

Duskul言語の説明

Version 1.3.0 (2025.05.22) 萩原剛志

1. はじめに

Duskul は、プロジェクト演習用に設計された、極めて小規模な手続き型プログラミング言語です。Pascal風の構文を持ち、再帰呼び出しが可能な手続きや関数（これらをサブルーチンと呼びます）を定義できますが、データ型は整数のみで、配列やポインタはありません。

2. 定数と宣言

2.1 識別子

変数、手続き、関数に付ける名前（識別子）は、英文字から始まり、英文字または数字の列から成ります。大文字、小文字は区別します。

以下の語は予約語として役割が決まっており、変数名などに利用できません。

and	break	call	declare	do	else	elsif
end	for	func	if	input	not	or
print	println	proc	return	step	then	to
var	while					

2.2 定数と文字列

定数は数字の列で整数を表します。数字列の直前に符号（+または-）を記述することができます。

文字列は複引用符（ダブルクォーテーション）で囲んで表現します。ただし、文字列は後述のprint文、println文でしか利用できません。

文字列には空白および文字として表示できるASCII文字（0x20～0x7e）、およびUTF-8エンコーディングのUnicode文字を含むことができます。C言語のように、'\' とその直後の1文字で特別な意味を表します。

`\"` 複引用符 `\\` バックスラッシュ `\n` 改行文字 `\t` タブ文字

2.3 変数宣言

変数はサブルーチンの外部で宣言する大域（グローバル）変数と、サブルーチンの内部で宣言する局所（ローカル）変数があります。扱えるデータ型は整数だけなので、型を指定する必要はありません。大域変数はプログラムの実行中はずっと存在していますが、局所変数はサブルーチンの実行中だけ存在します。

予約語の **var** に続けて変数名と、さらに **=** および初期値を記述します。初期値は（その値が使われなくても）必ず指定しなければなりません。

局所変数の初期値には一般の式を記述できますが、大域変数の初期値には整数定数しか指定できません。

局所変数の宣言は、新しい変数を宣言し、それに対する代入を行う実行文として記述します。サブルーチンの冒頭で記述する必要はありませんが、値を参照するより前に宣言する必要があります。

入れ子になった文列で宣言した変数は、外側の局所変数、または大域変数よりも優先します（C言語などと同じスコープ規則です）。

2.4 サブルーチン定義

Duskulでは結果の値を返すサブルーチンを関数、値を返さないものを手続きと呼びます。

関数は予約語 **func** を、手続きは予約語 **proc** を先頭に記述し、名前を表す識別子、仮引数列を続けます。仮引数列は **()** 内に識別子をカンマで区切って列挙します。仮引数が必要ない場合でも **()** は記述します。その後に局所変数の宣言や後述する文を列挙し、最後に予約語 **end** を置きます。関数の場合、必ず **return** 文で値を返す必要があります。

関数の定義例を示します（図2-1）。

```
func factorial(n)
    if n < 2 return 1
    var val = n * factorial(n - 1)
    return val
end
```

図2-1 関数の定義例（階乗の計算）

2.5 プログラムと前方参照

Duskulのプログラムは、大域変数とサブルーチン、およびサブルーチンの前方参照宣言（下記参照）からなります。

大域変数、前方参照宣言は存在しなくても構いませんが、**main** という名前の手続き、または関数は必ず定義する必要があります。サブルーチン **main** は任意個の仮引数を持つことができますが、その値は常に0です。関数として値を返した場合、その値はシェルに対する終了ステータスとなります。

サブルーチンと、大域変数はどちらも、使うよりも前に宣言または定義されている必要があります。

サブルーチンが互いに呼び出しあう場合など、サブルーチン本体の定義よりも先に名前と仮引数の個数などを知りたいことがあります。その場合、サブルーチンの前方参照宣言を記述しておくことができます。前方参照宣言は、予約語 **declare** の後に **proc** または **func** と名前、仮引数列を記述します。図2-1の関数の例ならば次のように記述できます。

```
declare func factorial(n)
```

