プログラムとアルゴリズム

担当：新田

コンピュータ言語

人間がコンピュータに処理を指示するための言語

- **プログラミング言語**
  コンピュータに対して命令を与えるための言語
  アプリケーションやシステムの開発に使用される
  例）Java、Python、C言語、JavaScript
- マークアップ言語
  文書の構造や内容を記述するための言語
  タグ<>を使用して文書の構造をマークする
  例）HTML、XML
- スタイルシート言語
  文書の表示やレイアウトを定義するための言語
  例）CSS
- クエリ言語
  データベースを操作するための言語
  例）SQL（MySQL、PostgreSQL、Oracle）

コンピュータは0と1の世界

機械語（マシン語）

01101001
01011010
11010110
10100101

↑16進数に変換されたもの

- コンピュータ自身が命令を直接理解し、実行することができる
- 0と1の二進数で表現される
- コンピュータ内では0と1という情報をパルスとして回路内で伝送し、演算処理を行う
- 記憶装置では記憶素子の状態が0と1に対応してデータを記録する
- 人間が理解するのは至難の業！

低水準言語

アセンブリ

```
LD   A,  B        ロード命令
ADD  A,  B        加算命令
CLR  B            クリア命令
```

- 機械語を人間が扱いやすい形式で記述したもの
- 機械語とアセンブリ言語は1対1で対応
- CPUやメモリを操作ができるため、実行速度が速い
- 便利な制御構文や処理がアセンブリ言語にはないため
  すべて自分で記述する必要がある
- ハードウェアやメモリー管理の知識が不可欠
- 移植性が低く、コーディングが複雑で生産性は低い

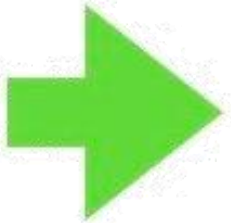
高水準言語

- 人間が理解しやすいように作られた言語
- 低水準言語よりも可読性に優れ、短時間でコードを書くことができる
- ハードウェアの深い知識が必要ない
- 実行速度は低水準言語よりも遅い

人間が読める
ソースコード

```
int
    int a,b;
    top = 0;
    char s[100];
    while(scanf("%s",s) != EOF){
        if(s[0] == '+'){
            int last = pop();
            int second_last = pop();
            push(last+second_last);
        }else if (s[0] == '-'){
            int last = pop();
            int second_last = pop();
            push(second_last-last);
        }else if (s[0] == '*'){
            int last = pop();
            int second_last = pop();
            push(last*second_last);
        }else if (s[0] == 'e'){
            printf("%d\n",pop());
            return 0;
        }else{
            push(atoi(s));
        }
    }
    return 0;
}
```

変換する



機械語

高水準言語のプログラム（ソースコード）を機械語に変換することを
コンパイル、この変換作業を行うソフトウェアを**コンパイラ**

コンパイル型言語

- プログラムの実行前にソースコードをまとめて機械語に翻訳する。
- インタプリタ型より高速に処理が可能。
- 実行するたびにすべてのコードをコンパイルする必要があるため、テスト作業に時間がかかる。

例）Java、C/C++、C#、Go、Rust

インタプリタ型言語

- プログラムの実行時にコードを1行ずつ機械語に翻訳する。
- 1行ずつ実行できるため、開発やテスト作業が迅速にできる。
- 実行速度やメモリ使用量ではコンパイラ型に劣る。

