3.7 プログラミング keyword ver. 0.1

プログラミングのモジュール化として、１行関数の定義としてfn、リターンが１つの場合の一般的な関数をつくるproc、そしてそのｎ個のリターンの場合の一般形であるproc(n)を見てきましたが、残る１つの方法は、keyword機能を作成するというものです。利用頻度はそう多くはないと思われますが、GAUSSのプログラムのテクニックを磨く上で練習する価値はあると思われます。合わせて、文字列からの数値の分離などの扱い方も説明します。

ポイント　Keywordで定義されるprocedureはインプットの引数が１つだけの文字列、リターンはなしの特殊な形。ただし、引数には数値も使える。

```
keyword 関数名(文字列変数);
（　計算部分　）
endp;
```

Keywordは、インプットにくる引数が文字列１個、リターン０個の特殊なprocedureの仲間をつくる機能を持ちます。リターンが０個というのは、想像がつくと思います。外部に受け渡すリターン変数がないという意味ですが、内部ではリターンとは別にprintもgraphも何でもできることはproc(n)の章のところで示したとおりです。しかしながら、インプットの引数に文字列が１個だけくるというのは、少しわかりづらいかもしれません。この点公式マニュアルでは言葉足らずの感があり、何か特殊なことをするような感じで書かれていますので、その点を改める意味でも、まずは、単純なプログラムから始めてみます。

プログラム

```
new; cls;
circle(5);

keyword circle(x);
    local a;
    a=x*x*pi;
    print "area of circle=" a;
    print "              if r=" x;
endp;
```

画面表示

```
area of circle=        78.539816
          if r=        5.0000000
```

上の例は、インプットの引数に数値を使ったものです。Keywordで定義される関数の引数に

は、文字列だけでなく、一般のprocやfnの時と同じように数値もくることができます。という意味において、keywordとprocの区別はありません。（公式マニュアルの記述は完全ではありません。）もし、文字列しかインプットの引数に受け付けないのであれば、

```
circle(“5”);
```

とプログラムの２行目はしなくてはいけないはずです。このプログラムは引数に数字を入れた場合のみ動作します。さて、上のプログラムは、同じ名前の国産のGAUSSというフリーソフトと同じことをするプログラムです。半径を与えたときに、その場合の円の面積を求めるプログラムです。それを、半径ｘをインプットして、ローカル変数aを用いて、circle(x)という関数で計算させ、その内部で「リターン変数を作成することなしに」円の面積とその際にインプットした半径の長さをprint文で注釈つきで画面表示させたものです。リターンはあってはいけませんから、当然ながら、retp文はありません。なお、retp;という形式で書かれていれば、リターン０ということを意味します。分岐した際には、分岐してそこで終わる一方にこのretp;をend文と同じ役割をさせています。（これはfn(0)の場合にも同様に使えます。）本来のGAUSSが意図したkeywordの使い方ではありませんが、円面積計算のようなprocの簡単なプログラムもできるということをまず理解しておいてください。

次に、少し文字列の操作と数値としての認識のしかたについて、予備知識として説明します。いま、変数ｓという中に「文字列として」数字を”1 2 3 4 5”と読み込んだとします。文字として代入していますので、引用符に包まれています。なお、GAUSSには空白の意味は数値と数値の間の区別、あるいは文字と文字の間の区別にしかつかわれませんから、ｓという変数の中には、一続きの文字列として、１と２と３と４と５が入っていることになります。これを、ループのアルゴリズムで左端の１から順に、２，３，４，５というふうに１つずつ取り出していきます。今仮に、これらの数字を左端から順に取り出して、かけていってその積の計を求めてみましょう。

プログラム

```
new; cls;
s="1 2 3 4 5";

mul = 1;
do until s $== "";
    { tok, s } = token(s);
    print stof(tok);
    mul = mul * stof(tok);
endo;

print "----------------";
```

```
    print mul;
```

画面表示

```
    1.0000000
    2.0000000
    3.0000000
    4.0000000
    5.0000000
----------------
    120.00000
```

ここで、文字列を処理する関数としてtokenという組込み関数とstofという組込み関数が新たに登場します。組込み関数tokenは、その引数に文字列を代入して、２つのリターンを返します。最初のリターンには、その文字列の左端の最初の文字（空白がある区切りまでといういみ、最初が11であれば、１ではなくて11がくる）が、２番目のリターンには、残りの文字列が入ります。

```
                                s="1 2 3 4 5"
                                ↓
                             token(s)
ループ１回目    リターン   tok="1"   s="2 3 4 5"
                                ↓
                             token(s)
ループ２回目    リターン   tok="2"   s="3 4 5"
                                ↓
                             token(s)
ループ３回目    リターン   tok="3"   s="4 5"
                                ↓
                             token(s)
ループ４回目    リターン   tok="4"   s="5"
                                ↓
                             token(s)
ループ５回目    リターン   tok="5"   s=""
                                ↓
                          s $== ""なのでループからぬける
```

