ソフトウェア2
[第1回]
(2020/11/19)

齋藤 大輔

本日のメニュー

- 講義概要紹介
  - スケジュール／講義ページ／成績評価
- 今日の題材（C言語復習）
  - テキスト入出力
  - ライフゲーム
- プログラム解説
  - プログラムの構造を読み解いていこう
- 実習
- ファイル入出力について
- レポート課題について

ソフトウェア2

- C言語について、より発展的な内容に取り組む。
  - 基礎的文法から入門編へ
- 少し規模の大きなプログラム（アプリケーション）のプログラミングを通して以下を学ぶ。
  - データ構造・アルゴリズムの基礎
  - プログラムの設計

スケジュール

- 本年度は全7回開講
  - 11/19, 11/26, 12/03, 12/10, 12/17, 12/24, 01/21
  - 01/21 は課題を出題しないフォローアップ講義の予定
- 木曜3, 4限／ Zoomにて開催 / Slack併用

講義用ページ

- URL

  https://www.gavo.t.u-tokyo.ac.jp/~dsk_saito/lecture/software2

- 進行はスライドを中心に行うが適宜ページも参照

成績評価

- 全6回のレポート課題により評価
  - 基本課題については全て採点対象
  - 発展課題については全6回中上位3回分を採用
- レポート期限は締切日の23:59:59 （日本時間）
- 出席はとりません
- ITC-LMS の登録を確認すること

テキスト入出力について

- 端末出力の発展版について
- scanfの取り扱いについて
- FILE型/文字列型をターゲットとしたテキストI/O

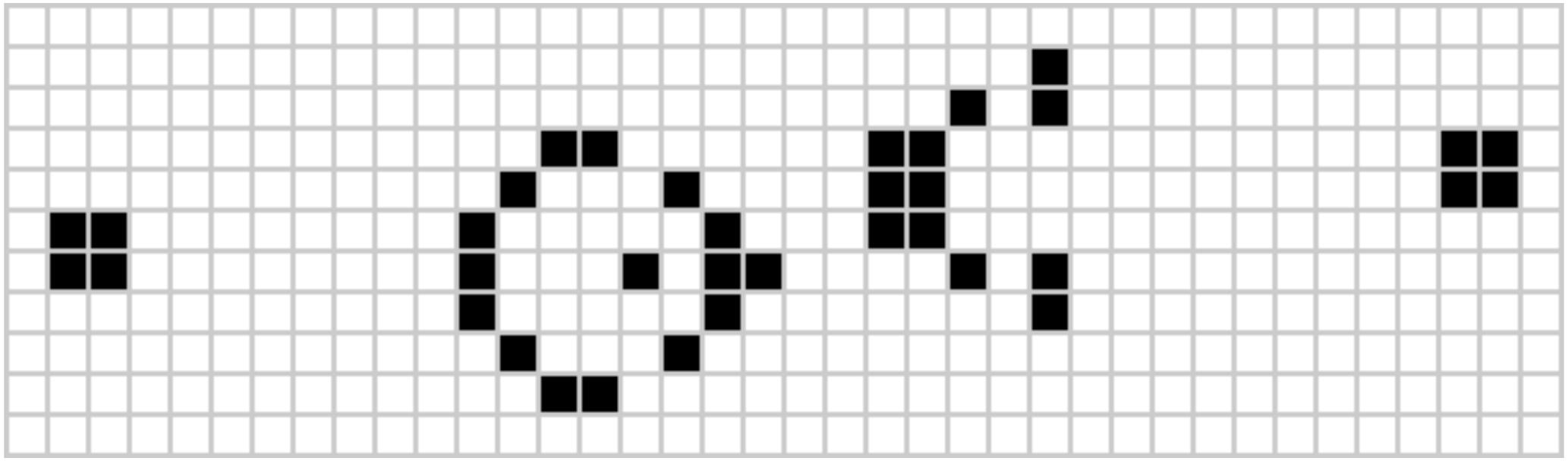
しばらくWebページベースで説明します。

本日のメニュー

- 講義概要紹介
  - スケジュール／講義ページ／成績評価
- **今日の題材（C言語復習）**
  - **ライフゲーム**
- プログラム解説
  - プログラムの構造を読み解いていこう
- 実習
- ファイル入出力について
- レポート課題について

