# 情報処理実習

## 第14回：まとめ

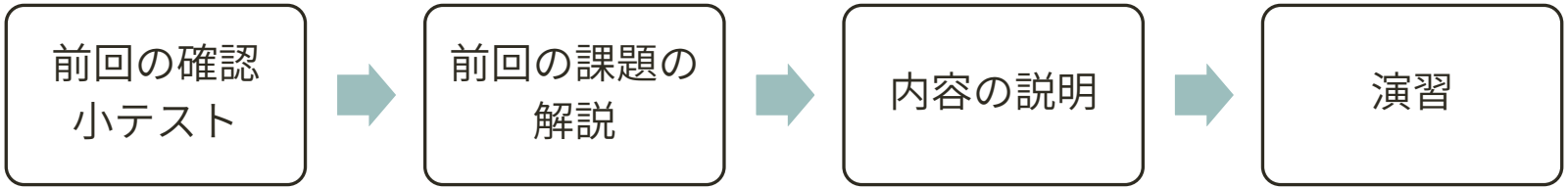
2018年1月15日(月)

担当：佐藤

# アナウンス

- 授業前にやること
  - 計算機にログオン
  - CoursePowerにログイン
- 注意事項
  - 教室内飲食禁止
  - 原則として，授業中ではなく休み時間にトイレにいきましょう

# 授業の進め方(確認)



# 基本事項の確認小テスト


① CoursePowerにログインして，すぐに解答が始められるよう準備する
② 開始の合図があるまで，解答を始めずに待機する

注意

「閉じる」ボタンは決して押さないこと！

# 前回課題の解説

# 課題1

```c
#include <stdio.h>

int main()
{
   FILE *fp;
   char f[] = "data.txt";

   /* 書き込みモードでファイルオープン&エラー処理 */
   fp = fopen(f, "w");
   if (fp == NULL) {
      printf("%s をオープンできません¥n", f);
      return 1;
   }

   /* ファイルクローズ */
   fclose(fp);

   return 0;
}
```

- ファイルオープンの手順
  - ① ファイルポインタを宣言
  - ② ファイルポインタをストリームにする
    (fopen()の戻り値をファイルポインタへ代入)
  - ③ エラー処理

明示的に構築したすべてのストリームについて行う
(stdinやstdoutは自動的にクローズされる)

data.txt

```
10 20 30
```

中身が消える！

# 課題2(1)

- 主な解説は第13回のスライドで既に行っているため省略
- <補足(1)>
  - 「fscanf()はファイルの中身を1行だけ読み込む」という説明は，fscanf()の書式(書式制御文字列，書式指定文字列，制御文字列，書式文字列，format)がファイルの中身の1行を読み込む形式になっていることを前提としている
    - printf()族の関数またはscanf()族の関数の引数の中の"で囲まれた部分("含む)を書式という
    - printf()族の書式は「期待される出力形式のイメージ」，scanf()族の書式は「期待される入力形式のイメージ」である

1行まとめて読み込み

```
fscanf("%d %d %d", &v[0], &v[1], &v[2]);
```

data.txt

```
10 20 30
```

形式を統一！

別解(数値ごとに読み込み)

```
for (i = 0; i < 3; i++) {
   fscanf("%d", &v[i]);
}
```

%dは先頭の空白類文字を無視する．そのため，「␣20」，「␣30」は20，30にそれぞれ変換される．よって，ループは3回行えばよい(空白文字を読み捨てるためのループは不要)