例）Python、JavaScript、Ruby、PHP

アルゴリズム

ある特定の問題を解く手順を、単純な計算や操作の組み合わせとして明確に定義したもの

- 探索アルゴリズム
- ソートアルゴリズム
- 暗号化アルゴリズム
- 機械学習アルゴリズム　など

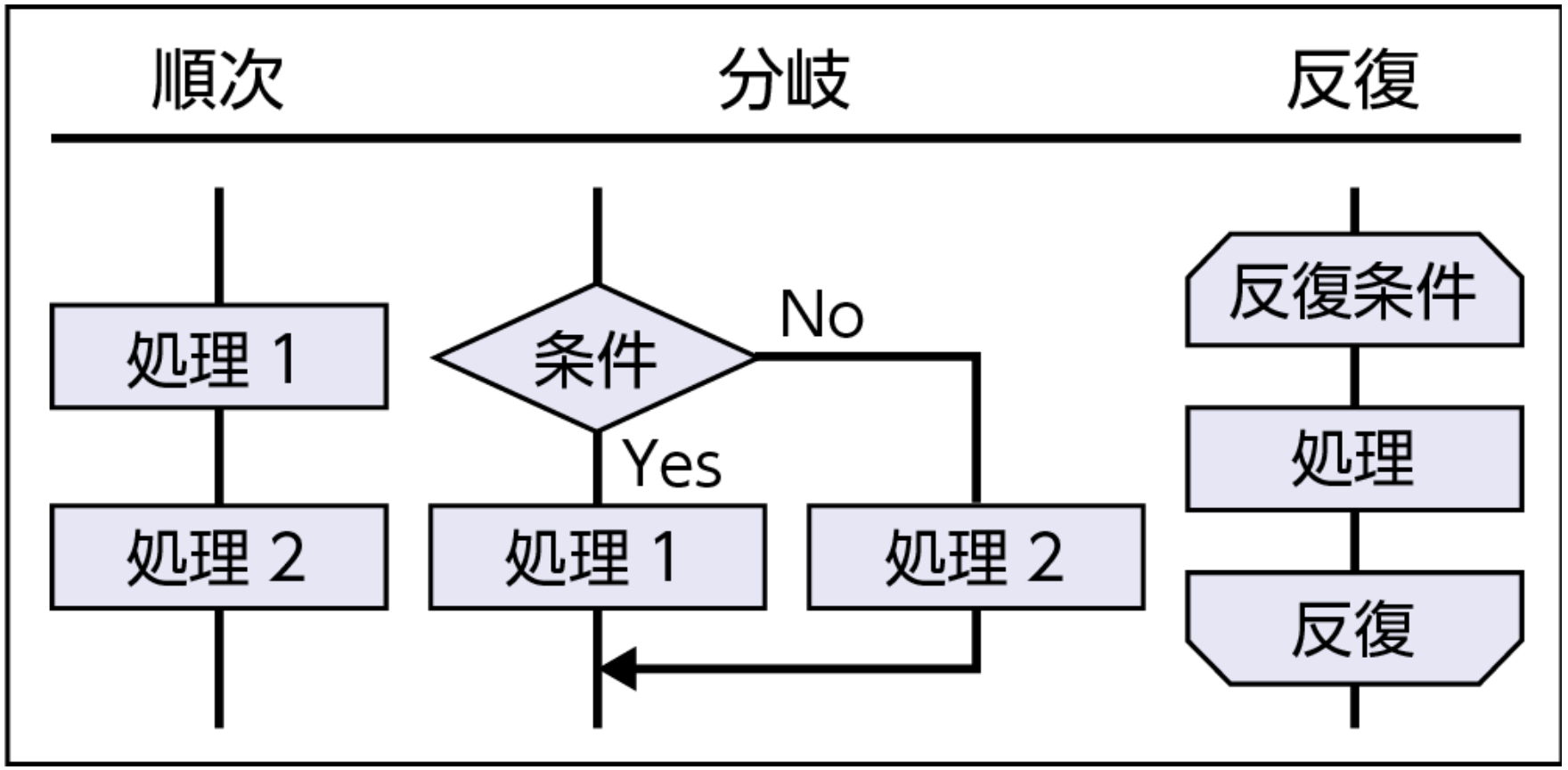
この手順を

人が理解しやすいように図にしたものが**フローチャート（流れ図）**

コンピューターに行わせるために記述したものが**プログラム**

プログラミング言語で記述されたプログラムを**ソースコード**

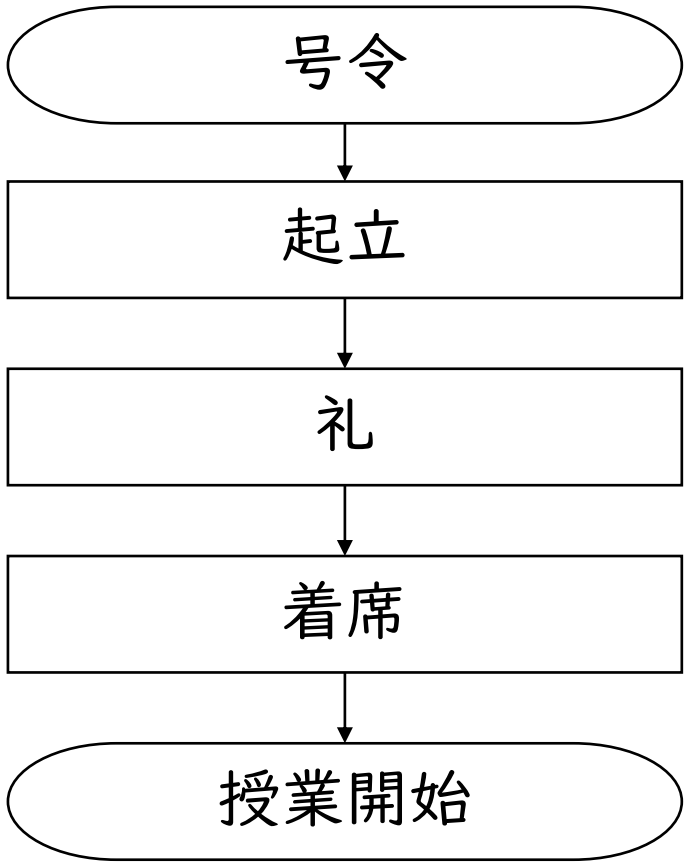
アルゴリズムの基本構造



複雑なアルゴリズムも、基本構造の組み合わせから成る。

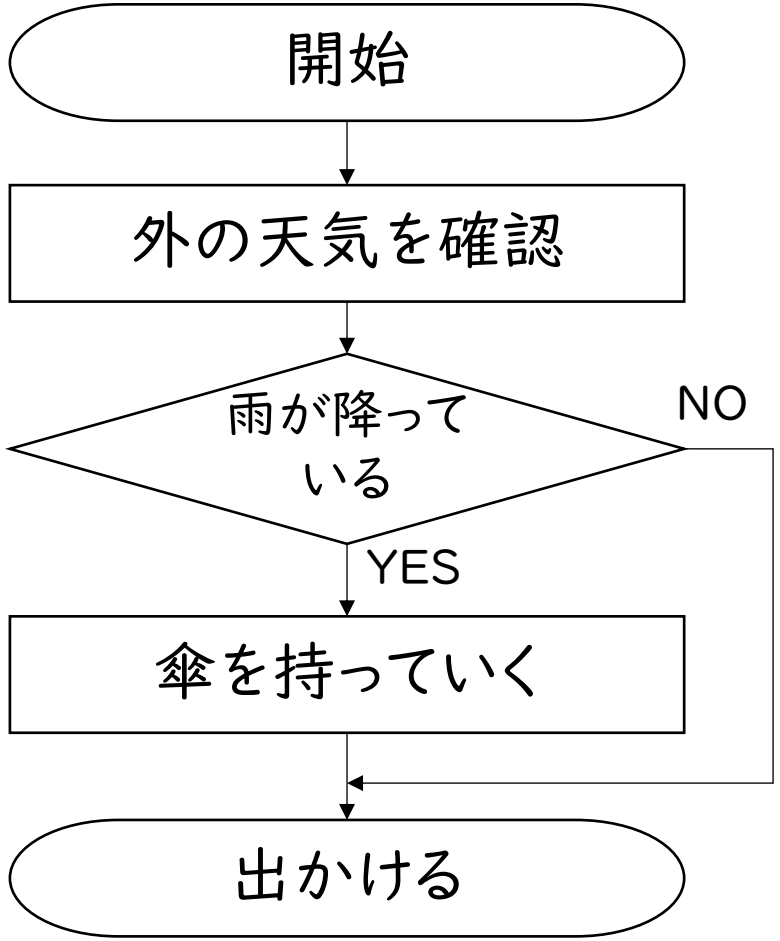
順次構造

ひとつの処理が終わったら次の処理に進む



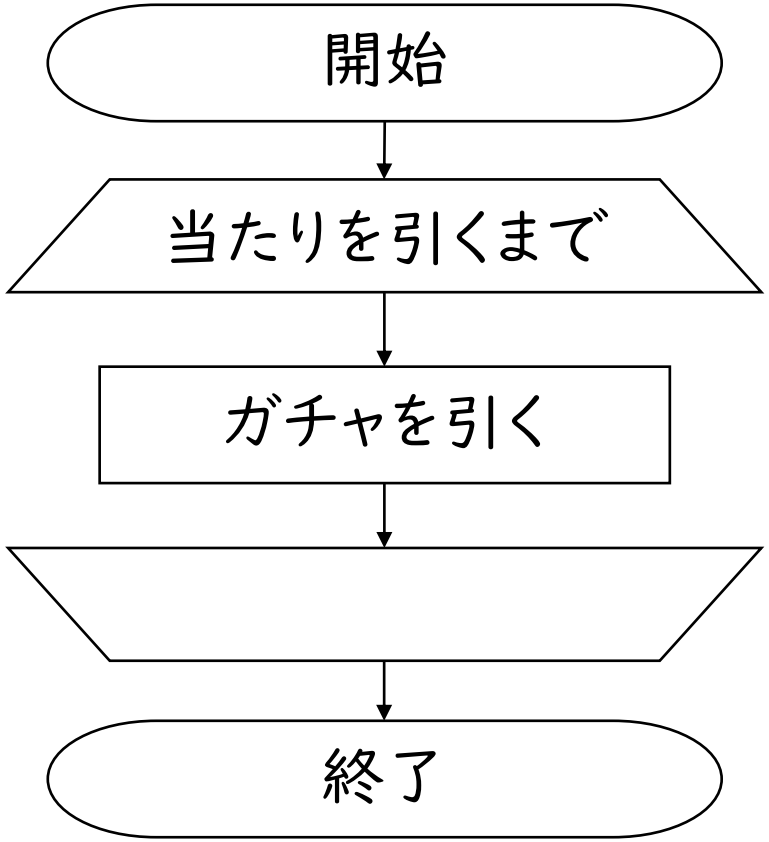
分岐構造

条件に基づいて実行する内容を変える



反復構造

一定の条件を満たすまで処理を繰り返す



疑似言語

- 擬似的なプログラミング言語で、一般的に使われる記述法を交えつつ、アルゴリズムの理解を助けるために用いられる。
- ITパスポート・基本情報技術者試験独自のものがある。
- 仕様書が添付されている。 https://www.ipa.go.jp/shiken/syllabus/doe3um0000002djj-att/shiken_yougo_ver5_1.pdf

実際のプログラミング言語↓

```
#include <stdio.h>
int main(void){
   printf("Hello, World!¥n");
   return 0;
}
```

```
class HelloWorldApp
{
    static void Main() {
        System.Console.WriteLine("Hello, World!");
     }
}
```

```
print("Hello, World!")
```

```
puts "Hello, World!"
```

```
int main(int argc, const char * argv[]) {
    @autoreleasepool {
        NSLog(@"Hello, World!");
    }
    return 0;
}
```

```
public class HelloWorld
{
