2025 年度入試版

擬似プログラミング言語による記述例と、Python による等価な実装例

京都産業大学の入学試験「情報プラス」方式では、プログラミングに関する問題および解答において、独自の記法を用いてプログラムを記述します。ここではその（疑似言語による）プログラムの記述例をいくつか示します。

また、この擬似言語自体は動作させることを前提にはしていませんが、いくつかのプログラミング言語でほぼ等価な記述を行なって動作を確認することが可能です。ここでは Python 言語で記述した例と、その実行結果を示します。

例題 1　1 から 10 までの整数の合計値を示すプログラム

はじめに合計値を保持する変数 sum をゼロで用意し、変数 i を 1 から 10 まで変化させながら合計に加えます。

疑似言語による
記述例

```
var sum = 0
var i
for i = 1 to 10
    sum = sum + i
end
print("合計は: ", sum)
```

プログラム 1-1．1 から 10 の合計（擬似言語）

ポイント：

- 変数の前にある「var」は「宣言」と呼ばれるものです。変数を使う場合は最初にこの宣言作業が必要です。C, Go, Rust など幾つかのプログラミング言語でも変数宣言があります。
- sum のように宣言と同時に初期値を設定することができます。変数 i のように初期値の設定が不要でも宣言は必要です。
- 擬似言語の for 文では、範囲の最後の数（この例では 10）までを繰り返しの値とします。

Python による
記述例と実行結果

```
sum = 0
for i in range(1, 11):
    sum = sum + i
print("合計は: ", sum)
```

```
合計は:  55
```

プログラム 1-2．1 から 10 の合計（Python）とその実行結果

ポイント：

- Python の range 関数を用いた for 文では、合計する値に 10 を含めるためには第二引数に 11 を指定することになります。

例題 2　1 文字違いを判定するプログラム

本学情報理工学部の 2023 年度 AO 入試 1 次選考問題の IV 問題の設問(A)で示されたフローチャートを題材にしてみます。つまり文字数が同じ二つの文字列を比較して、異なる文字数を数え、1 文字違いのときだけ「1 文字違い」と表示する、というものです。数え方については図 2-1 あるいは元の問題(*1)を参照してください。

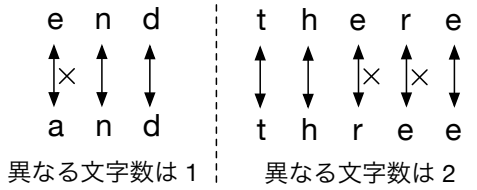
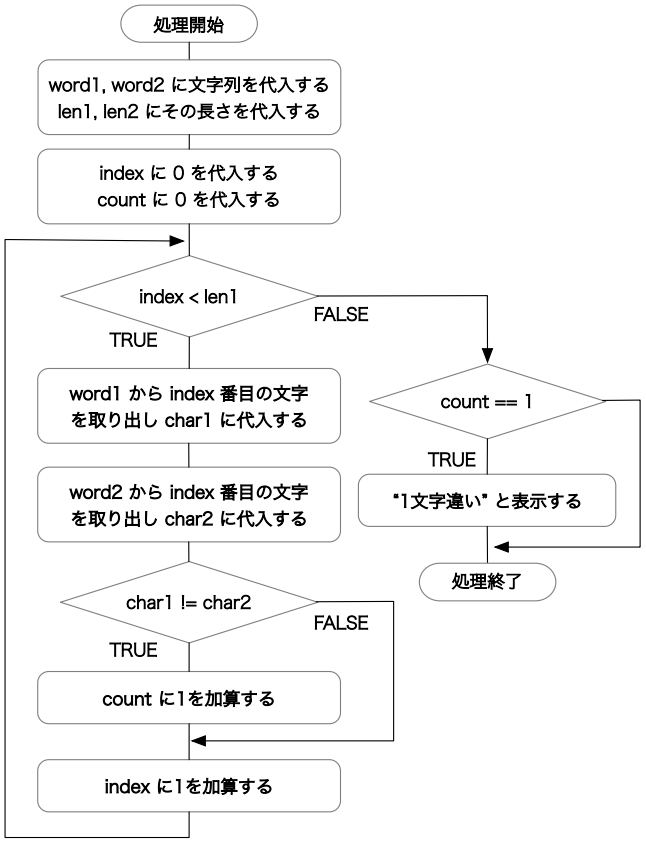


