画像処理
(4J)

第01回
～導入、Ｃ言語の復習（１）～

導入

授業の進め方について

●シラバス参照

- ※修正中　・・・　なぜか開講時期が前期になってる

●授業全体の 1/3 は、C言語の復習に充てます

●残りの 2/3 で、C言語を用いて実際に画像処理を実装していきます

- OpenCVは、画像データの入出力と、画像データの表示のみに利用
  (高機能な関数等は一切使わない)
- ピクセル単位での基本演算の組み合わせで画像処理を実装
- 車輪の再発明をすることで、画像処理の中身を学ぶ

●授業資料は、Moodle にアップロードします。

●課題提出も Moodle から。【期限厳守!】

評価について

●試験は、期末試験のみ実施　（中間試験は無し）

●学修単位なので、自己学修のエビデンスが必要　＝　課題を課します

- 課題の未提出が 1/4 を超えた場合は、60点未満に

●総合成績は、課題50% ＋ 試験50% で評価します。

【課題の重みが大きいので注意！】

- “デジタル画像の扱いや、基本的な画像処理アルゴリズムの理解の程度と、それをC言語で記述できるスキルを評価する。”

学修単位について確認

- 授業時間数(予定)：　全15回

- 1:2型（講義科目）の学修単位科目です。
  - 1[単位]=45[単位時間]の学習。
  - この授業は2単位なので、45×2=90[単位時間]の学修(授業+自己学習)が必要。
  - 1:2型では、
    「授業15単位時間+自己学習30単位時間」で1単位
    2単位だと　「授業時間30単位時間+自己学習60単位時間」

要は、1回の授業当たり、 **180分以上**(授業時間の倍)
自己学習しなさいということ。

C言語の復習

いろいろなプログラミング言語

●(機械語)
●(アセンブリ)

●C
●C++

コンパイラ型

●Java
●C#

中間コード型

●Python
●JavaScript
●PHP
●Ruby
●Perl
・・・

インタプリタ型

「高級言語(高水準言語)」
・・・ ”すごい言語”ではない

- 記述の抽象度が高い言語のこと
- (あまり)プロセッサに依存しない
- メモリ制御等のハードウェア寄りのことを意識しなくて良い
- 人にとって、比較的分かりやすい記述