# 課題2(2)
## 【発展】

- <補足(2)>
  - 課題2は，fscanf()でdata.txtの中の数値を読み込むという問題である．しかし，ストリームを介してファイルとプログラム間で授受できるのは(「ストリーム」の中を「流れる」ことができるのは)文字(1バイトのデータ)だけである．そのため，ファイル中の数値をプログラムに授けることを実現するには，入力ストリームから受け取った文字の並びをプログラム側で数値に変換するしかない．数字(0〜9)を数値に変換する方法は文字処理の回で学んだが，この方法を使うと2つ以上の数字の並びを数字に対応した1桁の数値の並びに変換するため，2桁以上の数値を読み込めない．文字の並びを2桁以上の数値に変換することは，おなじみの%dを使うだけでよい．%d変換指定子は10進数字の並びを数値に変換する．data.txtは1行のみからなるテキストファイルであり，その1行は'1''0''␣''2''0''␣''3''0''¥n'という文字の並びである(1行だけのファイルであってもテキストファイルならば末尾に'¥n'がある)．読み込む対象が1行だけならば末尾の'¥n'を無視して書式を書いても差し支えない．そのため，サンプルプログラム(2)では「"%d %d %d"」とした．仮に複数行ある場合，この書式で次にfscanf()を実行すると，次の行の先頭に読み込まれなかった'¥n'をつけた文字の並びを読み込む．仮にdata.txtに2行目が存在し，それが「40 50 60」であるとすると，'¥n''4''0''␣''5''0''␣''6''0''¥n'という文字の並びを読み込もうとする．%dは先頭の空白類文字は無視するので，最初の%dは先頭の'¥n'を無視して'¥n''4''0'を40に変換する．後は1行目と同じ．なお，"%d %d %d¥n"と書いておくと上述の'¥n'の読み飛ばしがなくなるが結果は同じ．また，scanf()族の関数の書式の中では，連続する空白類文字は空白類文字1文字として扱われるため，例えば"%d %d %d¥n¥n¥n"は"%d %d %d¥n"と同じ．また，scanf()族の関数の書式の中では，すべての空白類文字は同一に扱われる．そのため，例えば"%d %d %d¥n"は空白文字と改行文字を入れ替えた"%d¥n%d¥n%d"と同じである．ただし，紛らわしいためこのような記述は控え，なるべく空白文字も含めた期待される入力形式のイメージに忠実な書式にするべきである

# 課題2(3)

別解(数値ごとに読み込み)

```
#include <stdio.h>

int main() {
   FILE *fp;
   int x, i;
   char f[] = "data.txt";

   /* 読み込みモードでファイルオープン&エラー処理*/
   fp = fopen(f, "r");
   if (fp == NULL) {
      printf("%sをオープンできません¥n", f);
      return 1;
   }

   /* data.txtから数値の読み込みと表示を繰り返す */
   for (i = 0; i < 3; i++) {
      fscanf(fp, "%d", &x);
      printf("%d ", x);
   }
   printf("¥n");

   /* ファイルクローズ*/
   fclose(fp);

   return 0;
}
```

# 課題3(1)

## 5行5数値のデータ入力を前提とした解答例

```
#include <stdio.h>

int main() {
   FILE *fpr, *fpw;
   int v[5], i;
   char fr[] = "multiRowData.txt", fw[] = "result.txt";
   double avg;

   fpr = fopen(fr, "r");
   if (fpr == NULL) {
      printf("%sをオープンできません¥n", fr);
      return 1;
   }

   fpw = fopen(fw, "w");
   if (fpw == NULL) {
      printf("%sをオープンできません¥n", fw);
      return 1;
   }

   for (i = 0; i < 5; i++) {
      fscanf(fpr, "%d %d %d %d %d", &v[0], &v[1], &v[2], &v[3], &v[4]);
      avg = (double)(v[0] + v[1] + v[2] + v[3] + v[4]) / 5;
      fprintf(fpw, "%.2f¥n", avg);
   }

   fclose(fpr);
   fclose(fpw);

   return 0;
}
```

"%d%d%d%d%d"でもよい

1行あたり5数値であるため，ループごとに書式を変える必要がない

「1行まとめて読み込み，その行に記載された数値(整数)を配列に格納➔平均(実数)を計算➔小数点第2位まで表示」を5回繰り返す

# fscanf()の戻り値

- fscanf()はファイルからのデータの読み込みに失敗するとEOFを返す
- ファイルの終わりに達すると読み込むデータが存在しないため，読み込みは失敗する➔(ファイルの終わりに達するまで戻り値がEOFになることはないことを仮定すれば)戻り値がEOFになるまで繰り返しfscanf()を実行することでファイル中のデータを最初から最後まで読み込むことができる