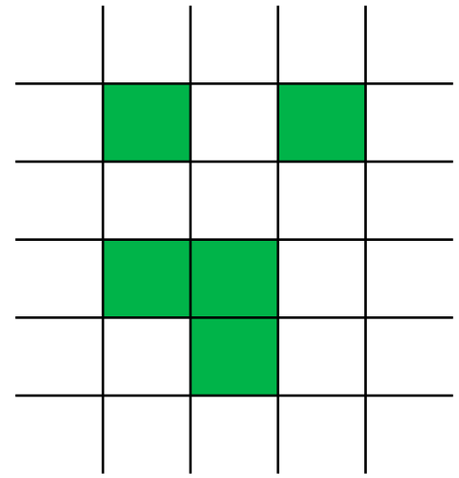
ライフゲーム

- Conway’s Game of Life (Conway 1970)
  - イギリスの数学者 John Conway が提案
  - 生命の誕生・衰亡・変化のシミュレーション



http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life

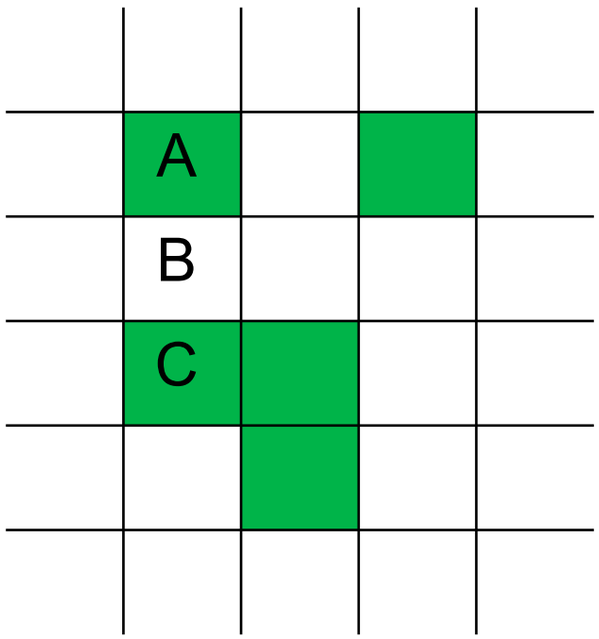
ライフゲームのルール



- 2次元グリッド上にセル（cell）が存在
- 次の世代
  - 生きているセルが有る場所
    - 周囲に生きているセルが2個または3個存在するならばそのまま
    - そうでなければセルは消滅
  - 生きているセルが無い場所
    - 周囲に生きているセルがちょうど3個存在するならばセルが誕生
    - そうでなければそのまま

例えば...