C言語は根強い人気・・・

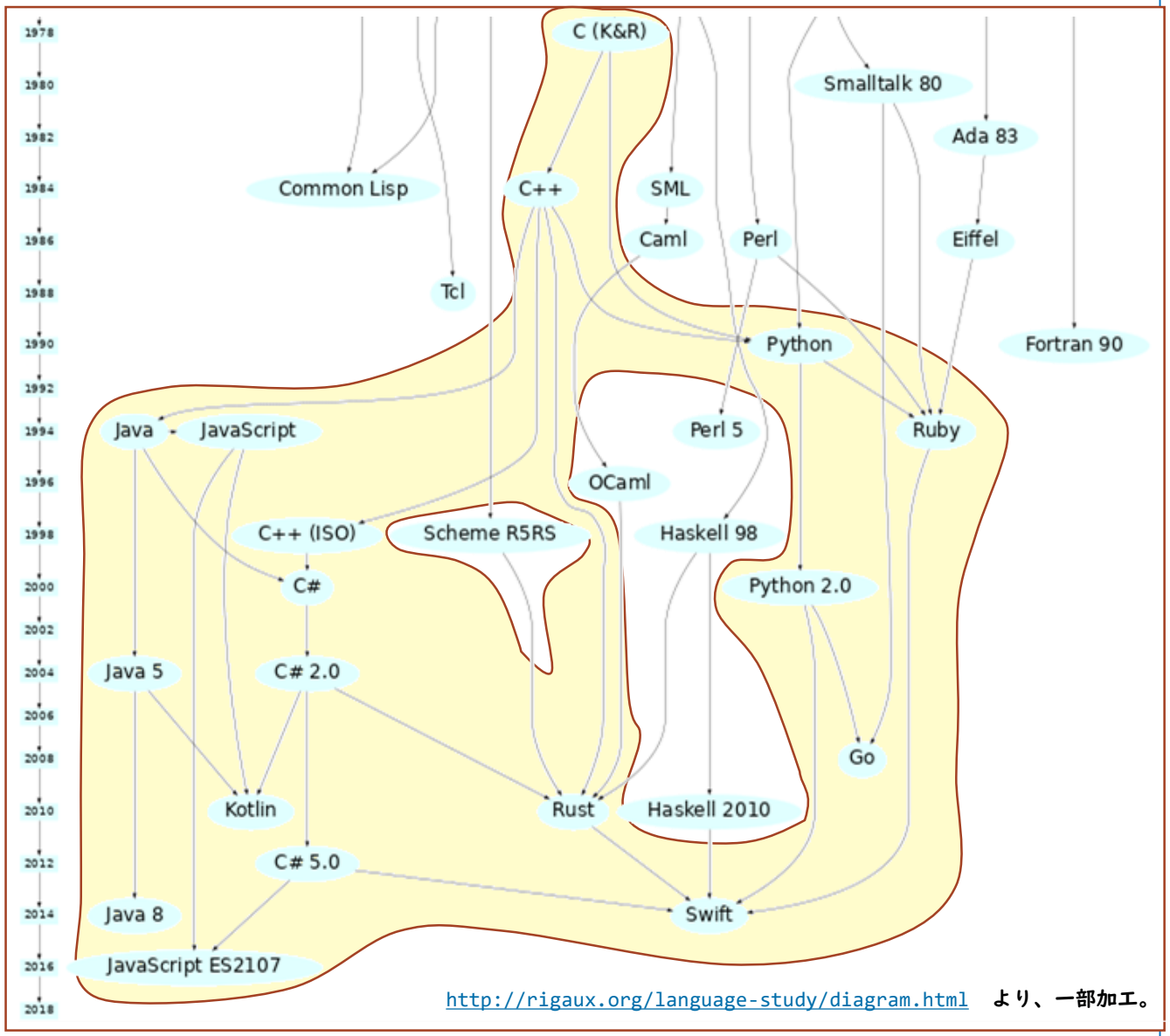
●低級言語寄りの記述ができる

- ポインタ・・・　ある意味諸刃の剣
  だが、よく理解して使えば
  高パフォーマンスかつ
  高効率のコーディングが可能。
  (一方で、ポインタ絡みのバグは、
  システムに致命的な影響を与える場合も…)
- 組み込み用途などでは有利
  - 限られた処理能力
  - 小さなメインメモリ

●様々な言語が影響を受けている

- C++ や C# は、構文上の共通点が多い
- JAVA や JavaScript も同様
- Python も強く影響を受けている

diagram & history of programming languages



http://rigaux.org/language-study/diagram.html　より、一部加工。

それでは実際に・・・

●Windowsの場合

- VisualStudio Community 2019 の c/c++言語環境
  （すでに環境がある人は、2017/2015/2013 どれでも構わない）
- gcc (MinGW)

●Macの場合

- gcc (Xcode の Command Line Tool?)

●「C言語の復習」後、画像処理の内容に入りますが、
その際は VisualStudio 2019 + OpenCV 2.4 の環境を使う
予定です。環境のセットアップ等については後ほど・・・。

●「C言語の復習」では、Web上のC言語実行環境を使って、
環境構築の手間なく、お手軽に試してみることにします。

web上のC言語実行環境

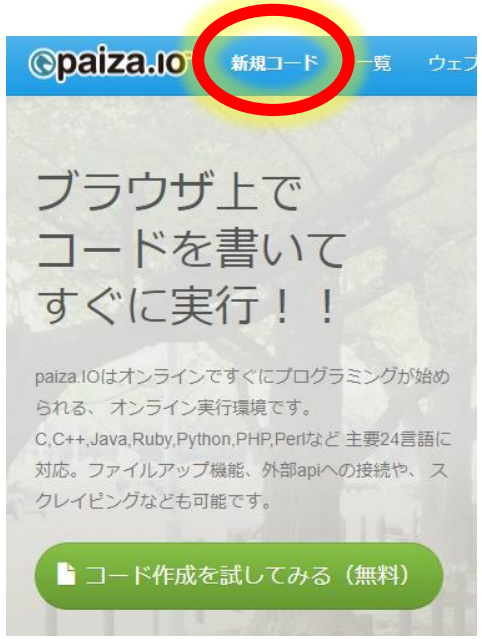
- paiza.io
  - https://paiza.io
  - 複数ファイルのコンパイルもできる(勝手にされる)　・・・オプション指定不可、リンカ見えない
  - ファイル読み込みが出来る
  - 日本語
- Wandbox を使う？
  - https://wandbox.org/
  - 結構自由度高そう
  - コンパイラのオプション指定もできる
  - 複数ファイルのコンパイルもOK
- coding ground を使う？
  - https://www.tutorialspoint.com/codingground.htm
  - コンパイラオプションを指定可能　・・・　[-v]オプションで、コード生成の詳細を見れる

