関数は、逆にVisual Studio以外でエラーになる場合もあります。この問題は、単なる関数名の違いでなく、引数や戻り値など、仕様が違うことです。

そこで、C言語研修で、標準の関数を使うか、“_s”付を使うかになりますが、多くのC言語の参考書、ネット情報では“_s”のない仕様で解説をしています。そのため、専門研修では、“_s”の付かない従来の関数で実習を進めます。

それで、Visual Studioで、`fopen()`を使うときは、プログラムの先頭に、下記の1行を入れてください。

`#pragma warning(disable:4996)` ← `#pragma` は、コンパイラオプションです。

本章のVisual Studioの説明の最後にある、「Visual Studio 2017で発生する4996のエラー」も参考にしてください。

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;
```

2021 JTEC m.h

22

ファイルの操作は下記の4つの手順が必要になります。

- (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に成功したとき、ファイルの情報が入っている構造体へポインタです。

FILE型のポインタで変数を定義します → 例: FILE *fp;

Visual StudioのFILE型は、構造体として下記の情報が含まれていますが、現在のVisual Studioでは直接見ることはできなくなっています。

FILE型は、他のコンパイラにもありますが、その内部構造がコンパイラによって異なるためです。

あくまで、参考用であり、実際のプログラムで中を見ることはありません。

```
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は標準入出力で使われているので3から始まる)
    int _charbuf;
    int _bufsiz;
    char *_tmpfname;
};
typedef struct _iobuf FILE;
```

この中で、_fileが実際のファイル番号です。0,1,2はシステムの標準入出力で使っているため、ユーザがファイルをオープンしたファイルは、3から連番になります。

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( fileName , 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
```

ファイルのオープンには、`fopen()`か、`fopen_s()`を使います。`fopen()`には引数が2つ必要です。最初の引数は、オープンするファイルのパスを文字列で指定します。2つ目は、ファイルをオープンするときのモードです。モードも文字列で指定します。モードについては、次のページで詳しく説明をします。

オープンに成功すると、**FILE**構造体へのポインタが返されます。

実際は、下記のように使います。

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

ちなみに、`fopen_s()`を使う場合は、下記のようになり、if文の条件が逆になるので注意をしてください。`fopen_s()` の場合戻り値はエラー情報となり、0がオープン成功です。

```
if ( (fopen_s(&fp, "sample_data1.txt", "r") ) == NULL){
    // オープン成功
}else{
    // オープン失敗
}
```

`fopen()`を使うか、`fopen_s()`を使うかは、客先のコーディングルールに従ってください。客先でのルールが最優先です。

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

2021 JTEC m.h

24

ファイルオープン時には、ファイル名(パス名)とオープンモードを指定します。

Windowsでは、ファイルのパスを「¥」(英語では「\」バックスラッシュ)で区切ります。しかし、「¥」記号は、C言語でエスケープ文字として使われているため、そのままでは正しいパスリストにはなりません。そのため、パスを書くときは、“¥¥”と、¥記号を2個書く必要があります。

ちなみに、Windows以外のLinux、Macなどでは、パスの区切り記号が"/"のため、そのまま"/"で区切って書きます。WebのURLも"/"で区切りますね。こうして見ると、マイクロソフトだけが区切り記号に「¥」を使っていることになります。

これには、歴史があります。初期のMS-DOSでは、現在のような階層化ディレクトリはサポートしておらず、コマンド引数のオプション指定に、「/」を使っていました。その後、MS-DOS V2.0で、階層化ディレクトリをサポートしたとき、過去のMS-DOSとの互換性から、オプション指定で使っていた「/」を使わず、¥(英語では「\」バックスラッシュ)を使うことになってしまいました。その時の決定が今で引き継いでいます。

さて、オープンのモードですが、下記のように、

"r","w","a"の3種類の基本モードがあります。

"r":読み込みモード、

"w":書き込みモード、

"a":追加書き込みモード

"w"と"a"の違いは、ファイルにデータを書き込むとき、すでにファイルにデータがあるとき、"w"はすでにあるデータを無視してファイル先頭から上書きとなり、"a"はすでにあるデータに追加で書き込まれます。新規にファイルを作成する場合は、どちらも同じになります。

そして、基本モードに対して、ファイルの形式と動作の指定をします。

基本モードに、“b”を付けるとバイナリ形式での読み書きに、“+”を付けると更新モードになります。

"b"を付けると0~0xFFのすべてデータをバイナリとして読み書きができます。しかし、“b”を付けるとテキストファイルと見なし、0などの読み込みはできません。

実際の指定は、基本モードと組み合わせて、“rb”とか“w+”のように指定します。

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も読み込まれている)
- 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)は読み込まれません。
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。