- A, B, C の位置にあるセルはそれぞれ次世代にどう変化するか？



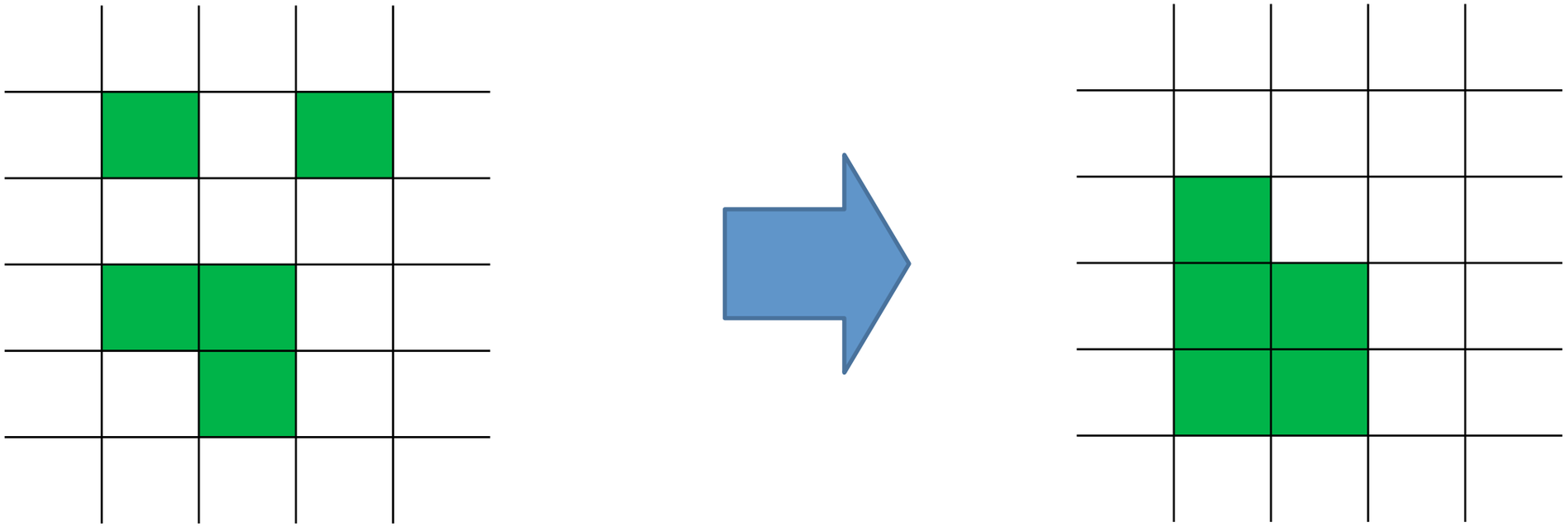
A: 周囲にセルが存在しないので消滅

B: 周囲にセルが3個存在するので新しいセルが誕生

C: 周囲にセルが2個存在するのでそのまま

例えば…

- １世代進むと結局、



これが　　こうなる

これを繰り返し実行するとセルはどのような状態になるのか

本日のメニュー

- 講義概要紹介
  - スケジュール／講義ページ／成績評価
- 今日の題材（C言語復習）
  - ライフゲーム
- **プログラム解説**
  - **プログラムの構造を読み解いていこう**
- 実習
- ファイル入出力について
- レポート課題について

サンプルコードを動かす

- コンパイル

```
% gcc -o lifegame -Wall life.c -L. -lgol
% gcc -o calibration calibration.c
```

- 表示領域の調整

```
% ./calibration
```

  ターミナルのサイズをマウスで調整（縦に大きく）してグリッド全体が表示されるようにする

- 実行

```
% ./lifegame
% ./lifegame gosperglidergun.lif
```

実行を停止する場合はCtrl-c

サンプルコードlife.c

- まずは細かい中身を読む前に全体像を俯瞰する
  - 50行程度のプログラム（それほど大きくない）
  - main関数（アプリケーションの本体）
  - 3つの関数（標準ではないもの, gol.h 参照）
    - void init_cells(const int, const int, int cell[][], FILE*)
    - void print_cells(FILE*, const int, const int, int cell[][])
    - void update_cells(const int, const int, int cell[][])

関数名からおよその機能が類推できそう？

メイン関数を俯瞰

```
int main(int argc, char **argv)
{
  FILE *fp = stdout ;
  const int height = 40;
  const int width = 40;
  int cell[height][width];

  // 中略

  print_cells(fp, 0, height, width, cell);
  // 中略

  for (int gen = 1 ;; gen++) {
    update_cells(height, width, cell);
    print_cells(fp, gen, height, width, cell);
    // 中略

  }

  return EXIT_SUCCESS;
}
```

グリッドの準備?

初期化（init）？

updateの繰り返し？

具体的な実装が未定でも、おおよそ全体の処理の見通しがつくはず

実装はどこに？

- コンパイルオプションに着目

```
% gcc -o lifegame -Wall life.c -L. -lgol
```

このオプションで当該フォルダ内にある、libgol.a を参照する
（自作のライブラリを読み込んでいる）

- 使用されている関数の入出力
  - gol.h に関数プロトタイプが記載されている
- 今日の目標は上記のオプションなしで実行できるように必要な関数を実装すること

```
% gcc -o mylifegame -Wall mylife.c
```

- 最終的に`gol.h` はコメントアウトする

実装方針

- 以下の関数を実装する（接頭辞my_）
  - void my_init_cells(const int, const int, int cell[][], FILE*)
  - void my_print_cells(FILE*, int, const int, const int, int cell[][])
  - void my_update_cells(const int, const int, int cell[][])
- 関数を一つ実装したら、他の関数を既存のものに固定して動かし、逐次動作確認をして進めること
  - その時点ではライブラリのコンパイルオプションは必要

life.c

- ヘッダーファイルの読み込み（インクルード）

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include ”gol.h”
```

  - コンパイラ（正確にはプリプロセッサ）が、個々の #include 文をヘッダーファイルで置き換える
- ヘッダーファイル
  - 関数の宣言やマクロの定義が記述されているファイル
  - 例えば、printf 関数をプログラム中で使いたい場合、その宣言の記述があるヘッダーファイル stdio.h をインクルードする必要がある
  - OS標準の場所の他、独自のヘッダーファイルも設置可能

life.c

- const を用いた定数の宣言

