Algorithm BootCamp

ver1.3

class:

name:

Chapter 1 疑似言語の前提知識

Section 1 記号の意味

○ ・・・ プログラム名、変数の定義、手続き(※メソッド、関数)

● ・・・ 処理

▲ ・・・ 条件式(判断)

■ ・・・ 条件式(繰返し)

Section 2 宣言部に記述する内容

①プログラム名

ex)

○ プログラム名:加算プログラム

○ プログラム名:主プログラム

○ プログラム名:副プログラム ・・・ 主プログラムから呼び出されるプログラム。
たまに、プログラム名がヒントになることもあります。

②変数の定義

プログラム中に使用する変数の一覧を記述する。もちろん、変数の型も指定します。

【型の種類】

整数型、文字型、実数型、論理型

外部参照・大域変数 ・・・ どのプログラムからでも参照可能な変数。

ex)

○ プログラム名:加算プログラム

○ 整数型:su1, su2, Ans ⇒ この変数の一覧が結構重要

③手続き

自プログラムから呼び出す別のプログラムを指定する。

ex)

○ プログラム名:加算プログラム

○ 整数型:su1, su2, Ans

○ 手続:結果表示プログラム(Ans) ※引数:プログラムに渡す値のこと

Section 3 処理部に記述する内容

そのプログラムで行う実際の処理を記述する部分です。

ex)

○ プログラム名：加算プログラム

○ 整数型：su1，su2，Ans

○ 手続：結果表示プログラム(Ans)　※引数：プログラムに渡す値のこと

- su1 ← 10　　//数値の格納
- su2 ← 20　　//数値の格納
- Ans ← su1 ＋ su2　　//加算処理
- 結果表示プログラム(Ans)　//副プログラムの呼び出し

※簡単なプログラムのうちから、処理に対して『コメント』を付けるようにしよう。

【ポイント】

上記の例はまだ簡単なプログラムですが、実際の国家試験のプログラムは長くて見るだけでも嫌気が指すようなものばかりです。しかし、どんなに長いプログラムであっても【処理の塊】は存在します。その『処理の塊』をいかに早く見つけられるかが重要になってきます。

ex)

○ プログラム名

○ 変数定義

○ 手続の指定

※ 左記のように、どれだけ長いプログラムでも細分化することによって処理内容を理解しやすくできる。
ただし、問題特有の処理等もあるのでそもそも問題として何の処理を行うかが分からないと細分化は難しい。

Section 4 ストラクチャード定理

すべてのプログラムはストラクチャード定理で記述されています。ストラクチャード定理とは、『順次・選択・繰返し』の形式でありプログラムはこの形式の組み合わせで実現されます。そのため、プログラムの基本であるストラクチャード定理のマスターが必須条件です。

①順次処理

上から下へ順番に処理を実行していく。一番オーソドックスな形式。

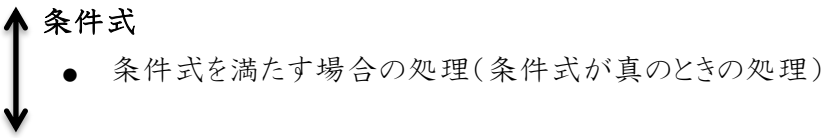
【形式】

- 処理1
- 処理2
- 処理3 ．．．

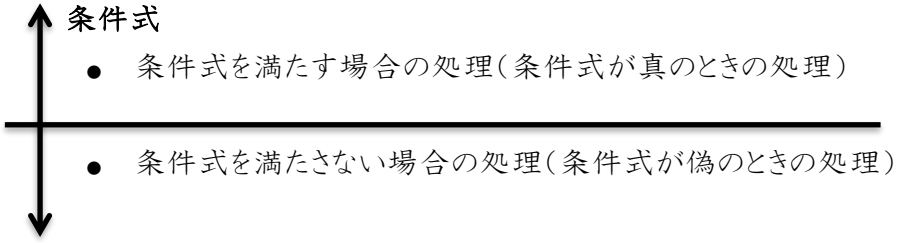
②選択処理

『条件式』によって、処理を実行するか・どの処理を実行するかを選択・判断する。

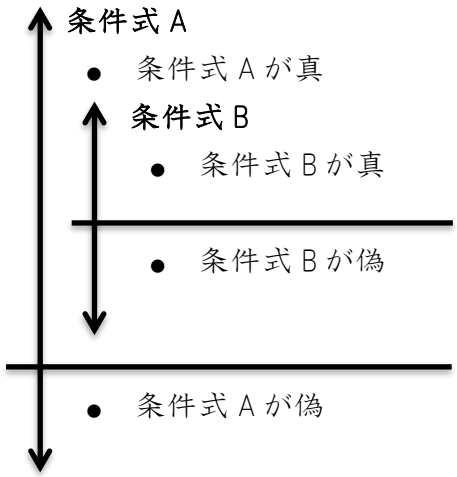
【形式1】



【形式2】



【形式3：入れ子】



※条件式の入れ子はよく出題される。
落ち着いてゆっくり見ること。

③繰返し処理

ループ記号の条件式(継続条件)を満たす間、ループ記号内の処理を繰り返す。

【形式1:前判定】

条件式
- 処理1
- 処理2 ．．．

※最低0回、ループ内の処理を実行する。
最初から条件式が偽となり、ループ内に入らないこともある。

【形式2:後判定】

- 処理1
- 処理2 ．．．

条件式

※最低1回、ループ内の処理を実行する。
（条件式の真偽に関わらず絶対1回処理実行）

【形式3:多重ループ】

条件式A
- 処理A－1
- **条件式B**
  - 処理B－1
  - 処理B－2 ．．．
- 処理A－2 ．．．

※多重ループは2次元配列などでよく使用される。

条件式Aのループ処理の中で、条件式Bのループ処理を実行する。
条件式Bのループ処理は条件式Aが偽になるまで何回も何回も実行される。
プログラムとして、多重ループやループ内での選択処理はあまり良くありません。処理速度が遅くなったり、可読性が下がったりしてしまいます。(要するに、あまり良くないプログラム)

【サンプル】

○ プログラム名：サンプルプログラム

○ 整数型：SU, SUM

○ 論理型：FLG

○ 手続：SumDisp(SUM), InputData(SU)

- InputData(SU)
- SUM ← 0
- FLG ← FALSE
- FLG ≠ TRUE
  - SUM ← SUM + SU
  - SU ← SU − 1
  - SU = 0
    - FLG ← TRUE
- SumDisp(SUM)