図2-2 factorial関数の前方参照宣言

2.6 記法

Duskulでは改行や字下げ（インデント）は意味を持ちません。これらは空白と同じです。

変数名、定数、予約語などのように空白で区切らないと意味が変わってしまう構文要素は適切に空白または改行を置いて区別します。

区切り文字として空白（または改行）が置ける場所に、コメントを記述できます。コメントは連続する2つの `"/`、つまり `//` から行末までです。コメントには日本語などの任意の文字を含めることができます。文字列内にはコメントは記述できません。

3. 式

3.1 演算子

Duskulの式では、表3-1に示す二項演算子が利用できます。優先順位の高い順に記述します。演算の順序は、`()` を使って明示的に指定することもできます。

論理演算子は引数（オペランド）を、0を偽、0以外を真として評価し、結果として 1 または 0 を返します。短絡評価は行いません。つまり、左側の引数の値を評価した段階で結果が決まる場合でも、右側の引数の評価も行います（この点はC言語と異なります）。

比較演算子で「等しくない」を表す記号はC言語と異なりますので注意して下さい。

単項演算子は表3-2に示す3種類だけで、いずれも引数の左側に置きます。複数の単項演算子を `()` なしで連続して置くことはできません。

表3-1 二項演算子

演算子	説明
<code>*</code> <code>/</code> <code>%</code>	乗算、除算、剰余
<code>+</code> <code>-</code>	加算、減算
<code>==</code> <code><></code> <code>></code> <code><</code> <code>>=</code> <code><=</code>	等しい、等しくない、より大きい、より小さい、以上、以下
<code>and</code>	論理積
<code>or</code>	論理和

表3-2 単項演算子

演算子	説明
<code>+</code>	正の単項演算（恒等演算子）
<code>-</code>	負の単項演算（符号を反転する）
<code>not</code>	論理否定

3.2 式と値

定数は式です。変数、関数呼び出し（3.3を参照）、`()` で括った式、またはそれらの前に単項演算子を置いたものも式です。さらに、2つの式に二項演算子を適用したものも式です。

Duskulはデータ型として整数だけを持ちます。制御構文の条件式では、0は偽、0以外は真と評価します。

3.3 関数呼び出し

式の中に関数呼び出しを記述できます。

関数呼び出しは、関数名に続いて () の中に実引数として式を記述します。実引数は、関数定義で示された仮引数と同じ個数だけ、カンマで区切って置きます（図3-1の例を参照）。引数がない場合でも () は必要です。

関数は再帰呼び出しが可能です（図2-1の例を参照）。

```
func max(a, b)
    if a > b return a
    return b
end

proc main()
    var m = 0
    var n = 0
    input(m, n) // 2つの値を端末から入力
    println("大きいのは ", max(m, n))
end
```

図3-1 2つの引数を入力して大きな値を返すプログラム

4. 文

4.1 代入文

大域変数、局所変数に式の値を代入できます。代入文自体は式ではありません。

変数 = 式

また、n += 1 のような複合代入演算子を用意されていません。

4.2 CALL文

手続きを呼び出します。

call 手続き名 (式 , ...)

引数が必要ない場合でも、() は記述する必要があります。

手続きは再帰呼び出しが可能です。

4.3 IF文

条件によって実行する内容を変更します。次の形式は、式が真と評価された時に文列（0個以上の文の連続）を実行します。

```
if 式 then 文列 end
```

次の形式は、式が真の時に文列1、偽の時に文列2を実行します。

```
if 式 then 文列1 else 文列2 end
```

次の形式は、式1が真の時に文列1を実行、そうでなければ式2が真の時に文列2を実行、そうでなければ式3が真の時に文列3を実行、そうでなければ文列4を実行します。elsifの節はいくつあってもよく、最後のelse節はなくても構いません。

```
if 式1 then 文列1  
elsif 式2 then 文列2  
elsif 式3 then 文列3  
else 文列4 end
```