```
const int height = 40;
const int width = 70;
```

  - const を用いることでプログラム中で書換えがおこるとコンパイルエラーとなる
  - プログラム上定数として扱えるものはこのように取り扱うとよい

life.c

- セルの状態変数（2次元配列）

```
int cell[height][width];
```

  - 整数型の2次元配列でセルの状態を保持
    - 大きさは height × width つまり 40 × 70 = 2800
- 配列を0で初期化（後のfor分）：生きている場合は1
- C99より、変数による動的なサイズ決定が可能
  - ただし宣言以前に値が決まっている必要あり
  - 本当に動的に確保したい場合はポインタを使う

init_cells() 関数

- いくつかのセルを生きている状態にする
- 現在の初期値はdefault.lif に対応
- 1行に横のインデックス、縦のインデックスの順で生きているセルの座標を記述する
  - 直接配列のインデックスとすればよい
  - cell配列のインデックスの順序に注意
- Life 1.06 というフォーマット
  - https://www.conwaylife.com/wiki/Life_1.06
  - 先頭が”#” の行はコメントとして無視する

print_cells() 関数

- セルを表示する
- ヒントはcalibration.c
- バッファに出力が蓄積しないようにfflush(fp) を実行

count_adjacent_cells() 関数

- 周囲のセルの数を数える
  - update_cells() 内で利用する
- 実装する際には配列インデックスの境界条件に注意
  - 隣が存在しない領域である可能性がある

周囲 8 マス

(i-1, j-1)	(i-1, j)	(i-1, j+1)
(i, j-1)	(i, j)	(i, j+1)
(i+1, j-1)	(i+1, j)	(i+1, j+1)

update_cells() 関数

- 次世代のセルの状態を計算
- 周辺のセルの個数に基づいて状態を更新
- 互いに依存関係があるため一時変数となる配列に保存後、一斉に更新するのが望ましい。

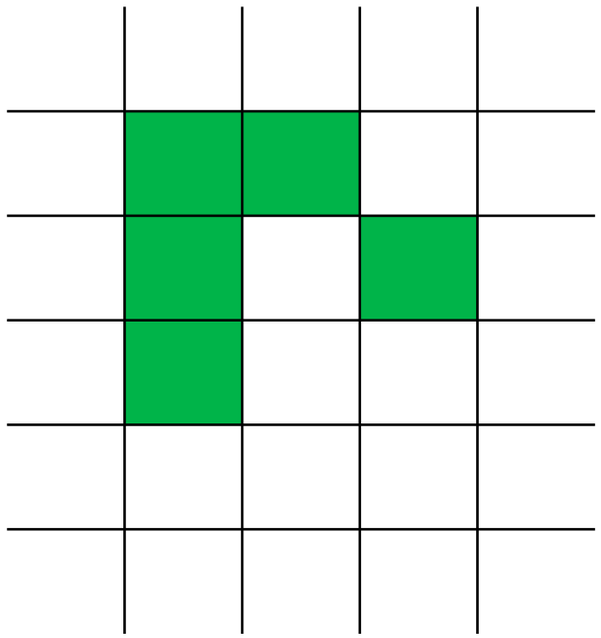
本日のメニュー

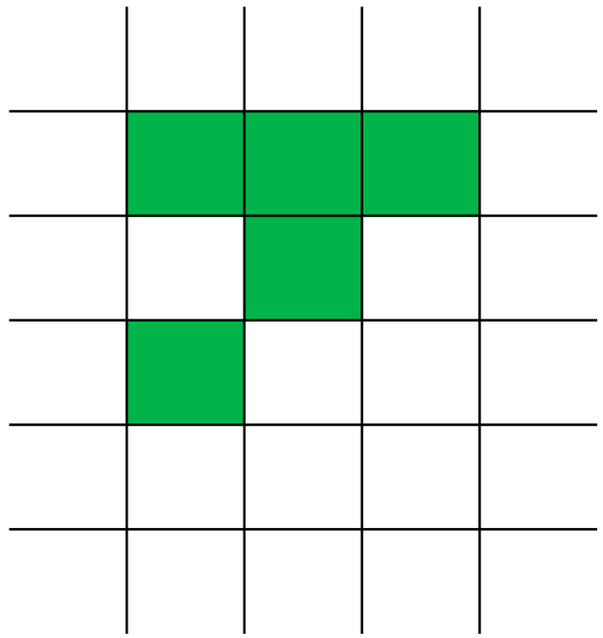
- 講義概要紹介
  - スケジュール／講義ページ／成績評価
- 今日の題材（C言語復習）
  - ライフゲーム
- プログラム解説
  - プログラムの構造を読み解いていこう
- **実習**
- ファイル入出力について
- レポート課題について

実習

1.各関数を実装する。

2.init_cellsでの初期値として以下の初期配置を試す。





FILE型

- FILE型
  - ファイルの入出力を行うためのデータ構造
  - 構造体で実装されていることが多く
    FILE構造体と呼ぶこともあり
  - ファイルアクセスに関する情報が保存されている
    - どのファイルのどの場所をアクセスしているか等

fopen() 関数について

- ファイルを開く

