1 Mapleでのプログラミング

Maple は普通のプログラミング言語と同じくプログラムが組めます．ヘビーな計算には向きませんが，ちょっとしたプログラムなら Maple で十分です．すでに組み込まれている Maple の関数を利用すれば，驚くほどはやくプログラムを組むことが出来ます．また，C 言語と似ていますので，書き換えることも容易です．

1.1 値の代入，出力

Maple は C 言語などのコンピュータ言語と違って変数の初期設定で型宣言をする必要がありません．

```
> i:=1;
```

$$i := 1$$

```
> x:=2;
> y:=3;
> z:=x+y;
```

$$x := 2$$
$$y := 3$$
$$z := 5$$

出力を明示的におこなうには print を使います．さらに，C 言語の printf と同じ形式で出力整形をすることが可能です．複素数の指定も出来ます．

```
> printf("%10.5f\n",z);
```

```
5.00000
```

```
> printf("%10.5Zf\n",z+y*I);
```

```
5.00000  +3.00000I
```

1.2 算術演算

算術演算子+,-,*,/が通常の和，差，積，商を表します．ただ，商は，

```
> 3/4;
> 3/4+2/5;
```

$$3/4$$
$$\frac{23}{20}$$

のように分数が出てきます．整数の割り算には irem(余り) と iquo(商) があります．

```
> irem(10,3);
> iquo(10,3);
```

$$\frac{1}{3}$$

浮動小数点数に直すには evalf を用います．第 2 引数が出力桁数を指定します．

```
> evalf(10/3);
```

$$3.333333333$$

```
> evalf(Pi,30);
```

$$3.14159265358979323846264338328$$

浮動小数点演算の有効数字はデフォルトでは 10 桁ですが，Digits に値を代入することによって有効数字を一括して指定することが可能です．

```
> Digits:=20;
> evalf(exp(1));
```

$$Digits := 20$$

$$2.7182818284590452354$$

浮動小数点数から整数に直すにはいくつかの関数があります．

trunc 数値から数直線で 0 に向って最も近い整数

round 数値の四捨五入

floor 数値より小さな最も大きな整数

ceil 数値より大きな最も小さな整数

負の値の時に floor と trunc は違った値を返します．小数点以下を取りだすには frac が用意されています．%は C 言語とは全く違った働きが与えられています．直前の出力結果を参照します．%%は二つ前，... となっています．

```
> %;
```

$$2.7182818284590452354$$

べき乗は^です．

```
> 1.2^3.4;
```

$$1.8587296919794811670$$

1.3 プログラムの流れの制御

1.3.1 if-else

条件分岐の基本となる if-else です．構文は

```
if <conditional expression> then
  <statement sequence>
| elif <conditional expression> then <statement sequence> |
```

```
| else <statement sequence> |
end if
(| |中はなくともよい)
```

です．具体例で，ある値の絶対値を返すには

```
> x:=-4;
if (x>0) then
  y:=x;
else
  y:=-x;
end if;
```

$$x := -4$$
$$y := 4$$

となります．関係演算子は<，<=，>，>=，=，<>で表記されます．論理演算子には and，or，xor，implies，not があります．その他にもブール値を返す関数として evalb, type などいくつかあり，条件分岐に使えます．

1.3.2 do-loop

何回も同じ動作を繰り返す for-loop です．構文は

```
for <name> from <expr> | by <expr> |  to <expr> do
  <statement sequence>
end do;
(| |中はなくともよい)
```

です．具体的には，

```
> total:=0;
for i from 1 to 5 do
  total:=total+i;
end do;
total;
```

$$total := 0$$
$$total := 1$$
$$total := 3$$
$$total := 6$$
$$total := 10$$
$$total := 15$$

です．loop 回数が少ない間は，loop の中身も出力されます．是を止めるには end do; の最後のセミコロンをコロンに換えます．while-loop も同じように使えます．

```
while <conditional expression> do
  <statement sequence>
end do;
```

1.3.3 next, break

do-loop の途中で違った操作を加えるための命令です．next に来たら do-loop を一回スキップします．break に来たらそこで do-loop を一つ抜けます．具体例で見てください．

```
> for i from 1 to 5 do
>   if (i=3) then next; end if;
>   i;
> end do;
```

$$1$$
$$2$$
$$4$$
$$5$$

```
> for i from 1 to 5 do
>   if (i=3) then break; end if;
>   i;
> end do;
```

$$1$$
$$2$$

1.4 配列

1.4.1 リスト

[ ] で囲まれた数字などの並びは，添字 1 から始まる 1 次元の配列を表します．

```
> list1:=[1,2,3,4];
```

$$list1 := [1, 2, 3, 4]$$

リストの要素を参照するのは以下の通りです．

```
> list1[3];
> list1[-1];
> list1[2..4];
```

$$3$$
$$4$$
$$[2, 3, 4]$$

添字で-1,-2 は最後から一つ目，二つ目です．C 言語と違い添字 0 はありません．

```
> list1[0];
```

```
Error, invalid subscript selector
```

op コマンドでリストの中身だけを取りだすことが出来ます．これを利用して，リストの頭やおシリに要素を付け加えることが可能です．

```
> op(list1);
```