予約語は **elsif** です。「elseif」や「elif」ではありません。また、「else if」と記述すると、if以降が別のif文と判断されます。

BREAK文または RETURN文（下記参照）を実行するだけの IF文を、簡略化された方法で記述できます。下の3つの形式は、右側のコメントのように記述するのと同じ働きをします。

```
if 式1 break           // if 式1 then break end  
if 式2 return          // if 式2 then return end  
if 式3 return 式4       // if 式3 then return 式4 end
```

条件に合致しなかった場合の動作を else節や elsif節に記述したい場合には、これらの形式（thenを記述しない形式）は利用できません。また、elsif節をこのような簡略形式で記述することはできません。

4.4 WHILE文

条件式が成立する間、文を実行し続けます。

```
while 式 do 文列 end
```

まず式を評価し、偽であれば文列は実行しません。式が真なら文列を実行し、式が真である間は式の評価と文列の実行を繰り返します。

4.5 FOR文

指定した変数の値を次々に変化させながら、文を繰り返し実行します。

```
for 変数 = 式1 to 式2 step 式3 do 文列 end
```

変数に式1の値を代入して、文列を実行します。次に、変数の値に式3の値を加算し、その値が式2を超えていなければ再び文列を実行します。その後、式3の加算、式2との比較、文列の実行を繰り返します。式1、式2、式3は for文の実行直後に一度だけ評価され、繰り返しの中で値は変化しません。

式3が定数1である場合は、stepと式3を省略して次の形式を使うことができます。

```
for 変数 = 式1 to 式2 do 文列 end
```

for文の終了条件は、stepの値（刻み幅）が正なら、変数の値が toの値よりも大きくなった場合です。stepの値が負なら、変数の値が toの値よりも小さくなった場合に終了します。stepの値が0の場合、文は一度も実行せずに終了します。

例えば、次の例は変数 x の値を 1, 2, 3, 4, 5 と変化させて文を実行します。

```
for x = 1 to 5 do 文 end
```

次のように指定すると、t の値は 0, 2, 4 となります。t = 6 の時は文は実行しません。

```
for t = 0 to 5 step 2 do 文 end
```

次の指定では、e の値は 9, 6, 3 となり、e = 0 の時は文は実行しません。

```
for e = 9 to 1 step -3 do 文 end
```

forの直後、変数の前に **var** を置くことができます。この場合、繰り返しを制御する変数はこのfor文の内部でのみ利用できる局所変数となります（図4-1のプログラム例を参照）。

```
for var i = 1 to n do call f(i) end
```

4.6 BREAK文

while文、または for文の繰り返しの中で実行し、繰り返しを中断して抜け出します。

```
break
```

繰り返し文がネストしている場合、最も内側の繰り返しから抜け出します。また、breakは文列の途中に記述することはできません。

4.7 RETURN文

サブルーチンの中で実行し、サブルーチンの実行を終了させて抜け出します。

関数では returnの次に返り値となる式を指定します。手続きでは式の指定はできません。

```
return 式
```

```
return
```

returnは文列の途中で記述することはできません。

4.8 PRINT文

式の値、および文字列を端末に表示します。

```
print( 式または文字列, ... )
```

```
println( 式または文字列, ... )
```

複数の式、または文字列を指定する場合はカンマで区切ります。出力される数字、文字列の間に空白などは挿入されません。

printを指定すると、出力はシステムのバッファ内に書き出されるだけで、端末には表示されません。printlnを指定すると、バッファ内の情報も含め、値を表示して改行します。引数を指定しない println() という書き方は改行だけを行います。

図4-1の例は掛け算の九九の表を表示します。

```
proc main()
  for var i = 1 to 9 do
    for var j = 1 to 9 do
      var m = i*j
      if m < 10 then print(" ") end
      print(" ", m)
    end
    println() // 改行だけを行う
  end
end
```

