共通テスト手順記述標準言語 (DNCL) の説明

独立行政法人大学入試センター

2022 年 1 月

高等学校におけるアルゴリズムやプログラムに関する教育では，採用されるプログラミング言語は多様で，プログラミングの実習時間も異なります。大学入試センターではこのような事情を考慮し，「情報関係基礎」の出題にあたり，共通テスト用の手順記述言語 (DNCL) を使用します。

以下，参考のために DNCL の基本を説明します。しかしながら，問題文の記述を簡潔にするなどの理由で，この説明文書の記述内容に従わない形式で出題することもあります。したがって，「情報関係基礎」の受験に際しては，当該問題文の中の説明や指示に注意し，それらに沿って解答してください。

目次

1 **変数と値** 2

2 **表示文** 2

3 **代入文** 3

4 **演算** 4
- 4.1 算術演算 . . . 4
- 4.2 比較演算 . . . 4
- 4.3 論理演算 . . . 5

5 **制御文** 6
- 5.1 条件分岐文 . . . 6
- 5.2 条件繰返し文 . . . 8
- 5.3 順次繰返し文 . . . 9

6 **用意された関数の呼び出し** 10
- 6.1 値を返す関数 . . . 10
- 6.2 値を返さない関数 . . . 10

7 **新しい関数の定義** 11

1　変数と値

変数名は，英字で始まる英数字と『_』の並びです。

例: kosu，kosu_gokei，Tokuten

特に指示がない限り，小文字で始まる変数は通常の変数を表し，大文字で始まる変数は配列を表します。また，すべて大文字の変数は実行中に変化しない値を表します。

配列の要素は，要素の番号を添字で指定します。2 次元以上の場合は，添字を『，』で区切ります。たとえば，(1 次元の) 配列 Tokuten や 2 次元配列 Gyoretu の要素は Tokuten[2] や Gyoretu[3, 2] のように表します。添字の値は 0 以上の整数ですが，問題によっては 1 以上の添字のみを扱います。

特に断らない限り，数値は 10 進法で表します。文字列は，文字の並びを 『「 』と『 」』，または，『"』と『"』でくくって表します。

例:　100
例:　99.999
例:「見つかりました」
例:　"It was found."

2　表示文

表示文で数値や文字列や変数の値を表示します。表示文では，複数の値を表示する場合は『と』で区切って並べ，最後に『を表示する』と書きます。

例:「整いました」を表示する　　(「整いました」と表示されます。)
例: kosu と「個見つかった」を表示する　　(kosu が 3 のとき，「3 個見つかった」と表示されます。)
例: "(" と x と ", " と y と ")" を表示する　　(x が 5，y が -1 のとき，「(5, -1)」と表示されます。)

3　代入文

代入文は変数に値を設定します。『←』の左辺に変数または添字付きの配列を，右辺に代入する値を書きます。また，配列の各要素に同じ値をまとめて代入することや，他の配列の内容に置き換えることもできます。

例: kosu ← 3
例: Tokuten[4] ← 100
例: Tokuten のすべての要素に 0 を代入する
例: Tokuten ← {87，45，72，100}

複数の代入文を，『，』で区切りながら，横に並べることができます。この場合は，代入文は左から順に実行されます。

例: kosu_gokei ← kosu，tokuten ← kosu × (kosu + 1)

同じ変数に対する加算や減算を伴う代入（インクリメントやデクリメント）は，『～を～増やす』や『～を～減らす』によって表すこともできます。

例:『kosu を 1 増やす』　　は『kosu ← kosu + 1』　　と同じです。
例:『saihu を syuppi 減らす』は『saihu ← saihu − syuppi』と同じです。

外部から入力された値を代入するために，次のように記述することもあります。

例: x ←【外部からの入力】

4 演算

この節では，算術演算と比較演算，そして論理演算について説明します。比較演算やそれを組み合わせる論理演算は，条件分岐文（5.1 節）や条件繰返し文（5.2 節）の **〈条件〉** で使うことができます。