$$1,\ 2,\ 3,\ 4$$

```
> list1:=[op(list1),5];
```

$$list1 := [1, 2, 3, 4, 5]$$

```
> list1:=[6,op(list1)];
```

$$list1 := [6, 1, 2, 3, 4, 5]$$

リストの一要素を置換するには以下のようにします．

```
> list1[4]:=x;
list1;
```

$$list1_4 := x$$

$$[6, 1, 2, x, 4, 5]$$

リストの一要素を削除するには以下のようにします．

```
> list1:=subsop(4=NULL,list1);
```

$$list1 := [6, 1, 2, 4, 5]$$

nops(L) で要素の数を返します．

```
> nops(list1);
```

$$5$$

よく使う手を二つ．(”＃”より後ろはコメント文です)

```
> aa:=[]; #空で初期化
> for i from 1 to 3 do
>   aa:=[op(aa),i]; #付け足していく
> end do:
> print(aa);
```

$$aa := []$$

$$[1, 2, 3]$$

```
> n:=nops(aa); #要素数の取りだし
> total:=0;
> for i from 1 to n do #for-loopの上限に使います
>   total:=total+aa[i];
> end do:
> print(total);
```

$$n := 3$$

$$total := 0$$

$$6$$

[ ] を二重化することで 2 次元の配列を作ることも可能で，リストのリスト (listlist) と呼ばれます．

```
> aa:=[[1,2],[3,4]];
```

$$aa := [[1,2],[3,4]]$$

```
> aa[1,2];
```

$$2$$

多重化すれば多次元配列が同様に使えます．

1.4.2 array

array というのは listlist とほとんど同じですが，添字がどこから始めても OK になっています．あらかじめサイズの分かっている配列や，1 からでない添字を使いたいときに便利です．

```
> A:=array(1..2,1..2,diagonal);
> print(A);
```

$$matrix\left([[A_{1,1},0],[0,A_{2,2}]]\right)$$

```
> B:=array(-1..1,-1..1,symmetric);
> print(B);
```

省略

convert で array と listlist などの相互変換が可能．

```
> A2:=convert(aa,array);
> print(A2);
```

$$[1,2,3] := vector\left([1,2,3]\right)$$
$$vector\left([1,2,3]\right)$$

1.4.3 set

{} によって囲まれた並びは set と呼ばれ，集合とおなじ取り扱いができます．リストと異なり，要素には重複がなく，基本的に順番がありません．したがって，以下の三種の set は同じ集合を意味する．

```
> {x,y,z},{y,z,x},{x,x,y,z,z,x};
```

$$\{z,y,x\}, \{z,y,x\}, \{z,y,x\}$$

集合の要素の操作には以下のような，和 (union)，積 (intersect)，差 (minus) が用意されている．

```
> {x,y,z} union {u,v,z};
```

$$\{z,v,y,x,u\}$$

```
> {x,y,z} intersect {u,v,z};
```

$$\{z\}$$

```
> {x,y,z} minus {z};
```

$$\{y, x\}$$

要素の取出はリストと同様に [ ] によるが，順序が Maple の内部事情で変わるので注意.

```
> {x,y,z}[1];
```

$$z$$

```
> {x,y,z};
```

$$\{z, y, x\}$$

1.5 proc 化

ここで紹介した制御の流れを使って複雑なプログラムを組むことが可能となります．ある程度長くなった，あるいは何度も呼びだすルーチンは関数として登録しておくと便利です．これには proc 関数を使い，構文は以下の通りです.

```
<name>:=proc(<引数>);
  local <変数>;
  global <変数>;
  <中身>
end proc
```

global,local を省略しても Maple が適当に判断してくれます．したがって最も単純には，動いたプログラムに，<name> :=proc() を頭につけて，最後に end proc とすれば即席関数ができ上がります．global,local は C 言語と同じ意味で，関数の内部だけで使われるのが local, 外部を参照するのが global です.

リストを受け取ってその和を返す関数を示します．proc の次の ( ) に受け取る引数を書き，戻り値は最後の実行文です．ここでは，a というリストを受け取り，要素数を見て，和をとり，最後にその和を返しています.

```
> total:=proc(a)
> local S,n,i;
> n:=nops(a);
> S:=0;
> for i from 1 to n do
>   S:=S+a[i];
> end do:
> eval(S);
> end proc;#ここまでが関数定義
> aa:=[3,5,7];
> total(aa);
```

$$aa := [3, 5, 7]$$

演習問題

1. C 言語で作った以下の素数判定プログラムを Maple script に書き換えよ．ただし，scanf は値の代入に置き換え，整数の余りは irem 関数を使え．

```
#include <stdio.h>
int main(void){
  int i,n;
  scanf("%d",&n);
  for (i=n-1;i>1;i--){
    if (n%i==0){
      break;
    }
  }
  if (i==1){
    printf("%d is a prime number.\n",n);
  } else {
    printf("%d is not a prime number.\n",n);
  }
  return 0;
}
```

2. グーグル、謎の人材募集広告の問題｛e の値中の、最初の連続する 10 桁の素数｝を解け．

   使用する Maple 関数：

   evalf ：指定した精度で実数表記．

   floor ：小数点以下を切り取った整数で表示

   isprime ：入力が素数なら true, 違えば false

   まず，evalf で 200 桁の exp(1) を求め，それぞれの桁の数値を floor と 10 倍計算を組み合わせて取りだす (下記参照)．その後，連続する整数を 10 桁の整数に直し，isprime で素数かどうかを検証するループを回す．

   それぞれの桁の数値を取りだす部分は，例えば

```
EE:=2.718; i:=1;
AA[i]:=floor(EE);
E0:=10*(EE-AA[i]);
```

   となる．