図 2-1. 異なる文字数を数えた例

図 2-2 にフローチャートを、プログラム 2-1 に等価な振る舞いをするプログラム例(疑似言語)を示します。



疑似言語による記述例

```
var word1[]={"e","n","d"}
var word2[]={"a","n","d"}
var len1 = 3, len2 = 3
var index = 0
var count = 0
var char1, char2
while index < len1
    char1 = word1[index]
    char2 = word2[index]
    if char1 != char2
        count = count + 1
    end
    index = index + 1
end
if count == 1
    print("1 文字違い")
end
```

図 2-2. 異なる文字数を数えるフローチャート　プログラム 2-1. 異なる文字数を数える(擬似言語)

ポイント:

- var によって配列変数を宣言する際に、各要素に初期値を設定することができます。
- 配列の要素番号は 0 から始まります。つまりプログラム 2-1 では、word1[0]に"e"、word1[1]に"n"、word1[2]に"d"が格納されています。

*1 元の問題文 https://www.kyoto-su.ac.jp/admissions/exam/pickup/ise_point/index.html

Python による記述例と実行結果

```
word1 = ["e","n","d"]
word2 = ["a","n","d"]
len1 = 3
len2 = 3
index = 0
count = 0
while index < len1:
    char1 = word1[index]
    char2 = word2[index]
    if char1 != char2:
        count = count + 1
    index = index + 1
if count == 1:
    print("1 文字違い")
```

```
1 文字違い
```

プログラム 2-2．異なる文字数を数える （Python）とその実行結果

注意：

- 疑似言語での記述例と揃えるために 1-4 行目をこのようにしましたが、Python に慣れた人は `word1 = "end"` や `len1 = len(word1)` のように修正した方が、アルゴリズムが読みとりやすくなるかもしれません。
- このプログラムは 1 文字違い以外の場合、なにも出力せず終了します。一般的には何かメッセージを出力する方が自然でしょうが、元の問題では提示するフローチャートを極力簡潔にするためにそのあたりは削っています。

例題３　ロボットの位置を操作するプログラム

横＝M、縦＝N の長方形領域の中で動くロボットの位置と向きを操作する処理を考えます。ロボットの位置は座標（x, y）で表し、x, y は整数とします（ 0 ≦ x ＜ M, 0 ≦ y ＜ N ）。この領域の周囲には壁があり、ロボットはこの壁の中でしか動くことができません。

ロボットの向き（前進する方向）は4方向で、x 軸正または負方向、y 軸正または負方向のいずれかであり、これらを次の値で表します。

0: x 軸正方向、 1: y 軸正方向、 2: x 軸負方向、 3: y 軸負方向

図 3-1 に、M=6、N=5 の長方形領域で、ロボットが位置（2, 1）で x 軸正方向を向いている状況を示します。

プログラムはコントローラからの指示（コマンド）にしたがって、ロボットの位置または向きを操作します。コマンドは整数値で表され、以下のような動作に対応します。

0: 動作停止、 1: 前進、 2: 後退、 3: 左へ 90°回転、 4: 右へ 90°回転

前進および後退コマンドでは距離 1 だけ移動するのですが、移動しようとした先が領域の外で、壁にぶつかる場合は移動せずに警告メッセージを表示します。

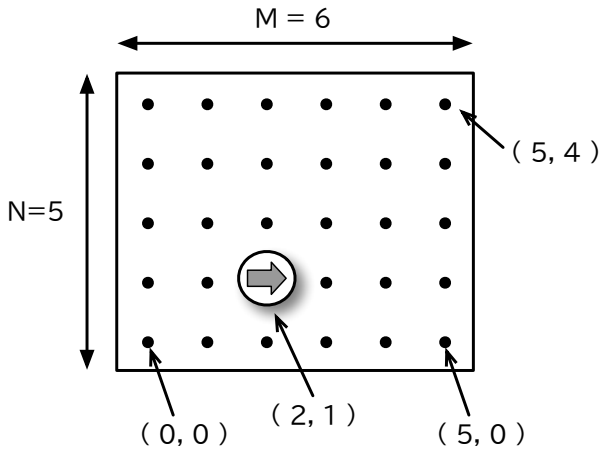


図 3-1. ロボットが動作する長方形領域

プログラム 3-1 は、このようなロボットを操作するプログラムをシミュレートするもので、コマンドはあらかじめ配列に入れておき、移動した場合に新しい座標を表示します。