【トレース表サンプル】

SU	SUM	FLG
?	?	?
10	0	FALSE
9	10	
8	19	
7	27	
6	34	
5	40	
4	45	
3	49	
2	52	
1	54	
0	55	TRUE

Chapter 2 トレース

Section 1 トレースとは

プログラム内で定義されている変数の値の変化を記述したものです。
このトレースの力は国家試験に合格するためだけに必要な力ではなく、プログラムやシステムエンジニアにおいては当たり前の力であるため、根気強く頑張っていきましょう。

Section 2 トレース表の書き方

よく『トレース表の書き方が分からない．．．そもそも何？』という人がいますが、トレース表自体はものすごく簡単です。

ex）

○プログラム名：足し算プログラム
○整数型：su1，su2，ans
○手続き：DispAns（ans）
　　．．．以下処理

網掛け部分の変数定義に注目してください。この部分をそのまま表にするだけです。
定義されている変数をすべて横並びに記述すれば完成です。

su1	su2	ans
?	?	?
．．．	．．．	．．．

実際には上記のように枠線を書く必要はありませんし、自分の見やすいように記述できればよいです。変数が多数ある場合は関連のある変数を近くに記述するなど工夫してください。

トレース表は、一つ一つの処理を実行していき変数の値が変化する度に下に値を書きます。そのため、トレース表は縦長になっていきます。また、変数を定義した段階では変数に何が格納されているかは不明なので、初期化処理が必ず必要です。

しかし、国家試験で出題される疑似言語のプログラムは変数の数も長さも膨大であり、すべてトレースするのは現実的には不可能です。そのため、必要な部分を必要なだけ正確にトレースできるようになりましょう。

Chapter 3 順次処理とトレース

一番オーソドックスな順次処理のトレース練習をやっていきます。下記のプログラムをトレースしていきましょう。最初の方は簡単です。

Section 1 初期化処理

○ プログラム名:PG1－1 ○ 整数型:SU1 ● SU1 ← 0						

○ プログラム名:PG1－2 ○ 整数型:SU2 ● SU2 ← 100						

○ プログラム名:PG1－3 ○ 論理型:SW1 ● SW1 ← true						

○ プログラム名:PG1－4 ○ 論理型:SW2 ● SW2 ← false						

○ プログラム名:PG1－5 ○ 整数型:Sum, Cnt ● Sum ← 0 ● Cnt ← 0						

○ プログラム名:PG1－6
○ 整数型:SU1, SU2
○ 論理型:Flg
- SU1 ← 10
- SU2 ← 7
- Flg ← false

Section 2 通常の格納

○ プログラム名:PG2－1
○ 整数型:SU1, SU2
- SU1 ← 5
- SU2 ← SU1

○ プログラム名:PG2－2
○ 整数型:SU1, SU2
- SU1 ← 2
- SU2 ← 4
- SU1 ← SU2

○ プログラム名:PG2－3
○ 論理型:SW1, SW2
- SW1 ← true
- SW2 ← SW1

○ プログラム名:PG2－4
○ 論理型:SW1, SW2
- SW1 ← false
- SW2 ← true
- SW2 ← SW1

○ プログラム名:PG2－5
○ 整数型:SU1, SU2
○ 論理型:SW1, SW2
- SW1 ← false
- SU2 ← 5
- SW1 ← true
- SW2 ← SW1
- SU1 ← SU2
- SU2 ← 4
- SW2 ← false

Section 3 計算結果の格納

○ プログラム名:PG3－1
○ 整数型:Sum
- Sum ← 5 + 10

○ プログラム名:PG3－2
○ 整数型:Sum
- Sum ← 0
- Sum ← Sum + 6

○ プログラム名:PG3－3
○ 整数型:Sum, SU
- SU ← 20
- Sum ← SU － 10

○ プログラム名：PG3－4
○ 整数型：Sum，SU
- Sum ← 0
- SU ← 100
- Sum ← Sum ＋ SU

○ プログラム名：PG3－5
○ 整数型：Res，SU1，SU2
- SU1 ← 100
- SU2 ← 2
- Res ← SU1 ／ SU2

Section 4 入替処理

○ プログラム名：PG4－1
○ 整数型：SU1，SU2，wk
- SU1 ← 7
- SU2 ← 2
- wk ← SU1
- SU1 ← SU2
- SU2 ← wk

○ プログラム名：PG4－2
○ 論理型：SW1，SW2，wk
- SW2 ← false
- wk ← SW2
- SW1 ← true
- SW2 ← SW1
- SW1 ← wk

○ プログラム名 :PG4－3						
○ 整数型:SU1, SU2, wk1						
○ 論理型:SW1, SW2, wk2						
● SW1 ← true						
● SU1 ← 5						
● SW2 ← false						
● SU2 ← 8						
● wk2 ← SW1						
● wk1 ← SU2						
● SW1 ← SW2						
● SU2 ← SU1						
● Sw2 ← wk2						
● SU1 ← wk1						
● wk1 ← SU2						
● wk2 ← SW1						
● SU2 ← SU1						
● SW1 ← SW2						
● SW2 ← wk2						
● SU1 ← wk1						

Chapter 4 条件分岐処理とトレース

『条件式』によって処理実行の有無を判断したり、実行する処理を選択したりします。この条件分岐で躓いている人もいるかと思いますが、簡単なプログラムから練習していきましょう。

Section 1 条件分岐①

○ プログラム名：PG5－1
○ 整数型：SU
● SU ← 100
SU ＜ 10
　● SU ← SU ＋ 10

○ プログラム名：PG5－2
○ 整数型：SU
● SU ← 0
SU ＜ 10
　● SU ← SU ＋ 10

○ プログラム名：PG5－3
○ 整数型：SU1，SU2
● SU1 ← 5
● SU2 ← 3
SU1 ＞ SU2
　● SU1 ← SU1 － SU2

○ プログラム名：PG5－4
○ 整数型：SU1，SU2
● SU1 ← 7
● SU2 ← 8
（SU1 ＋ SU2） ＞ 10
　● SU1 ← SU1 ＋ SU2

○ プログラム名：PG5－5
○ 整数型：SU
○ 論理型：SW
● SW ← true
● SU ← 0
SW //SW＝trueで真の処理を行う
　● SU ← 10

○ プログラム名：PG5－6
○ 整数型：SU
○ 論理型：SW
● SW ← false
SW
　● SU ← 10

○ プログラム名：PG5－7
○ 整数型：SU
○ 論理型：SW
● SU ← 15
● SW ← false
SU ＞ 10
　● SW ← true

SW
　● SU ← SU － 1

○ プログラム名：PG5－8
○ 整数型：SU1，SU2
○ 論理型：Flg
● SU1 ← 10
● SU2 ← 5
● Flg ← false
（SU1 － SU2） ≧ 5
　● Flg ← true
Flg
　● SU1 ← 0