4.1 算術演算

加減乗除の四則演算は，『＋』，『－』，『×』，『 / 』で指定します。
整数の除算では，商を『÷』で，余りを『%』で計算することができます。

例: atai ← 7 / 2　　（atai には 3.5 が代入されます。）
　　syo ← 7÷ 2　　（syo には 3 が代入されます。）
　　amari ← 10% 3　（amari には 1 が代入されます。）

複数の演算子を使った式の計算では，基本的に左側の演算子が先に計算されますが，『×』，『 / 』，『÷』，『%』は，『＋』，『－』より先に計算されます。また，丸括弧『(』と『)』で式をくくって，演算の順序を明示することができます。

例: sogaku ← ne1－ ne2－ ne3　　は，
　　sogaku ← (ne1－ ne2)－ ne3　と同じです。
例: kosu ← 1＋ kazu÷ 3　　は，
　　kosu ← 1＋ (kazu÷ 3)　　と同じです。
例: heikin ← (hidari＋ migi)÷ 2 は，
　　heikin ← hidari＋ migi÷ 2　と異なります。

4.2 比較演算

数値の比較演算は，『＝』，『≠』（あるいは『≒』），『＞』，『≧』，『≦』，『＜』で指定します。演算結果は，真か偽の値となります。

例:　kosu ＞ 3　　（kosu が 3 より大きければ真となります。）
例:　ninzu× 2 ≦ 8　　（ninzu の 2 倍が 8 以下であれば真となります。）
例:　kaisu ≠ 0　　（kaisu が 0 でなければ真となります。）

文字列の比較演算は，『＝』，『≠』（あるいは『≒』）を利用することができます。『＝』は，左辺と右辺が同じ文字列の場合に真となり，それ以外の場合は偽となります。『≠』（あるいは『≒』）は，左辺と右辺が異なる文字列の場合に真となり，それ以外の場合（同じ文字列の場合）は偽となります。

例: 「あいうえお」＝「あいうえお」　　（真となります。）
例: 「あいうえお」＝「あいう」　　（偽となります。）
例:　"ABC"＝"ABC"　　（真となります。）
例:　"ABC"＝"abc"　　（偽となります。）
例: 「あいうえお」≠「あいうえお」　　（偽となります。）

例: 「あいうえお」≠「あいう」　　(真となります。)
例: "ABC"≠"ABC"　　(偽となります。)
例: "ABC"≠"abc"　　(真となります。)

4.3 論理演算

論理演算は，真か偽を返す式に対する演算で，『かつ』，『または』，『でない』の演算子で指定します。論理演算子に優先順位はなく，左側の論理演算が先に実行されますが，丸括弧『(』と『)』で，演算の順序を指定することができます。

『**〈式 1〉** かつ **〈式 2〉**』は，**〈式 1〉** と **〈式 2〉** の結果がいずれも真である場合に真となり，それ以外の場合は偽となります。『**〈式 1〉** または **〈式 2〉**』は，**〈式 1〉** と **〈式 2〉** の結果のどちらかが真である場合に真となり，それ以外の場合は偽となります。『**〈式〉** でない』は，**〈式〉** の結果が真である場合に偽となり，偽の場合は真となります。

例: kosu ≧ 12 かつ kosu ≦ 27　　(kosu が 12 以上 27 以下なら真となります。)
例: kosu**%** 2 ＝ 0 または kosu ＜ 0　　(kosu が偶数か負の値なら真となります。)
例: kosu ＞ 75 でない　　(kosu が 75 より大きくなければ真となります。)
例: kosu ＞ 12 かつ kosu ＜ 27 でない　　は,
(kosu ＞ 12 かつ kosu ＜ 27) でない　　と同じです。(左側の論理演算子が先に実行されるため。)
例: kosu ＞ 12 かつ kosu ＜ 27 でない　　は,
kosu ＞ 12 かつ (kosu ＜ 27 でない)　　と異なります。