文法

```
while (fscanf(ファイルポインタ, 書式, &変数1, …, 変数n) != EOF) { … }
```

  - 書式の書き方によって，1行ずつ，1…n数値ずつ，1…n文字ずつなど様々な読み込みができる

# 課題3(2)

## 可変行数可変個数値のデータ入力に対応する汎用性を持つ解答例

```
#include <stdio.h>

int main() {
   FILE *fpr, *fpw;
   int v[5], i, avg = 0;
   char fr[] = "multiRowData.txt", fw[] = "result.txt", c;

   fpr = fopen(fr, "r");
   if (fpr == NULL) {
      printf("%sをオープンできません¥n", fr);
      return 1;
   }
   fpw = fopen(fw, "w");
   if (fpw == NULL) {
      printf("%sをオープンできません¥n", fw);
      return 1;
   }
   for (i = 0; fscanf(fpr, "%d%c", &v[i], &c) != EOF; i++) {
      avg += v[i];
      if (c == '¥n') {
         fprintf(fpw, "%.2f¥n", avg / (double)(i + 1));
         avg = 0;
         i = -1;
      }
   }
   fclose(fpr);
   fclose(fpw);

   return 0;
}
```

vの要素数を1行あたりの数値の最大個数以上に設定

数値の後の'␣'または'¥n'を格納する変数

独力で解読してみよう！

# fgets()の戻り値

- fgets()はファイルからの文字列の読み込みに失敗するとNULLを返す
- ファイルの終わりに達すると読み込む文字列が存在しないため，読み込みは失敗する➔(ファイルの終わりに達するまで戻り値がNULLになることはないことを仮定すれば)戻り値がNULLになるまで繰り返しfgets()を実行することでファイル中の文字列を先頭行から最終行まで読み込むことができる

文法

```
while (fgets(文字配列名, 読み込む文字数, ファイルポインタ) != NULL) { … }
```

# 課題4

```c
#include <stdio.h>

int main() {
   FILE *fpr, *fpw;
   char str[256], fr[] = "kadai13_4.c", fw[] = "copy13_4.txt";
   int i;

   if ((fpr = fopen(fr, "r")) == NULL) {
      printf("%sをオープンできません¥n", fr);
      return 1;
   }

   if ((fpw = fopen(fw, "w")) == NULL) {
      printf("%sをオープンできません¥n", fw);
      return 1;
   }

   for (i = 1; fgets(str, 256, fpr) != NULL; i++) {
      printf(str);
      fprintf(fpw, "%2d: ", i);
      fprintf(fpw, "%s", str);
   }

   fclose(fpr);
   fclose(fpw);

   return 0;
}
```

ループ変数としてだけでなく行数としても使う

行数と合わせるため0ではなく1から始める

繰り返し文の中でループ変数を使う場合，while文ではなくfor文を使うとよい

# 学習内容の振り返り

# 学習してきた内容

- 式
- 変数
- printf文，scanf文
- 条件分岐
- 繰り返し処理
- 配列
- 関数
- 文字・文字列
- ファイル入出力

中間まとめのスライド参照

後半の学習内容の
要点を解説する

# 配列の要点

# 配列の宣言

文法

```
データ型 配列名[要素数] = { … };
```

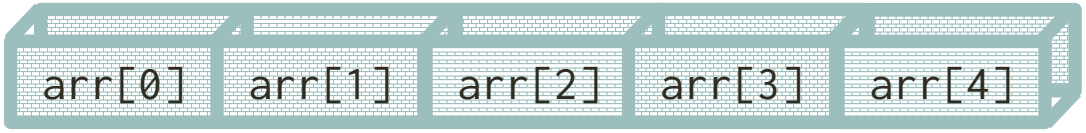
- { … }を**初期化子**といい，各要素の初期値を指定する
  - 初期化子は省略可能(「= { … }」を書かなくてよい)
  - 初期化子の省略は配列の宣言だけを行うことを意味する．その場合，個々の配列要素には不定値が入る➔必要に応じて初期値を設定する必要がある
- 初期化子により要素数が確定する場合，要素数を省略することが可能