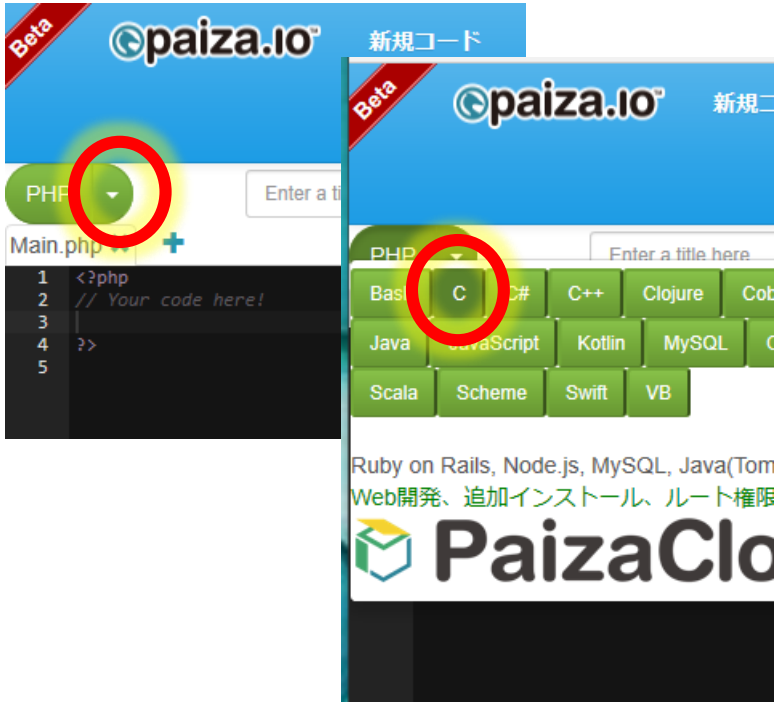
Web上のC言語実行環境 paiza.io で試す

https://paiza.io/ja/projects/new?language=c

または、「https://paiza.io」にアクセス

➡ 新規コード ➡ C言語を選択





本日の演習の目標

●その他　printf(), #include <>

●C言語の基本的な書き方がわかる

- 「;」　・・・ 行は「;」で終わる。
- 「{}」　・・・ ブロックは「{}」で囲う。
- 「/* 」～「 */」　・・・ コメント(囲んだ範囲)
- 「//」　・・・ 行末までのコメント
- main関数　・・・ エントリポイント

●各種演算子

- 「=」　・・・ 代入演算子
- 「+」「-」「*」「/」「%」　・・・ 算術演算子
- 「==」「!=」「<」「>」「<=」「>=」　・・・ 比較演算子

●型と定義

- 「int」「double」　変数の型と定義
- 「"」～「"」　文字列の定義

●制御文 (※典型的な利用例)

- if文

```
if (num == 100)  {
    // trueの場合の処理
}
else {
    // falseの場合の処理
}
```

- for文

```
for(int i=0; i<10; n++) {
    // 繰り返す処理
}
```

- White文

```
while(i < 10) {
    // 繰り返す処理
}
```

演習課題1: 以下の出力を得るプログラムを作成する

●文字列

Ichinoseki
KOSEN

を表示

```
Ichinoseki
KOSEN
```

演習課題1 ～解答例A～


```
#include <stdio.h>


int main(void) {
      printf("Ichinoseki¥nKOSEN¥n");
      return 0;
}
```

#で始まるもの ➡「プリプロセッサ命令」
・・・コンパイル前の、前処理を行う。
※行末に「;」を付けない!!

