センター試験用手順記述標準言語 (DNCL) の説明

独立行政法人大学入試センター

2011 年 1 月

目次

はじめに

高等学校でのアルゴリズムやプログラムに関する教育で採用されているプログラミング言語が多様なことや，学科によってプログラミングの実習の量が異なることを考慮して，大学入試センターでは，「情報関係基礎」の出題に当たっては，従前から既存のプログラミング言語を用いずに，「センター試験用手順記述標準言語」(DNCL) という独自のプログラミング言語を使用しています。

以下に参考のために，DNCL の必要最低限の説明文を掲載します。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や，この説明文書そのものが変更されることもあります。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。

1　変数と定数

変数名は，英字で始まる英数字と『_』の並びです。

例: kosu，　Tokuten，　kosu_gokei

配列の要素は，要素の番号を添字で指定します。二次元以上の場合は，添字を『,』で区切ります。特に断らない限り試験問題では，配列変数は先頭文字を大文字で開始し，通常の変数は小文字で開始します。

例: Tokuten[2]，　Tokuten[1,2]（配列変数）
　　tokuten，　ninzu　　（通常の変数）

変数は，値の代入または配列要素の値の設定により初期化してから使います。

例: kosu ← 0
例: Tokuten のすべての値を 0 にする

文字列は，値を『「』と『」』，または，『"』と『"』で括って表します。

例:「見つかりました」を表示する
例: "it was found." を表示する

2　代入文

代入文は変数に値を代入します。代入文は，6 節で説明する**〈処理〉**で使うことができます。『←』の左辺に変数を，右辺に代入する値を書きます。

例: kosu ← 3

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。

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』，『～を～減らす』で，指定することもできます。

例:『kosu を 1 増やす』　　と『kosu ← kosu ＋ 1』　　は同じ働き
例:『saihu を shuppi 減らす』と『saihu ← saihu−shuppi』は同じ働き

3　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，6.2 節や 6.3 節の**〈条件〉**の箇所で用います。

3.1 算術演算

加減乗除の四則演算は，『+』，『−』，『×』，『/』 (商) で指定します。
整数の除算では，商の整数部分を『÷』で，余りを『%』(剰余) で指定することができます.

例: sho ← 8 ÷ 3　　(sho には 2 が代入されます.)
　　amari ← 10 % 3　(amari には 1 が代入されます.)

計算では，『×』，『/』，『÷』，『%』 は，『+』，『−』より先に計算されます。このことを『×，/，÷，% は，+，− より優先順位が高い。』と言います。優先順位が等しい場合は，左側の演算が先に実行されます。また，丸括弧『(』と『)』で式を括って，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。
例: kosu ← 1 + kazu / 3　　は，
　　kosu ← 1 + (kazu / 3)　　と同じ。
例: heikin ← (migi + hidari) / 2　と，
　　heikin ← migi + hidari / 2　　は異なる。

3.2 比較演算

比較演算は，『=』，『≒』(あるいは『≠』)，『>』，『≧』，『≦』，『<』で指定します。

例: kosu > 3
例: (kosu + 1) ≒ 3

3.3 論理演算

論理演算は，比較演算の結果，あるいは問題文の中で定義された関数（4.1 節）が返した結果に対する演算で，『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。
『かつ』は，左辺と右辺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。『または』は，左辺と右辺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。『でない』は，左辺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 ≦ 3 かつ kosu ≧ 10　　(これを満たす kosu の値は存在しません.)
例: kosu < 3 または kosu = 3　　(これは kosu ≦ 3 と同じことです.)
例: kosu > 3 かつ kosu < 10 でない　　と，
　　(kosu > 3 かつ kosu < 10) でない　は同じです.

4　関数

4.1　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。

のように定義された関数を，算術演算（3.1 節），比較演算（3.2 節），あるいは論理演算（3.3 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『)』の間にパラメータ（引数）を書きます。複数のパラメータを書く場合は，『 , 』で区切ります。

> 例: y ← 二乗 (x)
> 例: z ← 二乗 (x) ＋ べき乗 (x，y)

4.2　値を返さない関数

同様に，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」を用意する。

のように定義された，値を返さない関数を 6 節で説明する**〈処理〉**で使うことができます。

> 例: 二進で表示 (x)

5　表示文

表示文で変数や定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: kosu と「個見つかった」と "found" を表示する

6　制御文

6.1　〈処理〉や 〈条件〉として指定することができるもの

この節で説明する，条件分岐文 や 順次繰返し文，条件繰返し文 の中の**〈処理〉**として，代入文 と 表示文，値を返さない関数，条件分岐文，順次繰返し文，条件繰返し文 を，1 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（3.2 節） と 論理演算（3.3 節） を使用することができます。

6.2 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〉**が成り立つときに限り処理を実行する場合は，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
```

処理を 1 行で書ける場合は，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〉**

例:

```
もし x ＜ 3 ならば x ← x ＋ 1
```

条件が成り立つときに実行する処理のほかに，条件が成り立たないときに実行する処理を指定する場合は，『そうでなければ』で指定します。条件が成り立たないときに他の条件を指定する場合は，『そうでなくもし』で新しい条件を指定します。

《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，そうでなければ
　x ← x − 1
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

6.3 条件繰返し文

条件繰返し文は，**〈条件〉**が成り立つ間，**〈処理〉**を繰り返し実行します。**〈条件〉**は処理を実行する前に調べられるため，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

6.4 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 **〈増分〉** ずつ 増やしながら，
　〈処理〉
を繰り返す

- **〈変数〉**に**〈初期値〉**が代入されます。
- **〈処理〉**を実行する前に**〈変数〉**の値と**〈終了値〉**の値を比較して，**〈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。
- **〈処理〉**を実行した後に，**〈変数〉**の値に**〈増分〉**が加えられます。

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〉**から減らしながら，その値が**〈終了値〉**以上である間，**〈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