上のように{ tok, s } = token(s);において、ループ１回目には関数の引数には"1 2 3 4 5"が入っていますから、リターンには、tok="1"で、s="2 3 4 5"が返されます。そのリターンｓが再帰的にまた関数tokenの引数に入って、これを、do until s $== "";　で示されるように、s

が空になるまで繰りかえします。ここで、””とは引用符が２つ重なった形で、中味が空という状態を表します。また、この場合数値としてではなく、sは文字列として表されていますから、論理等号関係を表す＝＝の前に、文字の演算としての＄のマークがついています。今、次の行のprint stof(tok);およびmul = mul * stof(tok);の行を後回しにして考えるとすると、endo文までの間のループを、リターンのsが１つずつ左側から減って、文字列の中味が””(空)になるまで繰り返します。そして、ｓが空になると、ループからぬけてendo文以降にプログラムの流れは移ります。英語でtokenとは、最小の字句単位のことで、ここでは文字列を左端の字句単位と残りの部分とにわける働きをしています。その次に、ループの中で使われているstofという組込み関数は、英語でstring to floating pointの略で、引数の文字列stringを小数floating pointに変換するものです。この組込み関数の操作をへて、はじめて、切り取られた最小字句単位tokenは、計算が可能となる通常の数字に変換されます。なお、この変換なしには、四則の演算などの計算はできません。その場合には、依然として＄をつけた演算を強いられることになります。ですから、最初に変数tokとして切り取られた”1”という文字は組込み関数stofを通って、1という数字になります。同様にして、２回目のループでは”2”という文字が2という数字に変換され、これが５まで繰り返されます。この数字に変換されたtokに入った数字を、１を初期値として

ループ１回目　　１　←　**1**×１
ループ２回目　　２　←　１×２
ループ３回目　　６　←　２×３
ループ４回目　　２４　←　６×４
ループ５回目　　１２０←２４×５

５回繰り返して、１×２×３×４×５の計算をさせているのです。アルゴリズムとしては、足し合わせるときには初期値は０にしておく必要がありますが、かけ合わせるときには１にしておく必要があります。mul = mul * stof(tok);のイコールサインは後ろのものを前のものに「代入する」という意味です。この場合、１回目のループで計算されたmulに入っている１が、２回目のループでは、後ろの２とかけられる方のmulに入って,、その計算がイコールサインの前のmulに入って２になったものが、３回目のループでは、後ろの３とかけられる方のmulに入って、その計算がイコールサインの前のmulに入ります。これを繰り返して、最後から１つ前のループの計算の答えのmulが、最後のループで５とかけ合わせるmulに使われて、その答えがmulに入った答えが最終的に120になっています。最後にループを抜けて、print文でしきりのマークの線を表示させた後に、この最終的な120が入った変数mulを画面表示させています。なお、ループの内部にprint stof(tok);を置くことによって、ループのたびの左端のtokenで切り取られた文字変数tokを小数に直した形、すなわち、そ

の場所の数値1,2,3,4,5をそれぞれのループの回数ごとに画面表示させています。

ポイント　{ tok, s } = token(s);　**文字列ｓの左端の最小字句単位を切りとってtokとし、残りの文字列をｓに代入格納する**

ポイント　stof(**文字変数**)　　**文字変数を小数（数字）に変換して返す**

さて、これを引数が１、リターンが０のkeyword形式のprocedureにしてみましょう。

プログラム

```
new; cls;
multip("1 2 3 4 5");

keyword multip(s);
    local mul,tok;
    mul = 1;
    do until s $== "";
        { tok, s } = token(s);
        print stof(tok);
        mul = mul * stof(tok);
    endo;
    print "----------------";
    print mul;
endp;
```

上のようにkeyword文とそれをしめるendp文、それにそのprocedure内で使われるローカル変数を宣言するlocal文を付け加えるとkeywordは完成です。ここでは、ローカル変数として、mulとtokの２つを使っています。関数名をmultipという名前に定めて、そのインプットである引数にsを用いています。この掛け算をさせるkeyword関数multipを呼び出すには、０リターンのprocedureと同様、関数名をそのまま書くか、call文で呼びます。ここでは、そのまま書いてやり、引用符に包まれた文字列としての1 2 3 4 5をその引数に代入しています。例えば、２行目をmultip("1 3 5");と変更してやれば、ループは３回だけまわって

画面結果

```
        1.0000000
        3.0000000
        5.0000000
----------------
        15.000000
```

というふうになります。もっと長い計算も自在にできます。ただし、この関数には引数に

文字列しか入ることはできませんから、引用符で包むことを忘れないようにしてください。

さて、ここまでが前置きです。上のような形でも一応procedureとして機能しているのですが、キーワードと呼ぶのには簡単に呼び出すことはできません。そこで、本当のKeywordというものを作成するのには、次のような作業をします。