    public static void main(String[] args){
        System.out.println("Hello, World!");
    }
}
```

```
document.writeln('Hello, World!');
```

```
<?php
 echo "Hello, World!";
?>
```

〔擬似言語の記述形式〕

記述形式	説明
○*手続名又は関数名*	手続又は関数を宣言する。
型名: *変数名*	変数を宣言する。
/* *注釈* */	注釈を記述する。
// *注釈*	
変数名 ← *式*	変数に*式*の値を代入する。
手続名又は関数名(*引数*, …)	手続又は関数を呼び出し，*引数*を受け渡す。

if（*条件式 1*）
　処理 1
elseif（*条件式 2*）
　処理 2
elseif（*条件式 n*）
　処理 n
else
　処理 n ＋ 1
endif
	選択処理を示す。
	*条件式*を上から評価し，最初に真になった*条件式*に対応する*処理*を実行する。以降の*条件式*は評価せず，対応する*処理*も実行しない。どの*条件式*も真にならないときは，*処理 n ＋ 1*を実行する。
	各*処理*は，0 以上の文の集まりである。
	elseif と*処理*の組みは，複数記述することがあり，省略することもある。
	else と*処理 n ＋ 1*の組みは一つだけ記述し，省略することもある。

while（*条件式*） 　*処理* endwhile	前判定繰返し処理を示す。 *条件式*が真の間，*処理*を繰返し実行する。 *処理*は，0 以上の文の集まりである。
do 　*処理* while（*条件式*）	後判定繰返し処理を示す。 *処理*を実行し，*条件式*が真の間，*処理*を繰返し実行する。 *処理*は，0 以上の文の集まりである。
for（*制御記述*） 　*処理* endfor	繰返し処理を示す。 *制御記述*の内容に基づいて，*処理*を繰返し実行する。 *処理*は，0 以上の文の集まりである。

〔演算子と優先順位〕

演算子の種類		演算子	優先度
式		()	高
単項演算子		not ＋ －	↕
二項演算子	乗除	mod × ÷	↕
	加減	＋ －	↕
	関係	≠ ≦ ≧ ＜ ＝ ＞	↕
	論理積	and	↕
	論理和	or	低

注記　演算子 mod は，剰余算を表す。

〔論理型の定数〕

true，false

〔配列〕

一次元配列において“{”は配列の内容の始まりを，“}”は配列の内容の終わりを表し，配列の要素は，“[”と“]”の間にアクセス対象要素の要素番号を指定することでアクセスする。