図4-1 九九の表を表示するプログラム

4.8 INPUT文

端末から値を読み込み、指定した変数に代入します（図3-1のプログラム例を参照）。

```
input( 変数, ... )
```

変数は1つ以上指定する必要があります。() 内に式は記述できません。

5. 実行

5.1 コンパイル

ソースプログラム一式をダウンロードしたら、適当なディレクトリに展開します。

ソースファイルの中に `makefile` というファイルが存在しているはずです。このファイルにプログラムをコンパイルして実行形式を作る方法が記述されています。ターミナルでソースファイルの存在するディレクトリをカレントディレクトリにしてから、次のコマンドを入力します。

```
make
```

すると、自動的にコンパイル、リンクが行われて **dusk** という実行形式のファイルが生成されるはずです。

5.2 実行とエラーメッセージ

この実行ファイル `dusk` のコマンドラインの引数として、`dusk` のソースファイル名を指定すると、プログラムの構文解析と実行が行われます。

構文解析の結果、プログラムに誤りがあれば、行数とエラーの内容が簡単に表示されます。下の例は `'')` が多すぎるためにエラーとなっています

```
$ ./dusk sample.dus
ERROR: 文が不正です [symbol ')]
Line#19:      println("(A) ", a + b - -c + -(100-d)))
```

ただし、プログラムが誤りと判断した時点の結果が表示されますので、（大抵のプログラミング言語と同様に）人間の感覚と多少ずれている場合もあります。

`duskul` のソースファイル名は任意で、拡張子にも決まりはありません。

6. 演習問題

- (1) 端末から正整数をいくつか読み込み、その最大値を出力する Duskul のプログラムを作成しなさい。ただし、データ列の終わりを表すために 0 を入力するものとする。
- (2) 端末から西暦年を表す整数を 1 つ読み、その年がうるう年かどうかを表示する Duskul のプログラムを作成しなさい。
- (3) 端末から正整数を 1 つ読み、それが素数かどうかを調べる Duskul のプログラムを作成しなさい。
- (4) 端末から 2 つの正整数を読み込み、それらの最小公倍数を表示する Duskul のプログラムを作成しなさい。
- (5) 端末から 1~10 の範囲の整数を 1 つ読み、その長さの辺を持つ四角形を表示する Duskul プログラムを作成しなさい。表示は右図（この場合、入力は 3）のようになさい。
- (6) 引数として指定した文字数まで次のような「目盛り」を表示する手続きを作成し、これを使う Duskul のプログラムを作成しなさい。引数は 100 以下の正整数と仮定してよい。例は引数に 36 を指定した場合である。

```
+----+
|    |
|    |
|    |
+----+
```

```
-----+-----1-----+-----2-----+-----3-----+-
```

- (7) 正整数を 2 進数として表記した場合の 0 と 1 の列を書き出す手続きを作成しなさい。これを使って、入力した 2 つの数とその和を 2 進数で表す Duskul のプログラムを作成しなさい。
- (8) 端末から読み込んだ正整数から、元の 10 進数表記を左右逆にした数を生成して出力する Duskul のプログラムを作成しなさい。例えば、10032 を入力すると、23001 を出力するようにする。
- (9) m 人を n 組に分ける方法が何通りあるか考えよう（ $m \geq n$ とする）。この組み合わせの数を $S(m, n)$ で表す。まず、 $S(m-1, n)$ がすでに分かっているとすると、最後の 1 人は n 組のいずれかに入ることになる。また、 $S(m-1, n-1)$ が分かっているとすると、最後の 1 人を新しい組にすることで n 組めを作ることができる。また、正の数 m について、 $S(m, m)$ は 1 通り、 $S(m, 1)$ も 1 通りしかない。 $S(m, 0)$ は 0 である。つまり、

$$S(m, 0) = 0$$

$$S(m, m) = S(m, 1) = 1$$

$$S(m, n) = n * S(m-1, n) + S(m-1, n-1) \quad \text{ただし } m > n.$$

2 つの整数 m と n （ただし $m \geq n$ ）を入力し、 $S(m, n)$ を計算する Duskul プログラムを作成しなさい。