```
FILE *fp;

if ((fp = fopen(“default.dat”, “r")) == NULL) {
  fprintf(stderr, "error: cannot open a file.\n");
  return 1;
}
```

- fopen() 関数は、第１引数でファイル名、第２引数でファイルアクセスのモードを指定
  - モード：　“r”（読み込み）, “w”（書き出し）, “a” （追記）
- ファイルのオープンに成功した場合はファイルポインタが返ってくる
  - 以後、そのファイルポインタを使ってそのファイルに対する読み書きを行うことができる
- ファイルのオープンに失敗するとNULL が返ってくる

if 文の書き方について

- ファイルを開く

```
FILE *fp;

if ((fp = fopen(“default.dat”, "r")) == NULL) {
  fprintf(stderr, "error: cannot open a file.\n");
  return 1;
}
```

- 上記の if 文は以下のように書いたのと同じ

```
fp = fopen(“default.dat”, “r");
if (fp == NULL) {
  fprintf(stderr, "error: cannot open a file.\n");
  return 1;
}
```

- 代入式の値は代入演算された後の値になるので

stderr について

- ファイルを開くのに失敗すると、標準エラー出力にメッセージを出力して、プログラムを終了する

```
fprintf(stderr, ”error: cannot open a file. \n”);
return 1;
```

- 標準エラー出力
  - 普通はシェル上でメッセージを見ることになる
  - stderr: 標準エラー出力へのファイルポインタ
- main関数の返り値
  - 正常終了ならゼロ、異常終了ならゼロ以外の値を返す

ファイル入出力のための関数

- ファイルアクセス関数

```
FILE *fopen(char *name, char *mode);
```
ファイルを開く

```
int fprintf(FILE *fp, const char *format);
```
ファイル`fp`へ書式付で文字列を出力

```
int fputc(int c, FILE *fp);
```
ファイル`fp`へ文字`c`を出力

```
int fclose(FILE *fp);
```
ファイルを閉じる

readfile.c

- ファイル入力のサンプルプログラム
  - ファイルを開く
  - 各行の文字数を表示

- 標準ライブラリ関数
  - `fgets(char *s, int size, FILE *fp);`
    ファイルから１行読み込んでバッファに保存
  - `size_t strlen(const char *s);`
    文字列の長さを取得

コマンドライン引数

- プログラムの実行時にパラメータやオプションを指定できる

コレ