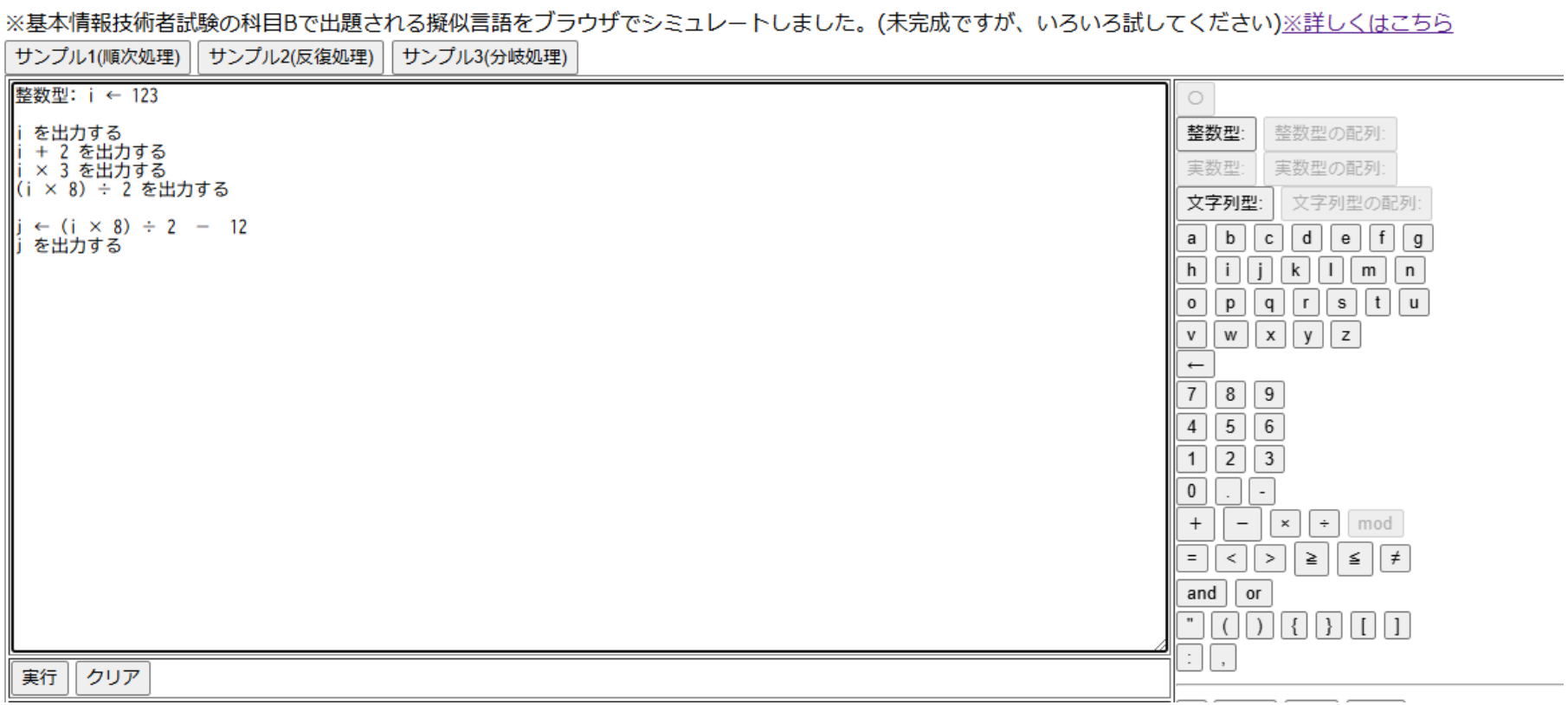
例　要素番号が 1 から始まる配列 exampleArray の要素が{11，12，13，14，15}のとき，要素番号 4 の要素の値（14）は exampleArray[4]でアクセスできる。

二次元配列において，内側の“{”と“}”に囲まれた部分は，1 行分の内容を表し，要素番号は，行番号，列番号の順に“,”で区切って指定する。

例　要素番号が 1 から始まる二次元配列 exampleArray の要素が{{11，12，13，14，15}，{21，22，23，24，25}}のとき，2 行目 5 列目の要素の値（25）は，exampleArray[2，5]でアクセスできる。

基本情報技術者試験(科目B)の擬似言語シミュレータ(beta版)

https://gijigengo.b-rain.jp/



一部機能は未実装だが、試験とほぼ同じ書き方で処理を再現できる。

基礎知識

変数

- 数字や単語などの**データを入れておく箱**。
- 箱には名前を付ける（ルールあり）。

"ABC"

"あいうえお"

12345

1.4142

False

命令文

配列[ ]

箱に入れるイメージ

x ← 12

xには12が格納される

型

変数に入れることのできるデータの種類を指定する

- 整数型（Integer等）　1，2，10，-196，123456
- 実数型（Double等）　3.14，44.5，100.00
- 文字列型（String）　“ABC”，“HELLO”，“あいうえお”
- 論理型(Boolean）　true，false