2021 JTEC m.h

25

ファイルオープンに成功したら、実際にファイルに対して、Read / Writeの操作をするわけですが、そのとき、適正なRead / Write関数を使う必要があります。

C言語研修では、

テキストを1行読込む fgets() と、バイナリデータを指定したバイト数を読込む fread() を使います。

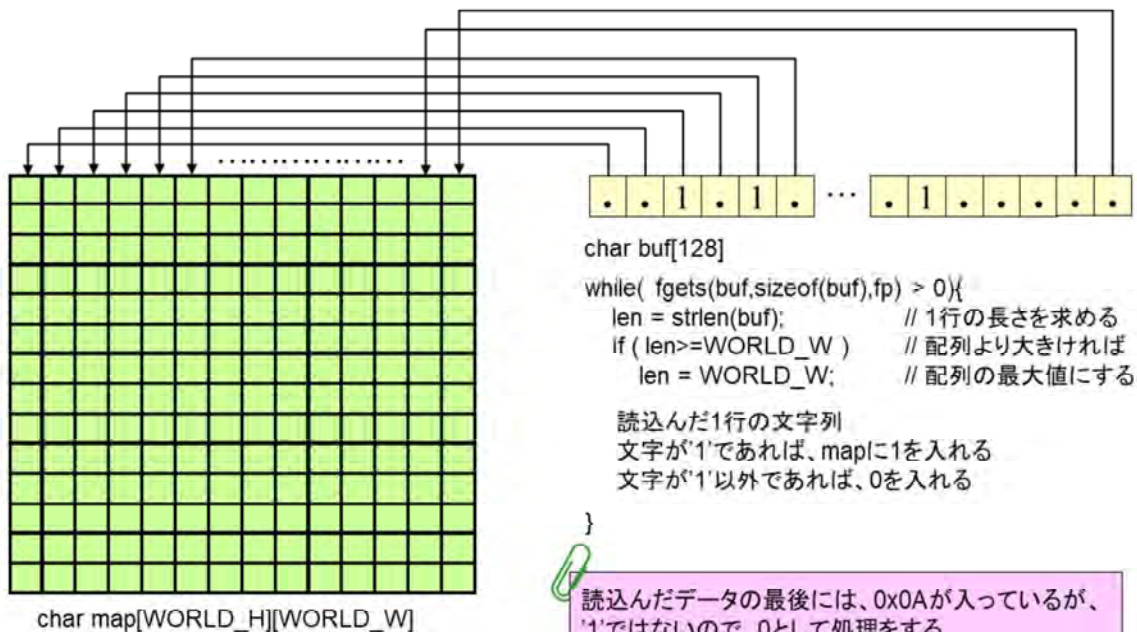
また、読み込みと書き込みは、対になった関数があります。まとめて覚えると良いでしょう。

- Fgetc() ⇔ fputc() 1文字の読み書き
- fgets() ⇔ fputs() 1行(¥nまで)の読み書き
- fread() ⇔ fwrite() バイナリの読み書き
- fscanf() ⇔ fprintf() 書式指定での読み書き

【注意】 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)は読み込まれません。
そして、OSの内部にLF(0x0A)が残っている所以注意が必要です。

初期データをファイルから読み込み初期化する

ファイルから初期データを読み込み、mapを初期化する



2021 JTEC m.h

26

この図は、ファイルから初期データを読み込んで、2次元配列のmapを初期化のイメージです。今まで、2次元配列のアクセスは、for文の2重ループでやっていました。ファイルから読み込んで初期化をする場合、ファイルのデータが何行あるか分かりません。それで、縦のループは、for文でなく、while文を使って、1行読むごとに縦番号を+1していきます。そのため、while ループの前に、縦番号の変数を0にします。

その後、whileループの中でファイルから1行を読み込みます。実際の読み込みは、while文の中に書いて、結果が0以上の間ループします。

```
while( ( bufに1行の読み込み) > 0 ){
```

このとき、1行の長さも何文字あるか分かりません。そこで、1行読んだら strlen()関数を使って、1行の文字数を調べます。その1行の文字数が横方向のループ最大数になります。

しかし、文字数が配列の大きさを超えている場合は、配列の領域を超えてアクセスすることになるので、1行の長さが配列の最大値より大きいかを調べて、大きければ1行の長さを強制的に最大値にします。