#include はファイルを取り込む命令。
(指定ファイルの内容を、すべてこの位置に貼り付けるのと同じ意味になる)

<> で囲うと、標準ディレクトリから、
"" で囲うと、カレントディレクトリ(無ければ標準ディレクトリ)のファイルが取り込まれる。

一般的には ヘッダファイル を取り込むのに使われる。(テキストファイルなら何でも良い)

int main(void) { … }
・・・ 特殊な関数(エントリポイント)。まずここから実行される。

int は「返り値の型」。
➡ なので、関数内で「return 0;」としてint型の値を返す。

()内は「引数の型」 ・・・ ここでは、何もなし、の意味。

"" で囲った内容は「文字列リテラル」

¥n は「エスケープシーケンス」の一つで、「改行」を意味する。

printf() は、stdio.h 内で宣言されている標準ライブラリ関数の一つ。
・・・ 書式を指定した、標準出力への出力が可能。

演習課題1 ～解答例B,C,D～


```c
// 解答例B
#include <stdio.h>

int main(void) {
    puts("Ichinoseki");
    puts("KOSEN");
    return 0;
}
```

他にもぜひ、
考えて
みて
ください。

```c
// 解答例C
#include <stdio.h>

int main(void) {
    char str1[] = "Ichinoseki";
    char str2[] = "KOSEN";

    puts(str1);
    puts(str2);
    return 0;
}
```

```c
// 解答例D
#include <stdio.h>

int main(void) {
    char str1[] = "Ichinoseki";
    char str2[] = "KOSEN";

    printf("%s¥n%s¥n", str1, str2);
    return 0;
}
```

```c
// 解答例D
#include <stdio.h>

int main(void) {
    char str[] = {73, 99, 104, 105, 110, 111, 115, 101, 107, 10, 75, 79, 83, 69, 78, 10, 0};
    printf("%s", str);

    return 0;
}
```

演習課題2: 以下の出力を得るプログラムを作成する

●「0～9」の数字を表示(間には半角スペースを入れる)

```
0 1 2 3 4 5 6 7 8 9
```

ヒント1


```
// この場合、初期化されるのは c のみなので注意
#include <stdio.h>
int main(void){
    int a, b, c = 100;
    printf("a=%d¥nb=%d¥nc=%d¥n", a, b, c);
    return 0;
}
```

●変数の宣言

- 変数は、使用する前に、”宣言”が必要
- 整数型「int」

```
int abc, def;
int xyz = 10;
```

··· 整数型の変数「abc」と「def」を宣言
··· 宣言と同時に値「10」で初期化

●C言語では、宣言しただけ(初期化しない状態)では、**値が「不定」**

··· 必ず値を代入してから使う!

```
// 未初期化だとどうなるか‥？
#include <stdio.h>
int main(void){
    int a, b;
    printf("a=%d¥nb=%d¥n", a, b);
    return 0;
}
```

```
a=2124203496
b=2124203512
```

```
a=-58221384
b=-58221368
```

```
a=230176824
b=230176840
```

複数回実行してみてください。
実行のたびに値が変わったりする様子も確認できるはず。

ヒント2

●for文
(p.142)

●典型的な使い方

```
int i;
for(i=0; i<5; i++) {
      // 繰り返す処理
      // この場合、 i は {0,1,2,3,4} の値になる(5にはならない!!)
}
// for文を抜けたあと、i は 5 の値になっている。
```

```
for(int i=0; i<5; n++) {
      // この場合、i が使えるのは{}内に限られる
      // 詳細は「スコープ」のところで解説します
}
// ここでは i は未宣言となるので利用できない
```

for (①最初に1回だけ実行; ②継続条件(trueの間繰り返す); ③最後に毎回実行) {}

↗比較演算子を活用 { ==, !=, <, >, <=, >= }

※繰り返しの文が1行の場合は {} は不要。

●printf()の書式指定(p.320) ･･･ %d=整数, %s=文字列

- 整数: int i=100; ➡ printf("%d", i); ･･･「100」が出力される
- 文字列: char str[]="ABCD" ➡ printf("%s", str); ･･･「ABCD」が出力される

このとき、 printf("%s ==> %d", str, i); ･･･「ABC ==> 100」が出力される。

for文の動作の確認

●動作と出力を考えてから、実際に実行結果を確認しよう！
(わからなければ、実行結果から動作を追ってみましょう)

```
// Case: A
#include <stdio.h>
int main(void){
   int i = 10;
   for (i=0; i<6; i++) {
      printf("i=%d¥n", i);
   }
   printf("for後の i=%d", i);
   return 0;
}
```

```
// Case: B
#include <stdio.h>
int main(void){
   int i = 10;
   for (i=10; i>4; i--) {
      printf("i=%d¥n", i);
   }
   printf("for後の i=%d", i);
   return 0;
}
```

```
// Case: C
#include <stdio.h>
int main(void){
   int i = 10;
   for ( ; i==10; i++) {
      printf("i=%d¥n", i);
   }
   printf("for後の i=%d", i);
   return 0;
}
```