# 配列宣言の例
## 初期化子を省略して宣言のみ行う場合

例

```
int arr[5];
```

- 上の宣言の後，arr[0]からarr[4]まで合計5個の配列要素(変数)を使用可能
- 使用可能な添字の範囲に注意
  - 配列要素は0番から始まるため，N個の要素を持つ配列を宣言すると0番からN-1番までの配列要素ができる(添字の範囲は0以上N-1以下)

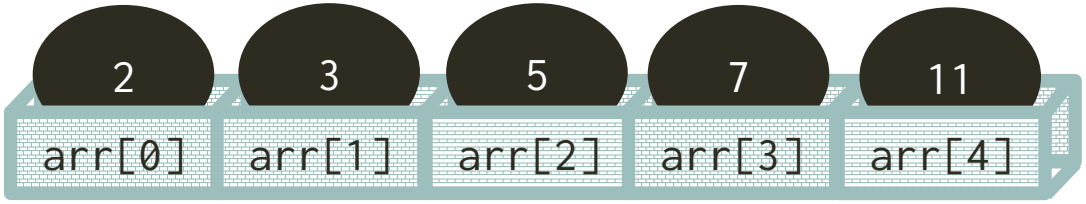


# 配列宣言の例
## 初期化子を省略しない場合

例

```
int arr[5] = {2, 3, 5, 7, 11};
```

- 上の宣言の後，arr[0]からarr[4]まで合計5個の配列要素(変数)を使用可能
- さらに，arr[0]からarr[4]まで各要素に初期値が設定される
- 初期値の数により要素数がわかるので，要素数を省略することも可能
  - `int arr[] = {2, 3, 5, 7, 11}; /* []の中に5と書いた場合と同じ */`



# 関数の要点

# 関数の定義

文法

```
戻り値の型 関数名(引数リスト) {
  …
  return 戻り値;
}
```

- 関数が値を返す場合，return文で返す値を指定する
- 関数が値を返さない場合，戻り値の型はvoidにしてreturn文を省略する
  - 戻り値を省略して「return;」と書いてもよいが，return文それ自体を書かずに省略した場合と同じことなのであえて書く必要はない

# 比べてみよう

```c
#include <stdio.h>

int main() {
   int a, b, c;

   printf("a = ");
   scanf("%d", &a);
   printf("b = ");
   scanf("%d", &b);
   while (b != 0) {
      c = a % b;
      a = b;
      b = c;
   }

   printf("GCD = %d¥n", a);

   return 0;
}
```

関数を使わないプログラム

```c
#include <stdio.h>

int gcd(int x, int y) {
   int z;

   while (b != 0) {
      c = a % b;
      a = b;
      b = c;
   }

   return x;
}

int main() {
   int a, b;

   printf("a = ");
   scanf("%d", &a);
   printf("b = ");
   scanf("%d", &b);
   printf("GCD = %d¥n", gcd(a, b));

   return 0;
}
```

関数を使ったプログラム

# 関数の存在意義

- よく使われる処理に名前をつけることにより，プログラム全体の見通しがよくなる
  - main関数が簡潔になるためプログラムの全体像が把握しやすくなる
- よく使われる処理を何度も書く代わりに，関数として一度書き，その処理を行う箇所で関数を呼び出せばよい．こうすることで以下のメリットがある
  - 書く手間が減る
  - ミスを修正する手間が減る
  - ある箇所では修正したのに，別の箇所では修正をし忘れるといった，修正し忘れのミスが減る
- 関数単位でプログラム作成の分業が可能になる(関数を「部品」のように扱える)
  - 関数の処理内容と関数の利用に必要な情報(戻り値の型，引数リスト，関数名)を共有することにより，既に別の誰かが作成した関数を使って(将来，別の誰かが作成する関数があることを前提として)自分の担当部分のプログラムが書ける