5　制御文

条件分岐文（5.1 節）や条件繰返し文（5.2 節），順次繰返し文（5.3 節）をまとめて制御文と呼びます。制御文の中の **〈処理〉** として，表示文（2 節），代入文（3 節），値を返さない関数（6.2 節），条件分岐文，順次繰返し文，条件繰返し文を，一つ以上並べて使うことができます。また，条件分岐文や条件繰返し文の中の **〈条件〉** として，比較演算（4.2 節）と論理演算（4.3 節）を使用することができます。

5.1　条件分岐文

条件分岐文は，**〈条件〉** が成り立つかどうかによって，実行する処理を切り替えます。

〈条件〉 が成り立つときにある処理を実行し，**〈条件〉** が成り立たないときに実行する処理がない場合は，次のように『ならば』で指定します。

《一般形》

もし **〈条件〉** ならば
│　**〈処理〉**
を実行する

例:　もし x ＜ 3 ならば
│　x ← x＋ 1
│　y ← y− 1
を実行する

〈処理〉 が 1 行しかない場合は，次のように全体を 1 行で書くこともできます。

《一般形》

もし **〈条件〉** ならば **〈処理〉** を実行する

例:　もし x ＜ 3 ならば x ← x＋ 1 を実行する

〈条件〉 が成り立つときにある処理を実行し，**〈条件〉** が成り立たないときに別の処理を実行する場合は，次のように『ならば』と『そうでなければ』を組み合わせて指定します。

《一般形》

もし **〈条件〉** ならば
│　**〈処理 1〉**
を実行し，そうでなければ
│　**〈処理 2〉**
を実行する

例: もし x ＜ 3 ならば
　　x ← x＋ 1
を実行し，そうでなければ
　　x ← x− 1
を実行する

改行位置によって実行結果が変わらないため，各処理が 1 行で書ける場合には，次のように書くこともあります。

《一般形》

もし **〈条件〉** ならば **〈処理 1〉** を実行し，
そうでなければ **〈処理 2〉** を実行する

例: もし x ＜ 3 ならば x ← x＋ 1 を実行し，
そうでなければ x ← x− 1 を実行する

条件分岐の中で複数の条件で実行する処理を切り替えたい場合は，次のように『ならば』と『そうでなければ』の間に『そうでなくもし』を使って条件を追加します。

《一般形》

もし **〈条件 1〉** ならば
　　〈処理 1〉
を実行し，そうでなくもし **〈条件 2〉** ならば
　　〈処理 2〉
を実行し，そうでなければ
　　〈処理 3〉
を実行する

例: もし x ＝ 3 ならば
　　x ← x＋ 1
を実行し，そうでなくもし y ＞ 2 ならば
　　y ← y＋ 1
を実行し，そうでなければ
　　y ← y− 1
を実行する

改行位置によって実行結果が変わらないため，各処理が 1 行で書ける場合には，次のように書くこともあります。

《一般形》

もし **〈条件 1〉** ならば **〈処理 1〉** を実行し，
そうでなくもし **〈条件 2〉** ならば **〈処理 2〉** を実行し，
そうでなければ **〈処理 3〉** を実行する

例: もし x ＝ 3 ならば x ← x＋ 1 を実行し，
そうでなくもし y ＞ 2 ならば y ← y＋ 1 を実行し，
そうでなければ y ← y− 1 を実行する