Section 2　条件分岐②

○ プログラム名：PG6－1
○ 整数型：SU
● SU ← 5
SU ＞ 0
　● SU ← SU － 1
　● SU ← SU ＋ 1

○ プログラム名：PG6－2
○ 整数型：SU
● SU ← －10
SU ＞ 0
　● SU ← SU － 1
　● SU ← SU ＋ 1

○ プログラム名：PG6－3
○ 整数型：SU1，SU2，Res
● SU1 ← 7
● SU2 ← 2
SU1 ＞ SU2
● Res ← SU1 － SU2

● Res ← SU2 － SU1

○ プログラム名：PG6－4
○ 整数型：SU1，SU2，Res
● SU1 ← 4
● SU2 ← 4
SU1 ＞ SU2
● Res ← SU1 － SU2

● Res ← SU2 － SU1

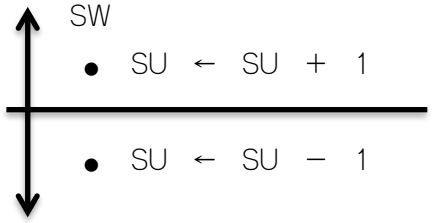
○ プログラム名：PG6－5
○ 整数型：SU
○ 論理型：SW
● SW ← true
SW
● SU ← 100

● SU ← 0

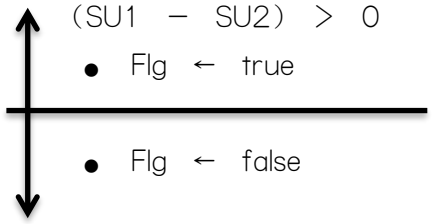
○ プログラム名：PG6－6
○ 整数型：SU
○ 論理型：SW
● SW ← false
SW
● SU ← 100

● SU ← 0

○ プログラム名：PG6－7
○ 整数型：SU
○ 論理型：SW
● SU ← 0
● SW ← true

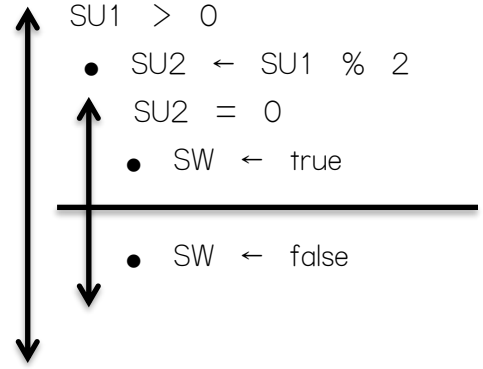


○ プログラム名：PG6－8
○ 整数型：SU1，SU2
○ 論理型：Flg
● SU1 ← 8
● SU2 ← 16



Section 3　条件分岐③

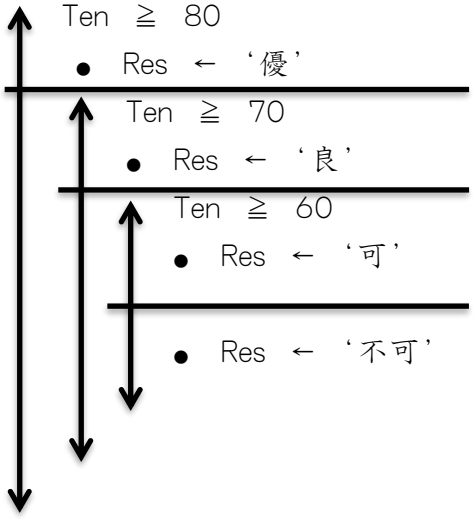
○ プログラム名：PG7－1
○ 整数型：SU1，SU2
○ 論理型：SW
● SU1 ← 10



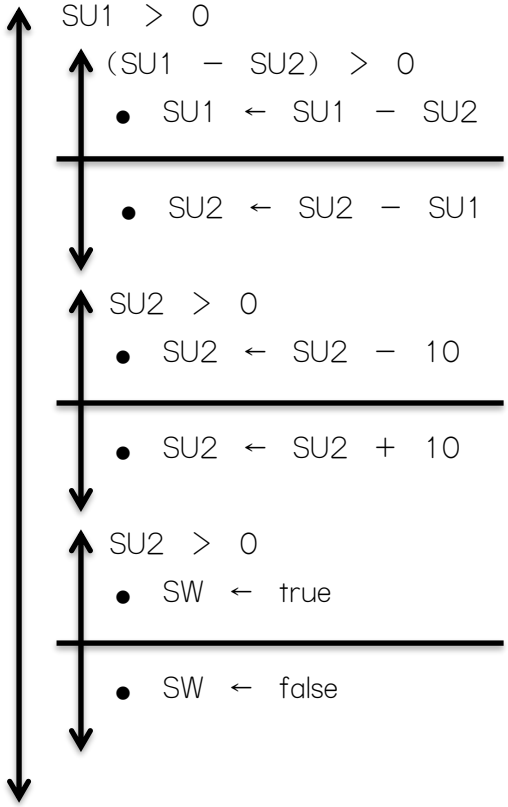
○ プログラム名：PG7－2
○ 整数型：SU1，SU2
○ 論理型：SW
● SU1 ← 10
SU1 ＞ 0
● SU2 ← SU1 － 100
SU1 ＞ 0
● SW ← true
● SW ← false

○ プログラム名：PG7－3
○ 整数型：SU1，SU2
● SU1 ← 5
SU1 ＞ 0
● SU2 ← SU1 ＊ 2
SU2 ≧ 10
● SU2 ← SU2 ＊ 10
● SU2 ← SU2 － SU2
● SU1 ← SU2 ／ 2

○ プログラム名：PG7－4
○ 文字列型：Res
○ 整数型：Ten
● Ten ← 70



○ プログラム名：PG7－5
○ 整数型：SU1，SU2
○ 論理型：SW
● SU1 ← 10
● SU2 ← 20



Chapter 5 繰返し処理とトレース

繰返し処理です。ここから本格的に分からなくなってくる人が出てくると思います。とにかく、アルゴリズムは理解ではなく慣れが重要ですので1行1行実行してみましょう。

Section 1 繰返し処理①

○ プログラム名：PG8－1
○ 整数型：Res, Cnt
- Res ← 0
- Cnt ← 0
- Cnt ＜ 10
  - Res ← Res ＋ Cnt
  - Cnt ← Cnt ＋ 1

○ プログラム名：PG8－2
○ 整数型：Res, Cnt
- Res ← 0
- Cnt ← 1
- Cnt ＜ 10
  - Res ← Res ＋ Cnt
  - Cnt ← Cnt ＋ 1