```
% ./a.out 0.1
```

  - 文字列へのポインタの配列として得られる
  - 数値に変換したいときは atoi, atof 等を利用

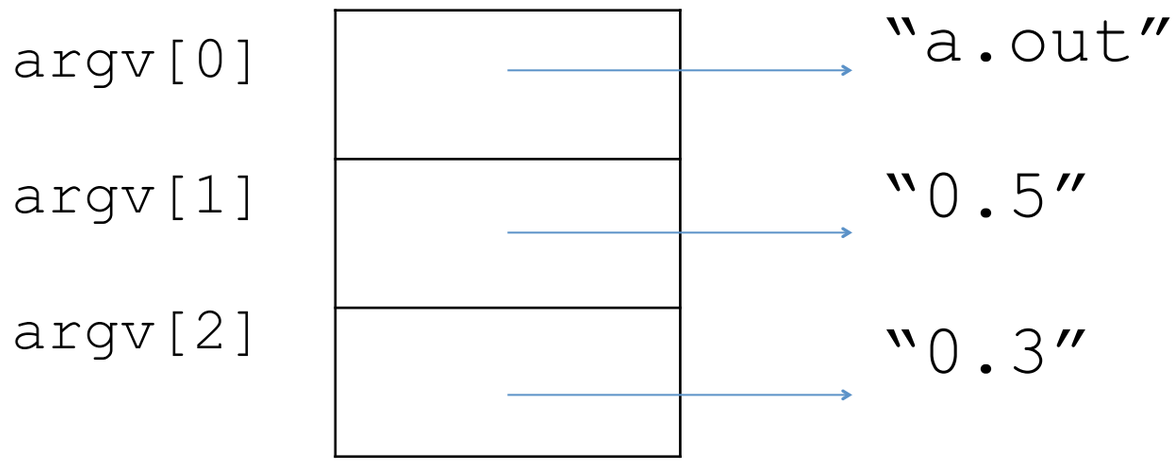
（“a.out” も含めた）引数の数

```
int main(int argc, char *argv[])
{
        for (int i = 0; i < argc; i++) {
                printf("%d %s\n", i, argv[i]);
        }
}
```

char *argv[] とは？

- 文字（列）へのポインタの配列

```
% ./a.out 0.5 0.3
```



- よくある別な書き方

```
int main(int argc, char **argv)
```

文字（列）へのポインタのポインタ

本日のメニュー

- 講義概要紹介
  - スケジュール／講義ページ／成績評価
- 今日の題材（C言語復習）
  - ライフゲーム
- プログラム解説
  - プログラムの構造を読み解いていこう
- **実習**
- ファイル入出力について
- レポート課題について

レポート課題（締切11/25）

1. 各世代で存在するセルの比率を表示するように “life.c” を拡張せよ
   - プログラム名は “mylife1.c とする
2. 初回配置パターンがランダムになるように修正せよ
   - プログラム名は “mylife2.c とする
   - 生きたセルの割合がほぼ 10% になるようにすること
3. 初期配置パターンをRLEフォーマットのファイルから読み込めるように拡張せよ
   - プログラム名は “mylife3.c” とする
   - RLE: https://www.conwaylife.com/wiki/Run_Length_Encoded
   - ファイルからの入力に関しては “readfile.c” を参照
4. ［発展課題］ライフゲームのルールを適当に変更しその振る舞いを確認せよ
   - プログラム名は “mylife4.c” とする
   - どのようなルールに変更し、その結果として、どのような振る舞いが見られたかを簡単に説明すること
   - ルールは大幅に変更しても構わない（生物種を増やす、地形効果を導入、etc）

課題にあたって

- ライフゲームの本家Wiki に入力できるサンプルがあるので、これをテストに用いるとよい

  https://www.conwaylife.com/wiki/Main_Page

  - Plaintext 形式からLife 1.06 への変換等をやってみる

課題の提出方法（ITC-LMS）

- 形式: ファイルアップロード
  - 全てのプログラム/ファイルをまとめ、zipで圧縮し提出
  - git archive コマンドを用いる
    - 次回以降のgit での提出は鋭意準備中
  - SOFT-11-19-NNNNNNNN.zip または
    SOFT-11-19-NNNNNNNN.tar.gz
    - NNNNNNNN 部分は学籍番号（ハイフン除く）
    - JについてはJ??????? のようにしてください。
- 課題について
  - [基本課題] 毎回提出
  - [発展課題] 成績計算に全６回中上位３回分を採用する

乱数発生について

- `rand()` 関数
  - ヘッダーファイル： stdlib.h
  - 0 ～ RAND_MAX （非常に大きな整数）の範囲で疑似乱数を返す
  - 0以上5未満の整数乱数を得たいときは

```
int r = rand() % 5;
```

  - 浮動小数点数乱数を得たい場合は、たとえば、

```
double r = (double)rand() / (RAND_MAX+1);
```

とすると [0, 1) の範囲の乱数が得られる

乱数発生について

- 発生する疑似乱数の系列はつねに同じ
- `srand()` 関数
- 疑似乱数の seed （種）を設定
- プログラムの実行のたびに違う乱数系列を発生させたい場合はmain() の先頭で `srand(time(NULL));` を１度実行すればよい
- 確率を制御する場合、乱数の大小で挙動を変える