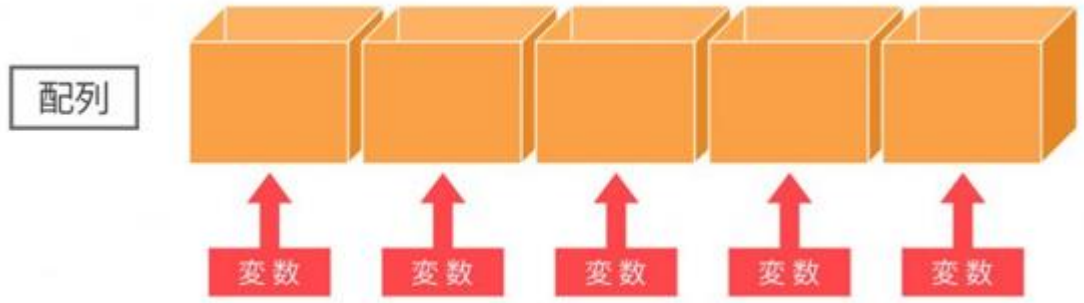
配列

データをまとめて扱うデータ構造の1つ

同じデータ型の複数の値を1つの変数にまとめることができる

目的のデータが何番目にあるかを指定することで、データを取り出せる

インデックス



※一般的に、配列の先頭は0

nameList = {‘田中’, ‘佐藤’, ‘鈴木’}

nameList[0] ・・・‘田中’を指定

プログラミングにおける関数

与えられた値（引数）をもとに、定められた独自の処理を実行し、
その結果を返す命令のこと

定められた処理を繰り返し利用することができる

与えられた値
（引数）

卵かけご飯を
作る関数

- 卵を1個割る
- ご飯の上にのせる
- 醤油をかける

処理の
結果

疑似言語の流れ図

//や /*　*/はコメントアウト
いわゆる注釈　処理には影響しない

文字列型: familyName, firstName　　//変数を宣言
整数型: age

familyName ← “田中”
firstName ← “太郎”
age ← 20　　//変数に値を代入

familyName を出力する
firstName を出力する
age ← age + 1
age を出力する

出力結果：
田中
太郎
21

疑似言語の関数

○戻り値の型： 関数○○○（引数の型：変数, ・・・・）

関数名

return △△

戻り値

※引数は複数ある場合、1つもない場合もある

○整数型：関数calcTax (整数型: yen)　　//関数の宣言

整数型：total

total ← yen × 1.1

return total　　//戻り値

関数の使い方

整数型：fee

fee ← calcTax（1000）　　出力結果：1100

fee を出力する

頻出パターン

○○のとき～、それ以外のとき～

```
if (　条件　)
　　処理
else
　　処理
endif
```

条件を満たすときと満たさないときで
処理を分岐させるアルゴリズムは多い！

VBA

```
If　条件　　Then
　　処理
Else
　　処理
End If
```

Java

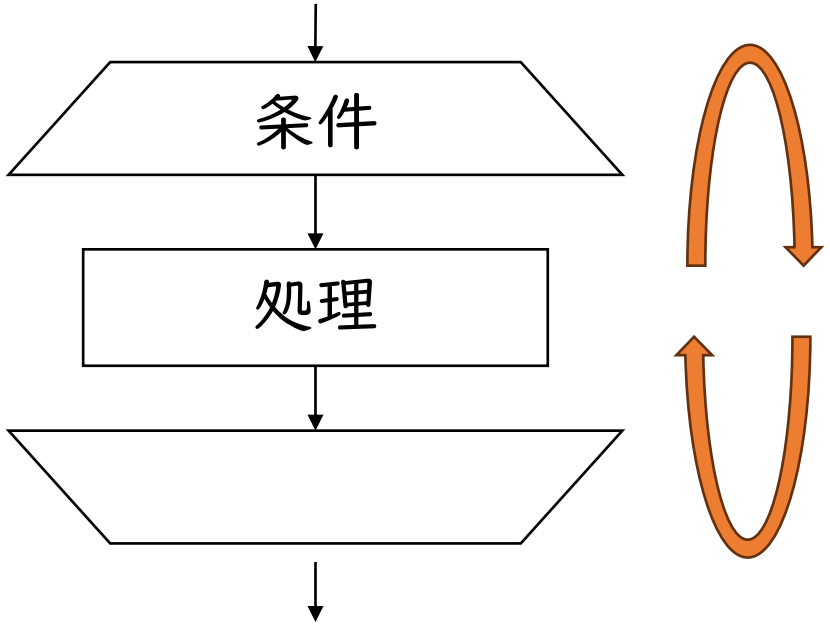
```
If ( 条件 ){
　　処理
}else{
　　処理
}
```

Python

```
if　条件:
　　処理
else:
　　処理
```

○○のときはずっと繰り返す

while (条件)
　　処理
endwhile



※繰り返し処理を終了できない（条件を満たし続けてしまう）場合、無限ループに陥り、フリーズや暴走を引き起こす。

VBA

```
Do while 条件
　　処理
Loop
```

Java

```
while( 条件 ){
　　処理
}
```

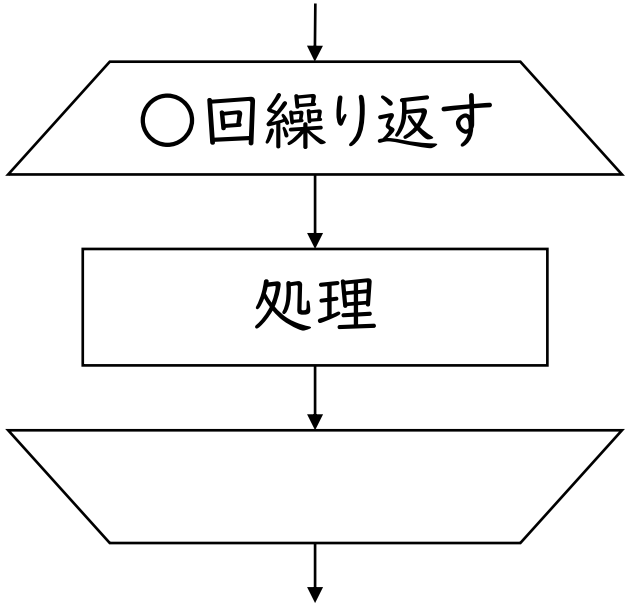
Python

```
while 条件:
　　処理
```

○○回繰り返しする

for (i を 1から ○○まで1ずつ増やす)
　　処理
endfor

※このときの i を**カウンタ**と呼ぶ



VBA
```
For i = 1 To ○○
　　処理
Next i
```

Java
```
for(int  i = 1; i <=○○; i++){
　　処理
}
```

Python
```
for  i range in (1, ○○+1):
　　処理
```

○と△を入れ替える

整数型: buf, a, b

a ← “りんご”

b ← “みかん”