ロボットは最初に（0, 0）の位置にあり、x 軸正方向を向いているものとします。

疑似言語による記述例

```
var commands[] = { 1, 1, 1, 3, 1, 1, 0 } # コマンド列
var cmd, index = 0  # コマンドを1つずつ取り出すための添え字
var x = 0, y = 0    # ロボットの現在の座標
var direction = 0   # ロボットの現在の向き

cmd = commands[index]          # 先頭のコマンドを取り出す
while cmd != 0
    if cmd == 1 OR cmd == 2    # 前進か後退
        var move = 1
        if cmd == 2
            move = -1          # 後退
        end
        var newx = x           # 次の座標
        var newy = y
        if direction == 0      # x軸正方向
            newx = newx + move
        elif direction == 1    # y軸正方向
            newy = newy + move
        elif direction == 2    # x軸負方向
            newx = newx - move
        else                   # y軸負方向
            newy = newy - move
        end
        if newx >= 0 AND newx < M AND newy >= 0 AND newy < N
            x = newx           # 領域内だったら新しい座標に更新する
            y = newy
            print("(", x, ", ", y, ")")  # 座標を表示する
        else
            print("壁に衝突")    # 警告メッセージを表示
        end
    else
        if cmd == 3            # 左へ90°回転
            direction = (direction + 1) % 4
        else                   # 右へ90°回転
            direction = (direction + 3) % 4
        end
    end
    index = index + 1
    cmd = commands[index]      # 次のコマンドを取り出す
end
print("停止")
```

プログラム 3-1．ロボットの制御プログラム(擬似言語)

Python による記述例

```
commands = [ 1, 1, 1, 3, 1, 1, 0 ]
index = 0           # コマンドを 1 つずつ取り出すための添え字
M = 6
N = 5
x = 0
y = 0               # ロボットの現在の座標 (x, y)
direction = 0       # ロボットの現在の向き

cmd = commands[index]          # 先頭のコマンドを取り出す
while cmd != 0:
    if cmd == 1 or cmd == 2:   # 前進か後退
        move = 1
        if cmd == 2:
            move = -1          # 後退
        newx = x
        newy = y
        if direction == 0:     # x 軸正方向
            newx = newx + move
        elif direction == 1:   # y 軸正方向
            newy = newy + move
        elif direction == 2:   # x 軸負方向
            newx = newx - move
        else:                  # y 軸負方向
            newy = newy - move
        if newx >= 0 and newx < M and newy >= 0 and newy < N:
            x = newx           # 領域内だったら新しい座標に更新する
            y = newy
            print("(", x, ",", y, ")")  # 座標を表示する
        else:
            print("壁に衝突")    # 警告メッセージを表示する
    else:
        if cmd == 3:           # 左へ 90°回転
            direction = (direction + 1) % 4
        else:                  # 右へ 90°回転
            direction = (direction + 3) % 4
    index = index + 1
    cmd = commands[index]      # 次のコマンドを取り出す
print("停止")
```

プログラム 3-2．ロボットの制御プログラム（Python）

ポイント:

- プログラム 3-1, 3-2 ではコマンド列として「 1, 1, 1, 3, 1, 1, 0 」を設定していますが、これは「前進、前進、前進、左へ 90°回転、前進、前進、停止」を意味します。
- ロボットは最初（0, 0）の位置にあり、x 軸正方向を向いています。
- このコマンド列によって図 3-2 の実行結果が示すように移動し、最終的に（3, 2）の位置で停止することを、まず手作業で(図の上で)確認してからプログラムを読むと良いでしょう。

```
( 1 , 0 )
( 2 , 0 )
( 3 , 0 )
( 3 , 1 )
( 3 , 2 )
停止
```

図 3-2. プログラム 3-2 の実行結果(1)

変数 commands に与えるコマンド列を変更することによって、ロボットの動作を変更できます。

図 3-3 は(プログラム 3-2 を修正して）[1, 3, 1, 4, 1, 3, 1, 4, 1, 3, 1, 4, 1, 0] というコマンド列で実行した場合の結果です。

図 3-4 は同様に [3, 1, 1, 4, 1, 1, 3, 2, 2, 2, 4, 1, 0] というコマンド列で実行した場合の結果で、壁に衝突していることがわかります。

```
( 1 , 0 )
( 1 , 1 )
( 2 , 1 )
( 2 , 2 )
( 3 , 2 )
( 3 , 3 )
( 4 , 3 )
停止
```

図 3-3. プログラム 3-2 の実行結果(2)

```
( 0 , 1 )
( 0 , 2 )
( 1 , 2 )
( 2 , 2 )
( 2 , 1 )
( 2 , 0 )
壁に衝突
( 3 , 0 )
停止
```

図 3-4. プログラム 3-2 の実行結果(3)