# 文字・文字列

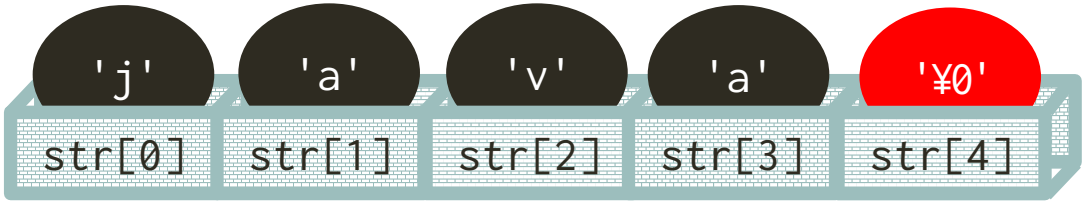
# 文字と文字列

- C言語は，文字1つ1つと文字が複数集まってできる文字列を区別する
- 文字1つを表すデータ型としてcharがある
- 文字列は**末尾がヌル文字('¥0')**の文字配列として表現される

# 文字列の例

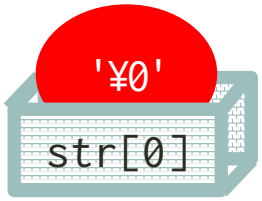
例

```
char str[] = "java";
```




例

```
char str[] = "";
```



# ファイル入出力の要点

# ファイル入出力の流れ

① ファイルオープン
1. ファイルポインタを宣言
2. ファイルポインタをストリームにする
   (fopen()の戻り値をファイルポインタへ代入)
3. エラー処理

② ファイルの読み書き

③ ファイルクローズ

# ファイルポインタ

文法

```
FILE *変数名;
```

- 上のように宣言された変数をファイルポインタと呼ぶ

# fopen()

文法

文字列定数

```
ファイルポインタ = fopen(ファイル名, モード);
```

- 代表的なモードは次の2つ(他にもモードはある)
  - "r": ファイルの読み込み
  - "w": ファイルの書き込み

# fscanf(), fgets(), fprintf()

- scanf()は，キーボードからデータを入力する関数
- printf()は，データをディスプレイへ出力する関数
- fscanf()は，ファイルからデータを入力する関数
- fprintf()は，データをファイルへ出力する関数
- fgets()は，ファイルから文字列を入力する関数

# fclose()

文法

```
fclose(ファイルポインタ);
```

- ファイル入出力の後，必ずファイルをクローズする必要がある

# 補足

# 初期化

- 変数に初期値を設定することを**初期化**という
- 初期化の方法
  1. 初期化を伴う宣言(**初期化**)
     - 変数を生成する時に値を入れる
  2. 代入
     - 生成済みの変数に値を入れる
- ①と②は初期値を設定するタイミングが異なる．そのため，便宜的に①のことを初期化といい②と区別する場合もある．初期値を設定するという意味の「初期化」なのか，初期化の方法のひとつである初期化を伴う宣言のことをいう「初期化」なのかは文脈で判断するしかない

①の文法

```
データ型 変数名 = 値;
```

この「=」は代入演算子ではない！

# 乱数

# 疑似乱数
【発展】

- ランダムな数列(**乱数列**)の要素(項)を**乱数(らんすう)**という
  - 例：サイコロの目
- 何らかのアルゴリズムを用いて生成される乱数を**疑似乱数(ぎじらんすう)**という
  - サイコロの目は予測不能．疑似乱数は予測可能
- コンピュータは疑似乱数を生成することしかできず，真の乱数を生成することはできない！

# 乱数の生成
【発展】

- C言語で乱数を生成するには，stdlib.hをインクルードすることで使用可能になるrand()を使う

文法

```
rand()
```

()の中には何も書かない

- rand()の戻り値の型はint，戻り値は0以上RAND_MAX以下のいずれかの整数
  - RAND_MAXは，stdlib.hをインクルードすると使用可能になる整数定数．値は処理系依存であるが，通常は32767