➡ 簡単すぎて、ヒマな人は次ページも！

for文の動作の確認

●簡単すぎて、ヒマな人だけ、これらについても考えてみてください。

```
// Case: D
#include <stdio.h>
int main(void){
   int i=10;
   for (i=0; i<6; i--) {
      printf("i=%d ", i);
      i+=3;
      printf(" --> %d¥n", i);
   }
   printf("for後の i=%d", i);
   return 0;
}
```

```
// Case: E
#include <stdio.h>
int main(void){
   int i=10;
   for (i=0; i<6; printf("i=%d¥n", i++)) {
      ;
   }
   printf("for後の i=%d", i);
   return 0;
}
```

```
// Case: F
#include <stdio.h>
int main(void){
   for (int i=0; i-6; printf("i=%d¥n", i)) {
      i++;
   }
   return 0;
}
```

変数の有効範囲 ”スコープ”の話

- スコープが違うと、変数にアクセスできない
- スコープが違う場所なら、同名の"別の変数"も定義可能

```c
// Example: A-1
#include <stdio.h>

int main(void){
    int a = 999;
    printf("main: a = %d¥n", a);

    for (a = 0; a < 5; a++){
        printf("for内: a = %d¥n", a);
    }

    printf("main: a = %d¥n", a);
    return 0;
}
```

```c
// Example: A-2
#include <stdio.h>

int main(void){
    int a = 999;
    printf("main: a = %d¥n", a);

    for (int a = 0; a < 5; a++){
        printf("for内: a = %d¥n", a);
    }

    printf("main: a = %d¥n", a);
    return 0;
}
```

変数の有効範囲 ”スコープ”の話

- スコープが違うと、変数にアクセスできない
- スコープが違う場所なら、同名の"別の変数"も定義可能

```c
// Example: B-1
#include <stdio.h>

void sub(void) {
    int a = 10;  // ローカル変数
    printf("sub: a=%d¥n", a);
    a = 999;
}

int main(void){
    int a = 10;  // ローカル変数
    sub();
    printf("main: a=%d¥n", a);
    return 0;
}
```

```c
// Example: B-2
#include <stdio.h>

int a = 10;  // グローバル変数

void sub(void) {
    printf("sub: a=%d¥n", a);
    a = 999;
}

int main(void){
    sub();
    printf("main: a=%d¥n", a);
    return 0;
}
```

変数の有効範囲 ”スコープ”の話

- {}で囲うことで、スコープを限定することもできる
  (読みにくくなるのでおすすめしないが、前述のようにfor文などではよく使われている)

```
// Case: C-1
#include <stdio.h>

int main(void){
    int a = 999;
    printf("1: %d¥n", a);
    {
        int a = 333;
        printf("2: %d¥n", a);
        {
            int a = 111;
            printf("3: %d¥n", a);
            {
                int a = 0;
                printf("4: %d¥n", a);
            }
            printf("5: %d¥n", a);
        }
        printf("6: %d¥n", a);
    }
    printf("7: %d¥n", a);

    return 0;
}
```

```
// Case: C-2 (※コンパイルエラーになる)
#include <stdio.h>

int main(void){
    int a = 999;
    printf("1: %d¥n", a);

    int a = 333;
    printf("2: %d¥n", a);

    return 0;
}
```

```
Main.c:8:6: error: redefinition of 'a'
        int a = 333;
            ^
Main.c:5:6: note: previous definition is here
        int a = 999;
            ^
1 error generated.
```

同一の
スコープ内で、
同名の変数を
宣言することは
できない。
<コンパイル
エラーになる>

(区別がつかない
ので当然)

演習課題3: 以下の出力を得るプログラムを作成する

●「0～99」の数字を、10数字毎に改行して表示

```
 0  1  2  3  4  5  6  7  8  9
10 11 12 13 14 15 16 17 18 19
20 21 22 23 24 25 26 27 28 29
30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69
70 71 72 73 74 75 76 77 78 79
80 81 82 83 84 85 86 87 88 89
90 91 92 93 94 95 96 97 98 99
```

ヒント:
書式指定子 %d を %nd (nは正数)とすると、
数値を必ずn桁で出力します。

例:

```
printf("[%4d][%3d][%2d][%d]¥n"
                          ,10,10,10,10);
```