横ループの最大数が求まれば、for文のループで読んだ1行(この例ではbufという1次元配列)を1文字ずつ調べ、「1」であれば、mapの同じ位置に1を、1以外であれば0を代入します。

このとき、読んだテキストは、「文字」です。単に1と比較してはいけません。「文字の1と比較します」。

読んだ1行の処理が終わったら、縦番号を +1 します。

```
mapの内容をすべて0クリアする ◀ 重要(ファイルのデータは、配列より小さい場合もあるよ)
縦番号の変数 = 0;
while( (bufに1行の読み込み) > 0){
    1行の長さを調べ、必要であれば補正する。
    for文で1行の長さまでループ{
        読んだテキストが「1」という文字なら、mapに1を、それ以外なら0を代入
    }
    縦番号を +1 する
}
```

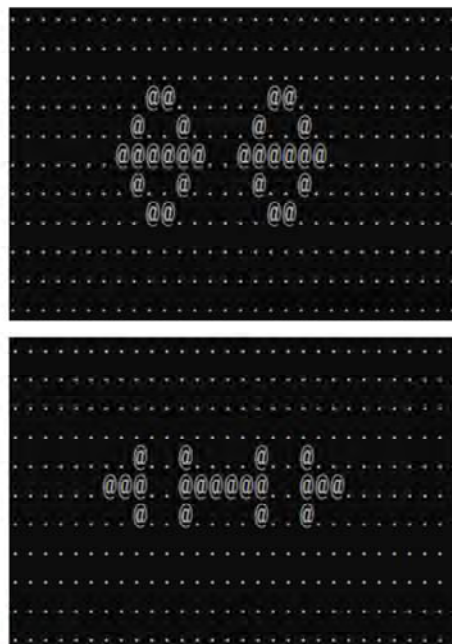
初期値のパターン(sample_data1.txt)

初期値のデータパターン

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

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。

実行結果



上記の2つのパターンが繰り返される

これは、sampe_data1.txt の内容です。

このパターンで初期化すると、右の例のように、2種類のパターンが交互に繰り返され続けます。

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



これは、sampe_data2.txt の内容です。

このパターンで初期化すると、右の例のように、上部からグライダー? が永遠に放出され続けます。

Step4:2つの初期化を切り替える

マップの初期化を、最初にやった乱数での初期化と、ファイルから読み込みの2種類の関数ができました。せっかくなので、`scanf()`で入力した文字列に応じて、これらを切り替えで実行できるようにします。

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 = ファイルから読んで初期化(map, inbuf);

if ( result != 0 )
    初期化失敗
```

2021 JTEC m.h

30

九九でもscanf()を使い、"%d"で、10進数として読み込みました。ここでは、"%s"として、文字列として入力します。

そして、入力した文字列の最初の文字が、0~9 までであれば、乱数モードとして乱数で初期化を行い、それ以外であれば、ファイル名とみなし、ファイルからの読んで初期化をします。

乱数モードでも、ファイルモードでも、入力したテキストを引数としてそのまま渡します。

そして、乱数モードでは、数字の文字コードを数値に変換して、srand()で乱数の種を設定します。実際の例は、下記ようになります。

```
int init_rand(char map[WORLD_H][WORLD_W], char *inbuf){
    int seed;
    seed = atoi(inbuf);    // 文字列を数値に変換する
    srand(seed);
    :
    :乱数による初期化(すでに作成済み)
    :
    return 0;              //乱数モードでは失敗ないので、常に0で戻る
}
```

一方、ファイルモードの場合は、入力された文字列をファイル名としてオープンします。

```
int init_file(char map[WORLD_H][WORLD_W], char *inbuf){
    inbufのファイル名でオープン
    オープンに失敗したら、return -1で戻る;
    :
}
```

参考資料:Windowsによる画面制御

エスケープシーケンスが正しく動作しなかった場合、**Windows**がサポートする画面制御機能で同様な制御が可能です。

エスケープシーケンスによる制御は、一般的な端末でも動作しますが、**Windows**の機能を使った場合は、**Windows**の環境のみでの動作になります。

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);
```

2021 JTEC m.h

32

エスケープシーケンスが正しく動作しない場合は、下記のようにWindowsの機能を使ってください。

```
#include <Windows.h>
```

```
int main(void) {
```

```
    ローカル変数の定義
```

```
    :
```

```
    COORD coord;
```

```
    HANDLE hStdout;
```

```
    hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
```

```
    system("cls"); // 画面のクリア
```

```
    :
```

```
    :
```

```
    for ループ{
```

```
        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
```

```
        マップの表示();
```

```
        世代を進める();
```

```
    }
```