○ プログラム名 :PG8－3

○ 整数型 :Res, Cnt

- Res ← 0
- Cnt ← 0
- Cnt ≦ 10
  - Res ← Res ＋ Cnt
  - Cnt ← Cnt ＋ 1

○ プログラム名 :PG8－4

○ 整数型 :Res, Cnt

- Res ← 0
- Cnt ← 1
- Cnt ≦ 10
  - Res ← Res ＋ Cnt
  - Cnt ← Cnt ＋ 1

○ プログラム名 :PG8－5

○ 整数型 :Res, Cnt

- Res ← 0
- Cnt ← 1
- Cnt ≦ 5000
  - Res ← Res ＋ Cnt
  - Cnt ← Cnt ＊ 3

○ プログラム名 :PG8－6

○ 整数型 :SU1, SU2, SU3, Cnt

- SU1 ← 0
- SU2 ← 0
- SU3 ← 0
- Cnt ← 1
- Cnt ≦ 5
  - SU1 ← SU1 ＋ 3
  - SU2 ← SU2 ＋ 1
  - SU3 ← SU3 ＋ 2
  - SU1 ← SU1 ＋ SU3
  - SU2 ← SU2 ＋ SU1
  - SU3 ← SU3 ＋ SU2
  - Cnt ← Cnt ＋ 1

○ プログラム名 :PG8－7

○ 整数型:SU, Sum, Avg, Cnt

- SU ← 3
- Sum ← 0
- Cnt ← 1
- SU ＜ 30
  - Sum ← Sum ＋ SU
  - SU ← SU ＋ 3
  - Cnt ← Cnt ＋ 1
- Avg ← Sum ／ Cnt

○ プログラム名 :PG8－8

○ 整数型:SU1, SU2, Data

- SU1 ← 4321
- SU2 ← 1234
- Data ← SU1 % SU2
- Data ≠ 0
  - SU1 ← SU2
  - SU2 ← Data
  - Data ← SU1 % SU2

Section 2　繰返し処理②

さて、ここから鬼門に入ります。とにかく頑張れ。

○　プログラム名：PG9－1

○　整数型：SU, Sum

- SU　←　1
- Sum　←　0

■ SU　≦　10

　↕ （SU　%　2）　＝　0

　　- Sum　←　Sum　＋　SU

　- SU　←　SU　＋　1

■

○　プログラム名：PG9－2

○　整数型：SU, Sum

- SU　←　1
- Sum　←　0

■ SU　≦　10

　↕ （SU　%　2）　＝　1

　　- Sum　←　Sum　＋　SU

　- SU　←　SU　＋　1

■

○ プログラム名：PG9－3
○ 整数型：SU, Sum
● SU ← 1
● Sum ← 0
■ SU ≦ 15
　↕ （SU % 3） = 0
　　● Sum ← Sum + SU
　● SU ← SU + 1

○ プログラム名：PG9－4
○ 整数型：SU, Sum, Data
○ 論理型：SW
● SU ← 0
● Sum ← 0
● SW ← true
■ SW
　● SU ← SU + 1
　● Sum ← Sum + SU
　↕ SU ≧ 15
　　● Data ← Sum ／ SU
　　● SW ← false

○ プログラム名：PG9－5

○ 整数型：SU，Sum1，Sum2，
Sum3，Sum4

○ 論理型：SW

- SU ← 0
- Sum1 ← 0
- Sum2 ← 0
- Sum3 ← 0
- Sum4 ← 0
- SW ← true

■ SW

- SU ← SU ＋ 1

（SU ％ 2） ＝ 0

- Sum1 ← Sum1 ＋ SU

（SU ％ 3） ＝ 0

- Sum2 ← Sum2 ＋ SU

（SU ％ 4） ＝ 0

- Sum3 ← Sum3 ＋ SU

（SU ％ 5） ＝ 0

- Sum4 ← Sum4 ＋ SU

SU ＞ 30

- SW ← false

■

○ プログラム名：PG9－6
○ 整数型：SU，Sum1，Sum2
- SU ← 1
- Sum1 ← 0
- Sum2 ← 0
- SU ≦ 15
  - （SU ％ 2） ＝ 0
    - Sum1 ← Sum1 ＋ SU
  - Sum2 ← Sum2 ＋ SU
  - SU ← SU ＋ 1

○ プログラム名：PG9－7
○ 整数型：SU，Sum1，Sum2
○ 論理型：SW
- SU ← 1
- Sum1 ← 0
- Sum2 ← 0
- SW ← true
- SU ＜ 15
  - SW
    - Sum2 ← Sum1 ＋ SU
    - SW ← false
  - Sum1 ← Sum2 ＋ SU
  - SW ← true
  - SU ← SU ＋ 1

Section 3　繰返し処理③

ループなど一回動きが分かれば大したことない！！所詮同じことの繰り返しだ！！！！

○ プログラム名:PG10－1
○ 整数型:SU1, SU2, SU3, Cnt
- SU1 ← 0
- SU2 ← 1
- SU3 ← 2
- Cnt ← 0
- 表示1に出力(SU2)
- 表示2に出力(SU3)
- SU1 ＜ 4
  - SU1 ← SU1 ＋ 1
  - SU2 ← 1
  - Cnt ← 0
  - Cnt ＜ SU1
    - SU2 ← SU2 ＊ 2
    - Cnt ← Cnt ＋ 1
  - 表示1に出力(SU2)
  - Cnt ← 0
  - Cnt ＜ SU1
    - SU3 ← SU3 ＊ 2
    - Cnt ← Cnt ＋ 1
  - 表示2に出力(SU3)

				表示1	表示2

○ プログラム名:PG10－2

○ 整数型:SU

● SU ← 0

■ SU ≦ 30

（SU % 15） = 0

● 表示に出力('FizzBuzz')

（SU % 5） = 0

● 表示に出力('Buzz')

（SU % 3） = 0

● 表示に出力('Fizz')

● 表示に出力(SU)

● SU ← SU + 1

				表示

○ プログラム名：PG10－3

○ 整数型：SU, Sum1, Sum2, Cnt

- SU ← 1
- Cnt ← 0
- Sum1 ← 0
- Sum2 ← 0
- SU ＜ 8
  - （SU ％ 2） ＝ 0
    - Cnt ＜ SU
      - Sum1 ← Sum1 ＋ SU
      - Cnt ← Cnt ＋ 1
  - Cnt ＜ SU
    - Sum2 ← Sum2 ＋ SU
    - Cnt ← Cnt ＋ 1
  - Cnt ← 0
  - SU ← SU ＋ 1