⬇実行結果

```
[  10][ 10][10][10]
```

●演習課題３　は、色々な考え方ができると思います。

●1つ出来た人は、いろいろな実装方法を試してみましょう。
（できるだけたくさん！）

- forの二重ループで
- 10での剰余が9のときに改行
- カウンタ変数を用意して、9回足したら改行してカウンタをリセット

演習課題④: 以下の出力を得るプログラムを作成する

●演習課題③のプログラムを修正して、

A) 20文字毎に改行、とする。

B)「0～199」までの数字、とする。

C)「0～n」までの数字を「m文字毎に改行」とする。
※nとmは変数で定義する。nは最大999とする。
(例えば n=149; m=10;)

・・・複数の実装を試した人は、どの実装方法が変更に強かった？

出力結果の見本(演習課題④)

A)

```
 0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39
40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59
60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79
80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99
```

B)

```
  0   1   2   3   4   5   6   7   8   9
 10  11  12  13  14  15  16  17  18  19
 20  21  22  23  24  25  26  27  28  29
 30  31  32  33  34  35  36  37  38  39
 40  41  42  43  44  45  46  47  48  49
 50  51  52  53  54  55  56  57  58  59
 60  61  62  63  64  65  66  67  68  69
 70  71  72  73  74  75  76  77  78  79
 80  81  82  83  84  85  86  87  88  89
 90  91  92  93  94  95  96  97  98  99
100 101 102 103 104 105 106 107 108 109
110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129
130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169
170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189
190 191 192 193 194 195 196 197 198 199
```

C)

n=149; m=10; の例

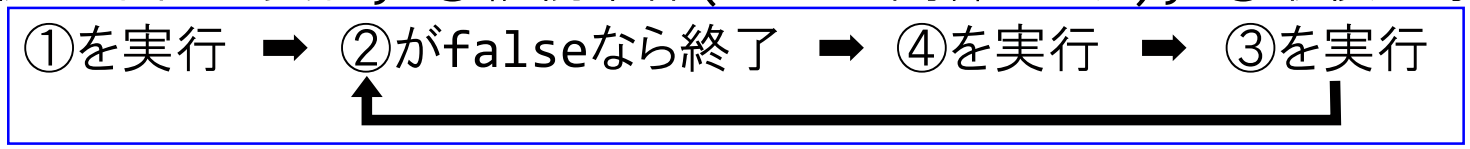
```
  0   1   2   3   4   5   6   7   8   9
 10  11  12  13  14  15  16  17  18  19
 20  21  22  23  24  25  26  27  28  29
 30  31  32  33  34  35  36  37  38  39
 40  41  42  43  44  45  46  47  48  49
 50  51  52  53  54  55  56  57  58  59
 60  61  62  63  64  65  66  67  68  69
 70  71  72  73  74  75  76  77  78  79
 80  81  82  83  84  85  86  87  88  89
 90  91  92  93  94  95  96  97  98  99
100 101 102 103 104 105 106 107 108 109
110 111 112 113 114 115 116 117 118 119
120 121 122 123 124 125 126 127 128 129
130 131 132 133 134 135 136 137 138 139
140 141 142 143 144 145 146 147 148 149
```

n=100; m=8; の例

```
  0   1   2   3   4   5   6   7
  8   9  10  11  12  13  14  15
 16  17  18  19  20  21  22  23
 24  25  26  27  28  29  30  31
 32  33  34  35  36  37  38  39
 40  41  42  43  44  45  46  47
 48  49  50  51  52  53  54  55
 56  57  58  59  60  61  62  63
 64  65  66  67  68  69  70  71
 72  73  74  75  76  77  78  79
 80  81  82  83  84  85  86  87
 88  89  90  91  92  93  94  95
 96  97  98  99 100
```

まとめ

- 標準ライブラリヘッダの取り込みと、main()関数の書き方
  - #include <stdio.h>
  - int main(void){ }
- 変数宣言など、基本的なCプログラムの書き方
  - int a=10;　　"文字列リテラル"　　基本的な演算子　　大文字と小文字の区別等
- printf()関数の基本的な使い方
  - 書式指定文字 %d, %s
- for文の使い方
  - for (①最初に1回だけ実行; ②継続条件(trueの間繰り返す); ③最後に毎回実行) { ④ }



- 変数のスコープ
  - ローカル変数(局所変数) ⇔ グローバル変数(大域変数)