# rand()を用いた乱数列の作成(1)
【発展】

- rand()により生成される乱数の列は変わらない

```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int i;

   for (i = 0; i < 5; i++) {
      printf("%d ", rand());
   }
   puts("");

   return 0;
}
```

rand()を使うため

rand()を実行するごとに新しい乱数が戻り値として呼び出し元へ返される

実行結果(1回目)

```
41 18467 6334 26500 19169
```

実行結果(2回目)

```
41 18467 6334 26500 19169
```

実行結果(3回目)

```
41 18467 6334 26500 19169
```

# 乱数の種
【発展】

- rand()で生成される乱数列を変更するには，stdlib.hをインクルードすると使用可能になるsrand()を使う
- srand()に渡す整数を(乱数の)種(たね)という

文法

```
srand(整数);
```

種

# srand()による乱数列の変更
【発展】

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
   int i;

   srand(1);
   for (i = 0; i < 5; i++) {
      printf("%d ", rand());
   }
   puts("");

   return 0;
}
```

種 == 1

実行結果

```
41 18467 6334 26500 19169
```

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
   int i;

   srand(2);
   for (i = 0; i < 5; i++) {
      printf("%d ", rand());
   }
   puts("");

   return 0;
}
```

種 == 2

実行結果

```
45 29216 24198 17795 29484
```

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
   int i;

   srand(3);
   for (i = 0; i < 5; i++) {
      printf("%d ", rand());
   }
   puts("");

   return 0;
}
```

種 == 3

実行結果

```
48 7196 9294 9091 7031
```

# srand()とtime()による乱数列の変更
【発展】

- srand()の種を変更すると異なる乱数列になることを確認した．しかし，種が同じならば同じ乱数列になるため，プログラムが実行されるごとに種を変更する仕組みが必要
- time.hをインクルードすることで使用可能になるtime()の戻り値をsrand()の種とすることにより，毎回異なる乱数列にすることができる
  - time()の戻り値は，グリニッジ標準時の1970年1月1日0時0分0秒から現在までの経過時間(秒)

文法

```
srand((unsigned)time(NULL));
```

# rand()を用いた乱数列の作成(2)
【発展】

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
   int i, j;

   for (i = 0; i < 5; i++) {
      srand((unsigned)time(NULL));
      for (j = 0; j < 5; j++) {
         printf("%d¥t", rand());
      }
      puts("");
   }

   return 0;
}
```

rand(), srand()

time()

srand()で乱数列を変更してから項数5の乱数列を表示することを5回繰り返す

実行結果

```
5149    7100    2906    29764   28838
5149    7100    2906    29764   28838
5149    7100    2906    29764   28838
5149    7100    2906    29764   28838
5149    7100    2906    29764   28838
```

乱数列が変更されない???

- time()の戻り値は秒なので「1秒間」値が変わらない．for文は1秒間に何回もループするので，その間srand()の種は変わらない

# rand()を用いた乱数列の作成(3)
【発展】

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
   int i, j;

   srand((unsigned)time(NULL));
   for (i = 0; i < 5; i++) {
      for (j = 0; j < 5; j++) {
         printf("%d¥t", rand());
      }
      puts("");
   }

   return 0;
}
```

srand()は，プログラムの冒頭で1回だけ実行すればよい！

実行結果

```
483     15819   1277    16921   2143
23468   21144   12748   17658   17154
1122    10059   32036   9420    17259
13109   7059    11064   4669    2751
2615    15300   5741    31481   26737
```

# 乱数の値の範囲
【発展】

- rand()で生成される乱数の値の範囲を「0以上1未満」にするには，次のように書く

実数

文法

```
(double)rand() / (RAND_MAX + 1)
```

- rand()で生成される乱数の値の範囲を「0以上n未満」にするには，次のように書く

整数

文法

```
rand() % n
```

- rand()で生成される乱数の値の範囲を「1以上n以下」にするには，次のように書く

整数

文法

```
rand() % n + 1
```

# 課題

# レポートの作成

① レポートの冒頭に以下を適切なレイアウトで書く
- 「情報処理実習第14回課題レポート」というタイトル
- 学生番号
- 氏名

② 課題ごとに以下を載せる
- 作成したプログラムのソースコード
- 作成したプログラムの実行結果を示すコマンドプロンプトのスクリーンショット

③ 完成したレポートをCoursePower上で提出する