Chapter 6 配列とトレース

まだまだこれから！！！！やれ！！

Section 1 配列（1次元配列）

配列とは、複数の値を格納できる変数です。変数を何十個と定義するより、一つの配列にまとめると、プログラムをより効率的に作ることができます。

①配列の定義

配列を利用するには、宣言部に記述します。

変数と同じように型を指定します。また、要素数（何個の値を格納できるか）も記述します。

ex） 整数を5個格納できる変数をarrayという名前で定義する

○ 整数型:array[5]

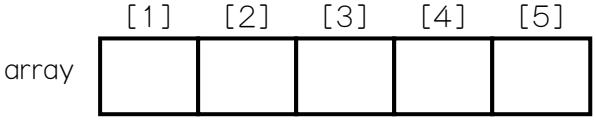
ex） 文字を10個格納できる変数をcharsという名前で定義する

○ 文字型:chars[10]

②配列の利用

配列は小さな箱を集めたようなものです。

整数型:array[5] と定義した配列は以下のようなイメージとなります。



この配列arrayに値を代入するには、複数ある要素（1つ1つの箱）の中から、1つを選ばなければなりません。1つの要素を選ぶには、要素番号（添え字）を利用します。

ex） arrayの要素に値を代入する

- array[1] ← 10
- array[3] ← 30
- array[5] ← array[1] ＋ array[3]

処理後の配列arrayの中身は以下となります。

	[1]	[2]	[3]	[4]	[5]
array	10		30		40

○ プログラム名:PG11－1

○ 整数型:T[5]

- T[1] ← 0
- T[2] ← 1
- T[3] ← 2
- T[4] ← 3
- T[5] ← 4

	[1]	[2]	[3]	[4]	[5]
T					
T					
T					
T					
T					

○ プログラム名:PG11－2

○ 整数型:T[5],i

- i ← 1
- T[i] ← 1
- i ← i ＋ 1
- T[i] ← 2
- i ← i ＋ 1
- T[i] ← 3
- i ← i ＋ 1
- T[i] ← 4
- i ← i ＋ 1
- T[i] ← 5

i		[1]	[2]	[3]	[4]	[5]
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					

○ プログラム名:PG11－3

○ 整数型:T[5],i

- i ← 1
- i ≦ 5
  - T[i] ← 0
  - i ← i ＋ 1

i		[1]	[2]	[3]	[4]	[5]
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					

○ プログラム名:PG11－4

○ 整数型:T[5],i

- i ← 1
- i ≦ 5
  - T[i] ← i
  - i ← i ＋ 1

i		[1]	[2]	[3]	[4]	[5]
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					

○ プログラム名:PG11－5
○ 整数型:T[5],i
● i ← 1
■ i ≦ 5
　● T[i] ← i ＊ 2
　● i ← i ＋ 1

i		[1]	[2]	[3]	[4]	[5]
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					

○ プログラム名:PG11－6
○ 整数型:T[5],i
● T[1] ← 1
● i ← 2
■ i ≦ 5
　● T[i] ← T[i－1] ＋ i
　● i ← i ＋ 1

i		[1]	[2]	[3]	[4]	[5]
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					

○ プログラム名:PG11－7
○ 整数型:T[5],i
● i ← 5
■ i ＞ 0
　● T[i] ← 6 － i
　● i ← i － 1

i		[1]	[2]	[3]	[4]	[5]
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					
	T					

○ プログラム名:PG11－8

○ 整数型:T[5]＝{4, 6, 3, 8, 5}, sum, i

- sum ← 0
- i ← 1
- i ≦ 5
  - sum ← sum ＋ T[i]
  - i ← i ＋ 1

○ プログラム名:PG11－9

○ 整数型:T[5], su, i

- su ← 1
- i ← 1
- i ≦ 5
  - su % 2 ＝ 0
    - T[i] ← su
    - i ← i ＋ 1
  - su ← su ＋ 1

su	i		[1]	[2]	[3]	[4]	[5]
		T					
		T					
		T					
		T					
		T					
		T					
		T					
		T					
		T					
		T					
		T					
		T					
		T					
		T					
		T					
		T					
		T					

○ プログラム名:PG11－10
○ 整数型:A[5]＝{2, 5, 1, 6, 3}
○ 整数型:B[5], i
● i ← 1
i ≦ 5
● B[6－i] ← A[i]
● i ← i ＋ 1

i			[1]	[2]	[3]	[4]	[5]
	B						
	B						
	B						
	B						
	B						
	B						
	B						
	B						
	B						
	B						
	B						

○ プログラム名:PG11－11
○ 整数型:A[6]＝{2, 5, 1, 6, 3, 0}, i
● i ← 1
i ＜ 6
● A[6] ← A[6] ＋ A[i]
● i ← i ＋ 1

i	A[6]					

○ プログラム名:PG11－12(フィボナッチ数列)
○ 整数型:A[15],i
- A[1] ← 0
- A[2] ← 1
- i ← 3
- i ≦ 15
  - A[i] ← A[i－2] ＋ A[i－1]
  - i ← i ＋ 1

○ プログラム名:PG11－13
○ 整数型:A[10]＝{3,5,7,8,4,5,1,2,8,9}
○ 整数型:B[10],i,p
- i ← 1
- i ≦ 10
  - p ← A[i]
  - B[i] ← A[p]
  - i ← i ＋ 1

○ プログラム名:PG11－14
○ 整数型:A[10]＝{6,4,5,9,8,6,7,8,3,4}
○ 整数型:B[10],i
- i ← 1
- B[i] ← A[A[i]]
- i ← i＋1
- i ≦ 10
  - B[i] ←B[i－1] ＋ A[A[i]]
  - i ← i ＋ 1

Section 2　2次元配列とは

2次元配列は[行]と[列]で構成された配列です。
[行][列]に意味を持たせることにより、様々なデータを効率的に管理できます。

①2次元配列の定義

1次元配列と同じように宣言部に定義しますが、[行数][列数]も指定します。

ex） 整数を3行5列で15個格納できる変数をarrayという名前で定義する

○　整数型:array[3][5]

②2次元配列の利用

整数部:array[3][5] と定義した2次元配列は以下のようなイメージとなります。

	[1]	[2]	[3]	[4]	[5]
array [1]					
[2]					
[3]					

この配列arrayに値を代入するには、行番号と列番号の2つを指定しなければなりません。

ex） arrayの要素に値を代入する

- array[1][3] ← 10
- array[2][1] ← 20
- array[3][5] ← array[1][3] ＋ array[2][1]