5.2 条件繰返し文

条件繰返し文には，「前判定」と「後判定」の 2 種類があります。

5.2.1 前判定

〈条件〉 が成り立つ間，**〈処理〉** を繰り返し実行します。

〈処理〉 を実行する前に **〈条件〉** が成り立つかどうか判定されるため，**〈処理〉** が 1 回も実行されないことがあります。

《一般形》

〈条件〉 の間，
　〈処理〉
を繰り返す

例: x ＜ 10 の間，
　gokei ← gokei＋ x
　x ← x＋ 1
を繰り返す

5.2.2 後判定

〈条件〉 が成り立つまで，**〈処理〉** を繰り返し実行します。

〈処理〉 を実行した後に **〈条件〉** が成り立つかどうか判定されるため，**〈処理〉** は少なくとも 1 回は実行されます。

《一般形》

繰り返し，
　〈処理〉
を，**〈条件〉** になるまで実行する

例: 繰り返し，
```
    gokei ← gokei＋ x
    x ← x＋ 1
```
を，x ≧ 10 になるまで実行する

5.3 順次繰返し文

順次繰返し文は，**〈変数〉** の値を増やしながら，**〈処理〉** を繰返し実行します。

《一般形》

〈変数〉 を **〈初期値〉** から **〈終了値〉** まで **〈差分〉** ずつ増やしながら，
```
    〈処理〉
```
を繰り返す

順次繰り返し文は，以下の手順で実行されます。

1. **〈変数〉** に **〈初期値〉** が代入されます。
2. **〈変数〉** の値が **〈終了値〉** よりも大きければ，繰り返しを終了します。
3. **〈処理〉** を実行し，**〈変数〉** の値に **〈差分〉** を加え，手順 2 に戻ります。

例: x を 1 から 10 まで 1 ずつ増やしながら，
```
    gokei ← gokei＋ x
```
を繰り返す

『増やしながら』を『減らしながら』にすると，**〈変数〉** の値を **〈初期値〉** から **〈差分〉** ずつ減らしながら，その値が **〈終了値〉** よりも小さくなるまで，**〈処理〉** を繰り返し実行します。

例: x を 10 から 1 まで 1 ずつ減らしながら，
```
    gokei ← gokei＋ x
```
を繰り返す

6　用意された関数の呼び出し

あらかじめ用意された関数には，値を返すものと値を返さないものがあります。関数の動作は，問題文の中で定義されます。

6.1　値を返す関数

問題文の中で

- 指定された値の二乗の値を返す関数「**二乗**」を用意する
- 値 m の n 乗の値を返す関数「**べき乗**(m, n)」を用意する
- 値 m 以上値 n 以下の整数をランダムに一つ返す関数「**乱数**(m, n)」を用意する
- 値 n が奇数のとき真を返し，そうでないとき偽を返す関数「**奇数**(n)」を用意する

のように定義された関数を，表示文 (2 節)，代入文 (3 節)，算術演算（4.1 節），比較演算（4.2 節），あるいは論理演算（4.3 節）の中で使うことができます。関数を呼び出すときは，関数名に続き，『(』と『)』の間に引数を書きます。複数の引数を指定する場合は，『, 』で区切ります。

例:　y ← **二乗**(x)　　(y に x の二乗が代入されます。)
例:　z ← **二乗**(x) ＋ **べき乗**(x, y)　　(z に x の二乗と x の y 乗の和が代入されます。)
例:　r ← **乱数**(1, 6)　　(r に 1 から 6 までの整数のうちいずれかが代入されます。)

6.2　値を返さない関数

問題文の中で

- 指定された値を 2 進表現で表示する関数「**二進で表示する**」を用意する

のように値を返さない関数が定義されることがあります。

例:　**二進で表示する**(11)　　(「1011」と表示されます。)

7　新しい関数の定義

新しい関数の定義は，DNCL を用いて次のように記述します。

《一般形》

関数 **〈関数名〉**（ **〈引数列〉** ）を
　〈処理〉
と定義する

関数が呼び出される時に引数として与えられる値は、引数列のところに書いた変数名で利用します。複数の引数を指定する場合は，『，』で区切ります。定義した関数は，用意された関数の呼び出し（6 節）と同じ記法で呼び出すことができます。

例:　1 から正の整数 n までの和を表示する関数「**和を表示する**（n）」の定義例

```
関数 和を表示する（n）を
    wa ← 0
    i を 1 から n まで 1 ずつ増やしながら，
        wa ← wa＋ i
    を繰り返す
    wa を表示する
と定義する
```

例:　値 m の n 乗の値を表示する関数「**べき乗を表示する**（m，n）」の定義例

```
関数 べき乗を表示する（m，n）を
    p ← 1
    i を 1 から n まで 1 ずつ増やしながら，
        p ← p× m
    を繰り返す
    p を表示する
と定義する
```