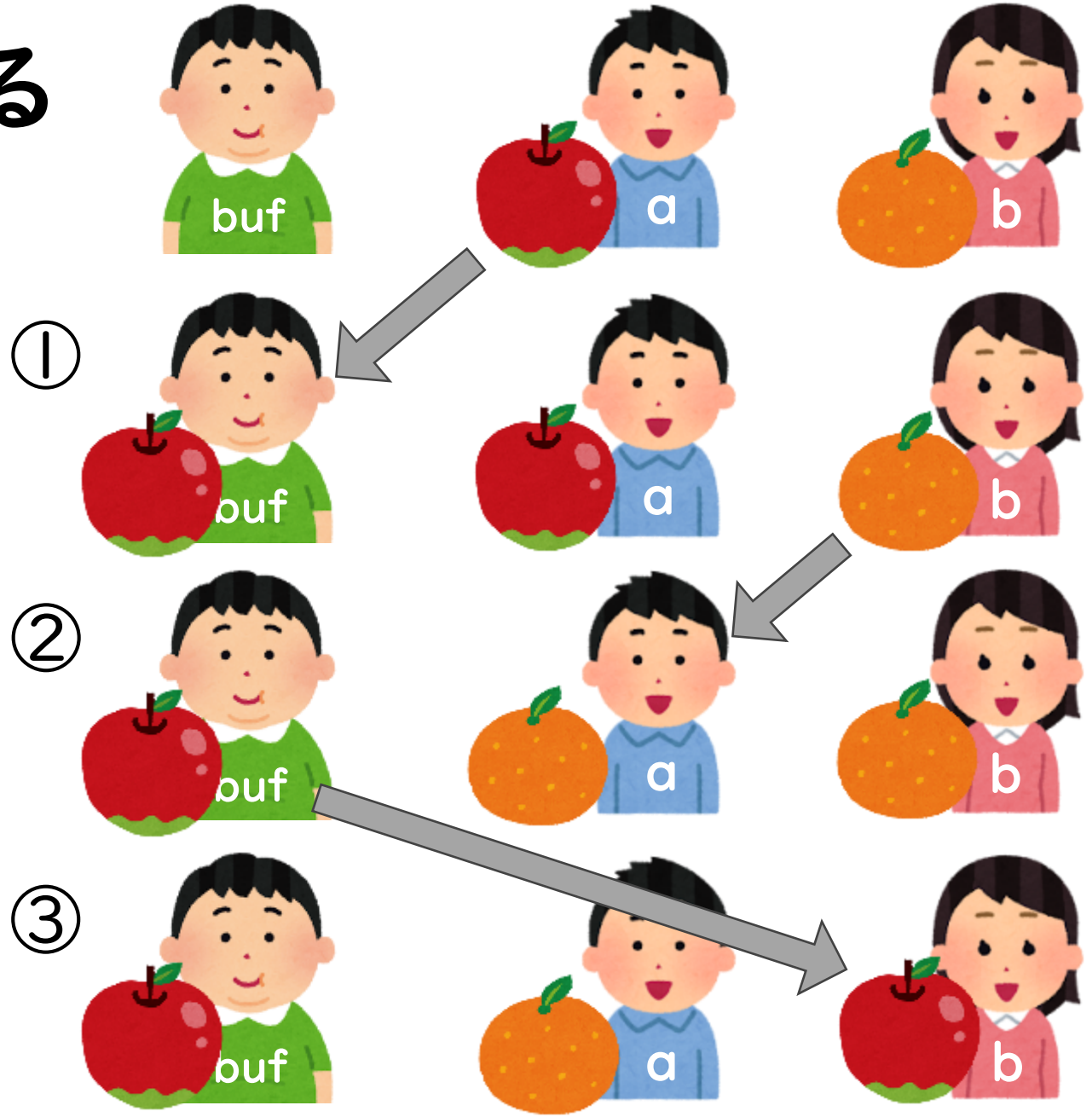
buf ← a　　・・・ ①

a ← b　　・・・ ②

b ← buf　　・・・ ③

別の変数に一時的に預けておく。

値をコピーしていき、入れ替える。



トレース表

流れ図（フローチャート）の経路を追いながら、変数の値が処理の過程でどのように変化していくかをまとめた表

整数型: m ← 1

整数型: n ← 3

m ← m + 1　　・・・ ①

n ← m　　・・・ ②

m ← m + n　　・・・ ③

トレース表

	m	n
初期値	1	3
①	2	3
②	2	2
③	4	2

手続 calcMod3 を呼び出したときの出力はどれか。

〔プログラム〕

```
○calcMod3()
  整数型: totalValue, i
  totalValue ← 0
  for (i を 1 から 7 まで 1 ずつ増やす)
    if (i ÷ 3 の余り が 0 と等しい)
      totalValue ← totalValue + i
    endif
  endfor
  totalValue を出力する
```

トレース表

1から7まで
1ずつ増やす

初期値0

i	totalValue

手続 calcMod3 を呼び出したときの出力はどれか。

〔プログラム〕

```
○calcMod3()
  整数型: totalValue, i
  totalValue ← 0
  for (i を 1 から 7 まで 1 ずつ増やす)
    if (i ÷ 3 の余り が 0 と等しい)
      totalValue ← totalValue + i
    endif
  endfor
  totalValue を出力する
```

トレース表

1から7まで 1ずつ増やす	初期値0
i	totalValue
1	0
2	0
3	0 + 3 = 3
4	3
5	3
6	3 + 6 = 9
7	9

気を付けたいポイント

境界値

ある範囲の最小値または最大値など
の同値分割した領域の端にあたる値

その数自体を含む、含まない場合によって、
＋1 や －1といった調整が必要!!

① ② ③ ④ ⑤ ⑥

a番目から b番目までの人数(a<b) ・・・・ b － a ＋１

a番目と b番目の人に挟まれている人数(a<b) ・・・・ b － a －１

要素が1から始まる配列： arr

整数型: i, arrLen　　//arrの長さ（要素の数）

arr ← {“いちご”, “ぶどう”, “みかん”, “りんご”, “なし”, “すいか”}

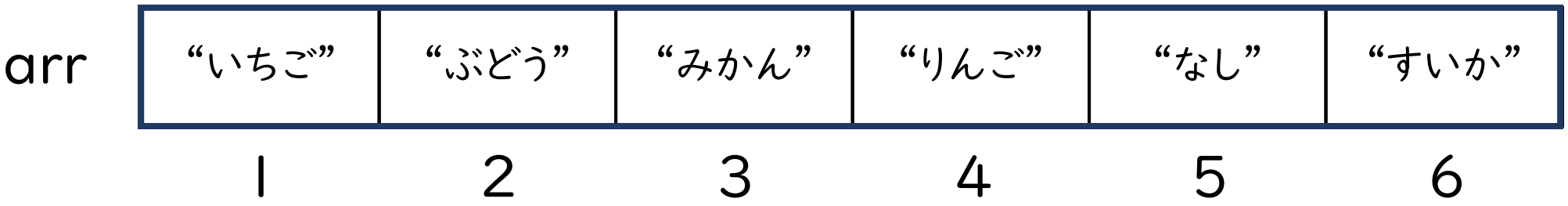
for (i を 1 から arrLen まで1ずつ増やす)