処理後の2次元配列arrayの中身は以下となります。

	[1]	[2]	[3]	[4]	[5]
array [1]			10		
[2]	20				
[3]					30

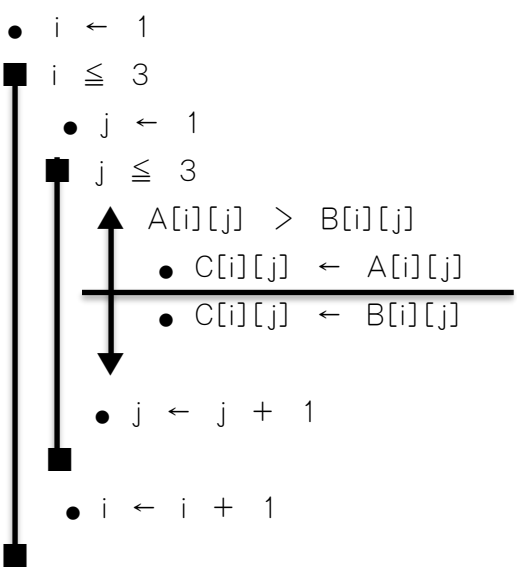
○ プログラム名:PG12－1
○ 整数型:A[5][5], i, j
- i ← 1
- i ≦ 5
  - j ← 1
  - A[i][j] ← 0
  - j ← j ＋ 1
  - A[i][j] ← 0
  - j ← j ＋ 1
  - A[i][j] ← 0
  - j ← j ＋ 1
  - A[i][j] ← 0
  - j ← j ＋ 1
  - A[i][j] ← 0
  - i ← i ＋ 1

○ プログラム名:PG12－2
○ 整数型:A[5][5], i, j
- i ← 1
- i ≦ 5
  - j ← 1
  - j ≦ 5
    - A[i][j] ← i ＊ j
    - j ← j ＋ 1
  - i ← i ＋ 1

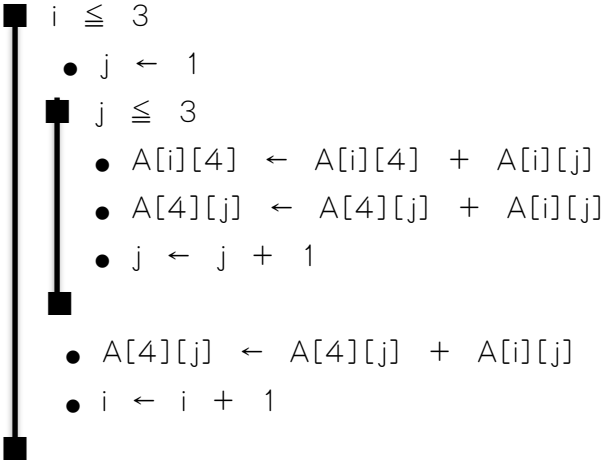
○ プログラム名:PG12－3
○ 整数型:A[3][6]＝{2, 8, 7, 5, 3, 0, 4, 2, 5, 8, 9, 0, 2, 5, 4, 7, 8, 0}
○ 整数型:i, j
● i ← 1
■ i ≦ 3
  ● j ← 1
  ■ j ≦ 5
    ● A[i][6] ← A[i][6] ＋ A[i][j]
    ● j ← j ＋ 1
  ■
  ● i ← i ＋ 1
■

○ プログラム名:PG12－4
○ 整数型:A[3][6]＝{1, 8, 9, 1, 8, 0, 7, 9, 4, 3, 2, 0, 7, 8, 4, 6, 2, 0}
○ 整数型:i, j
● i ← 1
■ i ≦ 5
  ● j ← 1
  ■ j ≦ 3
    ● A[j][6] ← A[j][6] ＋ A[j][i]
    ● j ← j ＋ 1
  ■
  ● i ← i ＋ 1
■

○ プログラム名:PG12－5
○ 整数型:A[3][3]＝{1, 8, 9, 1, 8, 7, 9, 4, 3}
○ 整数型:B[3][3]＝{2, 8, 7, 5, 3, 4, 2, 5, 8}
○ 整数型:C[3][3], i, j



○ プログラム名:PG12－6
○ 整数型:A[4][4]＝{1, 8, 9, 0, 1, 8, 7, 0, 9, 4, 3, 0, 0, 0, 0, 0}
○ 整数型:i, j



○ プログラム名:PG12－7
○ 整数型:A[5][5]＝{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}
○ 整数型:i, j, w
- i ← 1
- i ≦ 5
  - j ← 1
  - j ≦ 5 ／ 2
    - w ← A[i][j]
    - A[i][j] ← A[i][6－j]
    - A[i][6－j] ← w
    - j ← j ＋ 1
  - i ← i ＋ 1

○ プログラム名:PG12－8
○ 整数型:A[5][5]＝{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}
○ 整数型:i, j, w
- i ← 1
- i ≦ 5 ／ 2
  - j ← 1
  - j ≦ 5
    - w ← A[i][j]
    - A[i][j] ← A[6－i][j]
    - A[6－i][j] ← w
    - j ← j ＋ 1
  - i ← i ＋ 1

○ プログラム名:PG12－9
○ 整数型:A[5][5]＝{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}
○ 整数型:i, j, w
- i ← 1
- i ≦ 5 ／ 2
  - j ← i
  - j ＜ 6 － i
    - w ← A[i][j]
    - A[i][j] ← A[j][6－i]
    - A[j][6－i] ← A[6－i][6－j]
    - A[6－i][6－j] ← A[6－j][i]
    - A[6－j][i] ← w
    - j ← j ＋ 1
  - i ← i ＋ 1

○ プログラム名:PG12－10
○ 整数型:A[5][5]＝{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}
○ 整数型:i, j, w
- i ← 1
- i ≦ 5 ／ 2
  - j ← i
  - j ＜ 6 － i
    - w ← A[i][j]
    - A[i][j] ← A[6－j][i]
    - A[6－j][i] ← A[6－i][6－j]
    - A[6－i][6－j] ← A[j][6－i]
    - A[j][6－i] ← w
    - j ← j ＋ 1
  - i ← i ＋ 1