まず、キーワードを呼び出す部分を切り取って、keywordのprocedure部分だけにします。ここで下のように便宜上、print;文で区切りの線を一本をendp文の前に入れます。

プロシジャー（　multip.g　）

```
keyword multip(s);
    local mul,tok;
    mul = 1;
    do until s $== "";
        { tok, s } = token(s);
        print stof(tok);
        mul = mul * stof(tok);
    endo;
    print "----------------";
    print mul;
    print "----------------";
endp;
```

上のようにできあがったプロシジャー部分だけのところをGAUSSフォルダー内のsrcというもう１つ下の階層にあるフォルダーにmultip.gという名前をつけて保存します。ファイル名は、keywordで指定したものと同じ名前、拡張子はドット以下ｇとします。これで、掛け算をするキーワード機能の登録が完了しました。

登録したキーワードの使い方は、いたって簡単です。いままで使われていなかったウインドウであるCommand Windowを使います。>>のマークがあるウインドウです。ここに、直接、次のように書いてリターン（ENTER）を押します。（>>のマークは入力しません。）

```
>> multip 3 4 5;
```

画面結果

```
        3.0000000
        4.0000000
        5.0000000
----------------
        60.000000
```

一度目はエラーがでるかもしれませんが、２度目からはうまく行くはずです。関数が一度では登録されないからです。２回目には、srcフォルダーに定義された組込み関数の１つとして動作します。もし、エラー処理のアルゴリズムを組み込もうと思うならば、ローカル変数の宣言文の次の行あたりに、

```
if s $== “”;
    errorlog “The input is null.”;
    retp;
endif;
```

というふうに””、すなわち何もインプットしないで mutip; とだけ入力してリターン（ENTER）とすれば、The input is null.というエラーメッセージが出されるようになります。このアルゴリズムはあってもなくても、さほど大勢には影響はないでしょう。つけたければ、つけてください。

このように、keywordに限らず、独自をprocedureを作成してGAUSSの組込み関数と同じように使うには、拡張子をｇとして、procedureを保存します。この場合は、非常に特殊な場合で、Command Window上の入力は文字列として扱われるという特性を使って、キーワードを作成したのです。KeywordそのものがComand Window上の入力を文字列として扱って計算をする役割をもっているのではなくて、組込み関数tokenとstofの組み合わせでループをまわすことにより、画面上の文字列を最小字句単位ごとに変数の中に読み込んでいるのです。なお、Command Window上では、＋、－、／、＊の演算記号を使って直接、電卓と同じ計算を行なうことができます。結果は、Output Windowに出力されます。

```
>> 1+2+3+4+5+6
```

などと直接入力してリターン(ENTER)を押すと、電卓機能による計算結果がOutput Windowに表示されます。これは、標準のGAUSSの内部機能の１つです。

なお、新たに作成した組込み関数multipを通常のプログラムの中で呼び出すのには、

プログラム

```
new; cls;
multip("1 2 3");
```

画面表示

```
    1.0000000
    2.0000000
    3.0000000
----------------
    6.0000000
----------------
```

と通常のEditのwindowでプログラムすると、通常の組込み関数と遜色なく自由に使えます。この際、multipのkeyword形式のprocedureを後方に置く必要はまったくありません。すでに、srcのフォルダーにｇという拡張子がついたプログラムが存在することによって、GAUSSは自動的にその組込み関数にこのmutipを加えているからです。（なお、この機能は、GAUSSのautoloderという機能がOFFになっている場合には関数は認識されません。通常の設定ではONになっているはずです。）ただ気をつけるべきことは、この関数がたまたま文字列を扱う関数であるあるために、Edit画面上では引用符に包んで文字列として引数を記述する必要があります。これは、keywordだからということではなくて、procedureとして文字列を扱うtokenやstofといった関数を使って文字列を読み出すプログラムをしている結果、この関数が文字列を要求しているだけなのです。文字列を１つずつ読み出すプログラムにしていれば、Command Window上では、入力はすべて文字列としてGAUSSに受け渡されるという性質がありますから、

```
>>  multip 1 2 3;
```

という具合に書いて、計算することができるようになります。この点、非常に誤解の多いところなので、些細なところではありますが、しっかりと理解してください。公式マニュアルに書かれているのは、このプログラムの文字列が最初から空の場合を想定したエラー処理つきの足し算のバージョンで、初期値を０とおいて文字列を左端から同じように１つずつ読みこんで足し合わせています。procedure部分を切りとって、add.gと名前をつけてやった上で、GAUSSのsrcフォルダーの中に保存すれば、

```
>> add 1 2 3 4 5;
```

などとCommand Windowで入力すれば、足し算の結果がOutput Windowに表示されます。

ポイント　**procやkeywordでできた関数は、srcディレクトリーに拡張子.gをつけて保存すれば、autoloderがONであるかぎり、自動的に組込み関数として認識される**

ポイント　**Command Windowは電卓計算が行なえると同時に、Keywordでできたprocを呼び出して、「文字列として」数値を代入できる**