　　arr [i] を出力する

endfor

要素の数は 6なので、

要素番号は 1から6まで



要素が0から始まる配列： arr

整数型: i, arrLen　　//arrの長さ（要素の数）

arr ← {“いちご”, “ぶどう”, “みかん”, “りんご”, “なし”, “すいか”}

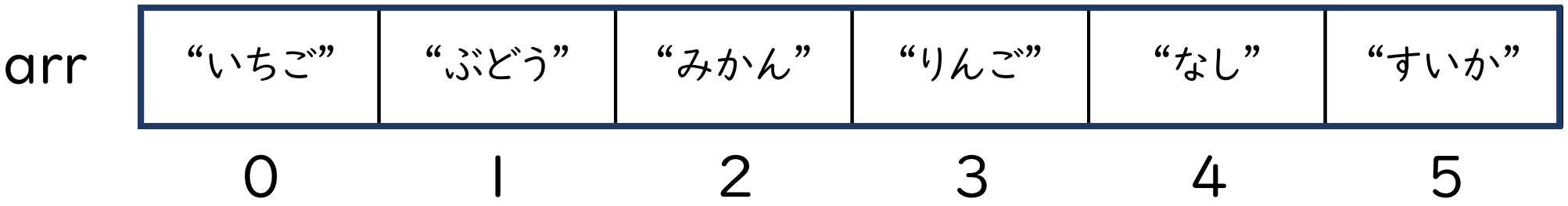
for (i を 0 から arrLen − 1 まで1ずつ増やす)

　　arr [i] を出力する

endfor

要素の数は 6なので、

要素番号は 0から5まで



a[arrLen] すなわち a[6]を呼びだそうとすると、定義されていないので、エラーになる

条件

〇〇より小さい　　〇〇以下

〇〇より大きい　　〇〇以上

値が等しいケースをどのように扱うか


①


②


③


④


⑤


⑥

この順番が身長順だった場合

a番目より身長が低い人数　・・・・・　0 ～ a－1

a番目以下の身長の人数　・・・・・　a － 1 ～ 全人数 － 1

全員同じ身長の場合がある

等しいときを含まない（〇〇より大きい）

整数型: a
a ← 3
while (a が 0 より大きい)　　a > 0
　　a ← a − 1
endwhile
a を出力する

0は条件を満たさない

ループ	a
初期値	3
1回目	2
2回目	1
3回目	0

出力結果：　0

等しいときを含む（〇〇以上）

整数型: a
a ← 3
while (a が 0 以上)　　a ≧ 0
　　a ← a − 1
endwhile
a を出力する　　0は条件を満たす

ループ	a
初期値	3
1回目	2
2回目	1
3回目	0
4回目	-1

出力結果：− 1

ある施設の入場料は0歳から5歳までは無料、6歳から11歳までは300円、12歳以上は500円である。関数feeは、年齢を表す0以上の整数を引数として受け取り、戻り値として入場料を返す。

```
〇整数型：fee (整数型：age)
整数型: ret
if (　ageが　　①　　のとき)
　　ret ← 0
elseif (　ageが　　②　　のとき)
　　ret ← 300
else
　　ret ← 500
endif
return ret
```

パターンA
① 0以上5以下　　②6以上11以下

パターンB
① 0以上6未満　　②6以上12未満

パターンC
① 5以下　　②11以下

パターンD
① 6未満　　②12未満

パターンE
① 0, 1, 2, 3, 4, 5　　② 6, 7, 8, 9, 10, 11

ある施設の入場料は0歳から5歳までは無料、6歳から11歳までは300円、12歳以上は500円である。関数feeは、年齢を表す0以上の整数を引数として受け取り、戻り値として入場料を返す。

```
〇整数型：fee (整数型：age)
整数型: ret
if (　ageが　　①　　のとき)
        ret ← 0
elseif (　ageが　　②　　のとき)
        ret ← 300
else
        ret ← 500
endif
return ret
```

パターンA
① 0以上5以下　　②6以上11以下

パターンB
① 0以上6未満　　②6以上12未満

パターンC
① 5以下　　②11以下

パターンD
① 6未満　　②12未満

パターンE
① 0, 1, 2, 3, 4, 5　　② 6, 7, 8, 9, 10, 11

すべて正解!!

境界値テスト

「〇〇以上」や「〇〇未満」などの値の境界となるところをテストする

ある施設の入場料は0歳から5歳までは無料、6歳から11歳までは300円、12歳以上は500円である。
関数feeは、年齢を表す0以上の整数を引数として受け取り、戻り値として入場料を返す。

境界前後で正しい結果を返す　⇒　すべての条件で正しいことが期待される!

※厳密には最小値・最大値の境目もテストする。
-1，0，人の最大の年齢を150歳とすると、150，151

ここから実践問題

次のプログラムを実行すると出力されるのは？

整数型: x ← 1
整数型: y ← 2
整数型: z ← 3
x ← y
y ← z
z ← x

yの値とzの値をカンマ区切りで出力する

次のプログラムを実行すると出力されるのは？

整数型: x ← 1
整数型: y ← 2
整数型: z ← 3
x ← y　　・・・・ ①
y ← z　　・・・・ ②
z ← x　　・・・・ ③

	x	y	z
初期値	1	2	3
①			
②			
③			

yの値とzの値をカンマ区切りで出力する

次のプログラムを実行すると出力されるのは？

整数型: x ← 1
整数型: y ← 2
整数型: z ← 3
x ← y　　・・・・ ①
y ← z　　・・・・ ②
z ← x　　・・・・ ③

	x	y	z
初期値	1	2	3
①	2	2	3
②	2	3	3
③	2	3	2

yの値とzの値をカンマ区切りで出力する

出力結果：3，2

関数 calcMean は要素数が1以上の実数型の一次元配列 dataArrayを引数として受け取り、要素の値の平均値を戻り値として返す。

```
〇実数型: calcMean （実数型の一次元配列: dataArray ）
実数型: sum, mean
整数型: i
for ( i を1 から dataArrayの要素数まで1ずつ増やす)
        sum ← ①
endfor
mean ← ②
return mean
```