Chapter 7 スタックとキューとリストとトレース

『もう無理』とか泣き言は言うな！！！這ってでもやり遂げろ！！

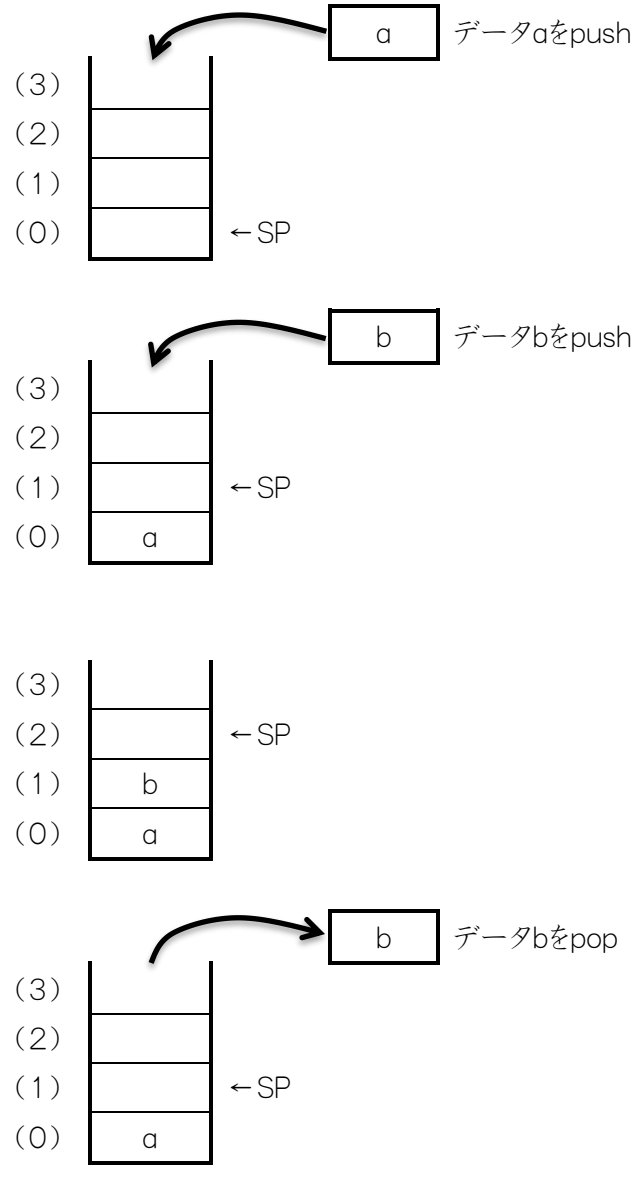
Section 1 スタックとは

後入れ先出し方式のデータ構造。(LIFO:Last In First Out)
データを平積みしていくイメージで、最後に格納したデータが一番上にくる。データを取り出す際はその最後に格納したデータから取り出され、最初に格納したデータは最後に取り出される。

【スタックへの操作と添字】

push:データをスタックに格納する。
pop :データをスタックから取りだす。
SP :スタックポインタ。次のデータ格納位置を示す。

【スタックへの格納と取り出しの流れ】



【プログラムの例】

○ 主プログラム名：スタックサンプル
○ 整数型：SU1，SU2，SU3
○ 外部参照：SP
- SP ← 0
- SU1 ← 10
- SU2 ← 20
- SU3 ← 30
- push(SU1)
- push(SU2)
- push(SU3)
- SU1 ← pop()
- SU2 ← pop()
- SU3 ← pop()

○ 副プログラム名：push(n)
○ 外部参照：Stack[]，SP
- Stack[Sp] ← n
- SP ← SP ＋ 1

○ 副プログラム名：pop
○ 外部参照：Stack[]，SP
- SP ← SP － 1
- Return Stack[SP]

【トレース練習】

○ プログラム名:PG13－1
○ 文字型:Text[]
○ 整数型:i
○ 外部参照:SP
- SP ← 0
- i ← 0
- i ＜ 5
  - push(Text[i])
  - i ← i ＋ 1
- i ← 0
- i ＜ 5
  - Text[i] ← pop()
  - i ← i ＋ 1

○ 副プログラム名:push(n)
○ 外部参照:Stack[], SP
- Stack[SP] ← n
- SP ← SP ＋ 1

○ 副プログラム名:pop
○ 外部参照:Stack[], SP
- SP ← SP － 1
- Return Stack[SP]

	(0)	(1)	(2)	(3)	(4)
Text	A	B	C	D	E

↓のように並びを逆順にする

Text	E	D	C	B	A

Section 2　キューとは

先入れ先出し方式のデータ構造。(FIFO:First In First Out)

キューとは、窓口に並ぶ順番待ちの待ち行列のこと。コンビニのレジのようなイメージであり、最初に格納されたデータから順番に取り出される。

【キューへの操作と添字】

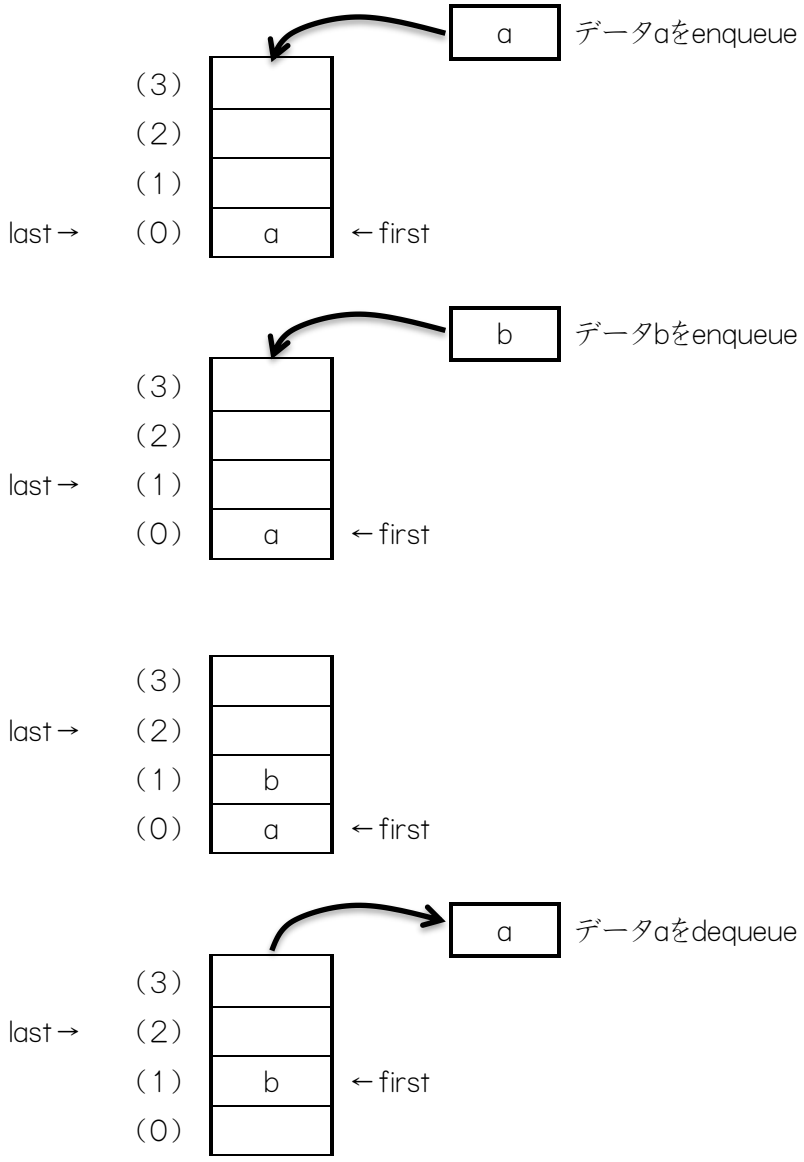
enqueue:データをキューに格納する。

dequeue:データをキューから取りだす。

first　　:キューの先頭。次の取り出し対象のデータ格納位置を示す。

last　　:キューの最後尾。次のデータ格納位置を示す。

【キューへの格納と取り出しの流れ】



【プログラムの例】

○ 主プログラム名：キューサンプル
○ 整数型：SU1，SU2，SU3
○ 外部参照：first，last
- first ← 0
- last ← 0
- SU1 ← 10
- SU2 ← 20
- SU3 ← 30
- enqueue(SU1)
- enqueue(SU2)
- enqueue(SU3)
- SU1 ← dequeue()
- SU2 ← dequeue()
- SU3 ← dequeue()