①の候補

A. i
B. dataArray[i]
C. sum + i
D. sum + dataArray[i]

②の候補

A. sum × dataArrayの要素数
B. sum ÷ dataArrayの要素数
C. mean × dataArrayの要素数
D. mean ÷ dataArrayの要素数

関数 calcMean は要素数が1以上の実数型の一次元配列 dataArrayを引数として受け取り、要素の値の平均値を戻り値として返す。

○実数型: calcMean （実数型の一次元配列: dataArray ）
実数型: sum, mean
整数型: i
for (i を1 から dataArrayの要素数まで1ずつ増やす)
　　sum ← ①
endfor
mean ← ②
return mean

①の候補
A. i
B. dataArray[i]
C. sum + i
D. sum + dataArray[i]

②の候補
A. sum × dataArrayの要素数
B. sum ÷ dataArrayの要素数
C. mean × dataArrayの要素数
D. mean ÷ dataArrayの要素数

関数 calcMax は要素数が1以上の実数型の一次元配列 dataArrayを引数として受け取り、要素の最大値を戻り値として返す。

```
○実数型: calcMax（実数型の一次元配列: dataArray ）
実数型: max
整数型: i
for ( i を1 から dataArrayの要素数まで1ずつ増やす)
        if (      ①      )
                max ← dataArray[ i ]
        endif
endfor
return max
```

①の候補

A. i が dataArray[i] より小さいとき

B. i が dataArray[i] より大きいとき

C. max が dataArray[i] より小さいとき

D. max が dataArray[i] より大きいとき

関数 calcMax は要素数が1以上の実数型の一次元配列 dataArrayを引数として受け取り、要素の最大値を戻り値として返す。

```
○実数型: calcMax（実数型の一次元配列: dataArray ）
実数型: max
整数型: i
for ( i を1 から dataArrayの要素数まで1ずつ増やす)
        if (      ①      )
                max ← dataArray[ i ]
        endif
endfor
return max
```

①の候補

A. i が dataArray[i] より小さいとき

B. i が dataArray[i] より大きいとき

C. max が dataArray[i] より小さいとき

D. max が dataArray[i] より大きいとき

関数 frameStar は1文字以上の任意の全角の文字列 inputを引数として受け取り、その文字列の周りを”★”で囲んで表示する。

inputは1行で表示が可能かつ、文字の幅は全て等しいものとする。

例）あいうえお　⇒

★★★★★★★
★あいうえお★
★★★★★★★

○ 関数 frameStr (文字列型 : input)
整数型: inputLength　// inputの文字数
整数型: i
for (i が1から　①　まで 1ずつ増やす)
　　“★”を表示する
endfor
改行する
“★”を表示する
inputを表示する
“★”を表示する
改行する
for (i が1から　①　まで 1ずつ増やす)
　　“★”を表示する
endfor

①の候補

A. inputLength - 1
B. inputLength
C. inputLength + 1
D. inputLength + 2

関数 frameStar は1文字以上の任意の全角の文字列 inputを引数として受け取り、その文字列の周りを”★”で囲んで表示する。

inputは1行で表示が可能かつ、文字の幅は全て等しいものとする。

例）あいうえお　⇒

★★★★★★★
★あいうえお★
★★★★★★★

○ 関数 frameStr (文字列型 : input)
整数型: inputLength　// inputの文字数
整数型: i
for (i が1から　①　まで 1ずつ増やす)
　　“★”を表示する
endfor
改行する
“★”を表示する
inputを表示する
“★”を表示する
改行する
for (i が1から　①　まで 1ずつ増やす)
　　“★”を表示する
endfor

①の候補

A. inputLength - 1
B. inputLength
C. inputLength + 1
D. inputLength + 2

関数 isPalindrome は1文字以上の任意の文字列 strを引数として受け取り、その文字列が回文のときは'YES'、回文でないときは'NO'という文字列を戻り値として返す。

```
○ isPalindrom（文字列型: str）
整数型: n ← 1
整数型: strLen ← str の文字列の長さ
文字列型: judge ← 'YES'
while（　　①　　）
        if（　　②　　）
                judge ← 'NO'
        endif
        n ← n + 1
endwhile
return judge
```

関数 isPalindrome は1文字以上の任意の文字列 strを引数として受け取り、その文字列が回文のときは'YES'、回文でないときは'NO'という文字列を戻り値として返す。

```
○ isPalindrom（文字列型: str）
整数型: n ← 1
整数型: strLen ← str の文字列の長さ
文字列型: judge ← 'YES'
while（　　①　　）
        if（　　②　　）
                judge ← 'NO'
        endif
        n ← n + 1
endwhile
return judge
```

解答例

① n が strLen/2（切り捨て） 以下のとき

②strの n番目の文字と strの (strLen+1-n)番目の文字が等しくないとき

プログラミング能力・アルゴリズムの理解度を測るには
paiza転職のスキルチェックがおすすめ!!

https://paiza.jp/career

自分のレベルに合わせてトライできる!!

一度は確認したいアルゴリズム

- 並び替え（ソート）
  - バブルソート
  - 選択ソート
  - 挿入ソート
  - クイックソート
  - マージソート
  - ヒープソート
- 探索（サーチ）
  - 線形探索
  - 二分探索
- その他
  - FizzBuzz問題
  - ユークリッドの互除法による最大公約数算出

参考URL

- 【ITパスポート試験】 情報処理推進機構
  https://www3.jitec.ipa.go.jp/JitesCbt/index.html
- ふっくゼミ アルゴリズム補講 基本編
  https://xn--tpto73d.jp/mobile/movie/ar
- アルゴリズムとは何か～ 文系理系問わず楽しめる精選 6 問 ～
  https://qiita.com/drken/items/f909b79ee03e679c7142
- 【Unity】ソートアルゴリズム12種を可視化してみた
  https://qiita.com/r-ngtm/items/f4fa55c77459f63a5228
- paiza転職 プログラミングスキルチェック
  https://paiza.jp/challenges/info