○ 副プログラム名：enqueue(n)
○ 外部参照：Queue[]，first，last
- Queue[last] ← n
- last ← last ＋ 1

○ 副プログラム名：dequeue
○ 外部参照：Queue[]，first，last
○ 整数型：Data
- Data ← Queue[first]
- first ← first ＋ 1
- Return Data

【トレース練習】

Chapter 8 ソート処理とトレース

ここは国試では一般常識のレベルだ！！！喰らい付いてこい！！！

○ プログラム名:PG14－1(選択ソート)
○ 整数型:A[5]＝{7, 8, 2, 3, 5}
○ 整数型:i, j, w
● i ← 1
■ i ＜ 5
　● j ← i ＋ 1
　■ j ≦ 5
　　▲ A[i] ＞ A[j]
　　● w ← A[i]
　　● A[i] ← A[j]
　　● A[j] ← w
　　▼
　　● j ← j ＋ 1
　■
　● i ← i ＋ 1
■

○ プログラム名:PG14－2(選択ソート)
○ 整数型:A[5]＝{2, 8, 7, 4, 5}
○ 整数型:i, j, w
● i ← 1
■ i ＜ 5
　● j ← i ＋ 1
　■ j ≦ 5
　　▲ A[i] ＜ A[j]
　　● w ← A[i]
　　● A[i] ← A[j]
　　● A[j] ← w
　　▼
　　● j ← j ＋ 1
　■
　● i ← i ＋ 1
■

○ プログラム名:PG14－3(選択ソート)
○ 整数型:A[5]＝{4, 8, 9, 3, 2}
○ 整数型:i, j, w
● i ← 5
■ i ＞ 1
　● j ← i － 1
　■ j ＞ 0
　　▲ A[i] ＞ A[j]
　　● w ← A[i]
　　● A[i] ← A[j]
　　● A[j] ← w
　　▼
　　● j ← j － 1
　■
　● i ← i － 1
■

○ プログラム名:PG14－4(選択ソート)
○ 整数型:A[5]＝{1, 8, 0, 9, 6}
○ 整数型:i, j, w, min
● i ← 1
■ i ＜ 5
　● min ← i
　● j ← i ＋ 1
　■ j ≦ 5
　　▲ A[min] ＞ A[j]
　　● min ← j
　　▼
　　● j ← j ＋ 1
　■
　▲ i ≠ min
　● w ← A[i]
　● A[i] ← A[min]
　● A[min] ← w
　▼
　● i ← i ＋ 1
■

○ プログラム名:PG14－5(バブルソート)
○ 整数型:A[5]＝{2, 0, 7, 8, 5}
○ 整数型:i, j, w
● i ← 1
■ i ＜ 5
　● j ← 1
　■ j ≦ 5 － i
　　▲ A[j] ＞ A[j+1]
　　● w ← A[j]
　　● A[j] ← A[j+1]
　　● A[j+1] ← w
　　▼
　　● j ← j ＋ 1
　■
　● i ← i ＋ 1
■

○ プログラム名:PG14－6(バブルソート)
○ 整数型:A[5]＝{9, 2, 3, 7, 8}
○ 整数型:i, j, w, sw
● sw ← 0
● i ← 1
■ i ＜ 5 and sw ＝ 0
　● j ← 1
　● sw ← 1
　■ j ≦ 5 － i
　　▲ A[j] ＞ A[j+1]
　　● w ← A[j]
　　● A[j] ← A[j+1]
　　● A[j+1] ← w
　　● sw ← 0
　　▼
　　● j ← j ＋ 1
　■
　● i ← i ＋ 1
■

○ プログラム名:PG14－7(バブルソート)
○ 整数型:A[5]＝{6, 3, 8, 4, 7}
○ 整数型:i, j, w
● i ← 2
■ i ≦ 5
　● j ← 5
　■ j ≧ i
　　▲ A[j] ＞ A[j－1]
　　● w ← A[j]
　　● A[j] ← A[j－1]
　　● A[j－1] ← w
　　▼
　　● j ← j － 1
　■
　● i ← i ＋ 1
■

○ プログラム名:PG14－8(挿入ソート)
○ 整数型:A[5]＝{5, 7, 4, 8, 2}
○ 整数型:i, j, w
● i ← 2
■ i ≦ 5
　● j ← i
　■ j ≧ 2 and A[j] ＜ A[j－1]
　　● w ← A[j]
　　● A[j] ← A[j－1]
　　● A[j－1] ← w
　　● j ← j － 1
　■
　● i ← i ＋ 1
■

○ プログラム名:PG14－9(挿入ソート)
○ 整数型:A[5]＝{4, 7, 5, 2, 8}
○ 整数型:i, j, tmp
- i ← 2
- i ≦ 5
  - j ← i
  - tmp ← A[i]
  - j ≧ 2 and tmp ＜ A[j－1]
    - A[j] ← A[j－1]
    - j ← j － 1
  - A[j] ← tmp
  - i ← i ＋ 1

○ プログラム名:PG14－10(挿入ソート)
○ 整数型:A[5]＝{2, 7, 8, 5, 4}
○ 整数型:i, j, tmp
- i ← 2
- i ≦ 5
  - j ← i
  - tmp ← A[i]
  - j ≧ 2 and tmp ＜ A[j－1]
    - A[j] ← A[j－1]
    - j ← j － 1
  - A[j] ← tmp
  - i ← i ＋ 1

編著者

KCS 福岡情報専門学校

吉田　亨

家永　和也

情報処理技術者試験対策講座

Algorithm　BootCamp

2015 年　9 月　11 日　　ver.1.0

2015 年 10 月　5 日　　ver.1.1

2016 年 2 月　29 日　　ver.1.2

2016 年 4 月　10 日　　ver.1.3

2019 年 7 月　24 日　　ver.1.4