[プログラミング入門Ｉ　（７）］ 関数とは（２)、再帰呼び出し

変数の有効範囲と有効期限

関数の中で宣言されている変数を**局所変数**（local variable, 自動変数:automatic variable ともいう）といいます。パラメータとして宣言されている変数も局所変数です。はじめに引数の値がセットされて以外は局所変数と同じです。局所変数は宣言されている関数の中でしか使うことができません。これに対して、関数の外で宣言された変数を**大域変数**(global variable)といいます。このような変数は、関数の間で共通に使うデータのための変数を宣言するために使います。関数のプロトタイプ宣言と同じように、大域変数は変数を使う前に（できれば、プログラムの先頭で宣言しなくてはなりません。void で宣言した手続きとしての関数は、大域変数で宣言されたデータに対してなんらかの操作を行うために使います。このように大域変数の値を変更することを**関数の副作用**といいます。

局所変数は、その宣言された関数が終了すると値が無効になってしまいます（変数のメモリが解放される)。大域変数は関数が終わっても値は保持されますので、もしも、関数が終わってもなにか値を保持する必要がある場合には大域変数にいれておくことができます。また、関数から返すことのできる値は１つなので、たくさんの値を返したい場合には、大域変数を用意してそこにいれて、関数からも戻ったときにその変数を参照することでできます。関数が終わっても値を保持したい場合には、**静的変数**を使うことができます。これは局所変数の宣言の先頭に static というキーワードをつけたものです。

変数を参照できる範囲のことを変数の**参照範囲**（**scope**）といいます。局所変数の有効範囲は関数の中、大域変数の有効範囲は変数が宣言され以降のプログラムです。

変数の値が保持される期間を**有効期間(extent)**といいます。局所変数の有効期間は関数の実行中、大域変数の有効期間はプログラム実行中です。静的変数とは参照範囲が関数内で、有効期間はプログラム実行中という変数です。

引数の値渡し

下のプログラムを考えてみましょう。

```
main() {
  int i;
  i = 10;
  j = foo(i);
   printf(“i=%d¥n”,i);
 }
int foo(int i){
    i = 100;
   return i;
}
```

プログラミング言語によっては、値でなく呼び出し側の i 自体が引数となることがあり、この場合は i は 100 になってしまいます(例えば、FORTRAN)。これを**参照渡し、名前渡し**いう。言語によってことなる。

foo の中で、i に 100 を代入して、i の値を変更していますが、main の変数 i は変わりません。したがって、printf の結果は 10 となります。これは、main の i の値（つまり 10）が、関数 foo にわたり、関数のパラメータ変数 i にセットされます。パラメータ変数は関数の中では局所変数と同じに扱われるので、これを変更しても、逆に main の変数 i はかわりません。このことを引数の**値渡し(call by value)**といいます。

なお、配列や文字列の場合は、呼び出し側の配列が変わってしまいます。これについてはポインターのところで説明します。

関数の再帰呼び出し

ある関数が、自分自身を呼び出すことを**再帰呼び出し(recursive call)**といいます。例えば、階乗を求める関数は次のように書くことができます。

```
int factorial(int n){
    if(n == 0) return 1;
    else return n*factorial(n-1);
}
```

つまり、n!は、(n-1)!に n をかけたものというわけです。但し、これをどこまでも続けても止まらなくなってしまうので、１のときには１！＝１としています。階乗の定義は、１から n まで掛けたものということあれば、このようなことをしなくても、単にループで１から n までかければいいのであまり意味が

わからないかもしれません。**しかし、再帰呼び出しの考え方はもっと重要な意味をもっています。この考え方を身につけることによって、いろいろな問題が簡単に解ける強力な考え方です。**
では、有名なハノイの塔の問題を考えることにしましょう。ハノイの塔とは、３本の柱があり、一番左の柱に下から大きな盤、その上に順に小さな盤が乗っているというものです。問題は

１、一度に１枚の盤しか移すことができない（片手しか使えない）
２、小さな盤の上に大きな盤を乗せてはいけない（壊れてしまう）

この条件で、左の柱から、右の柱に移すというパズルです。中間的な置き場として真ん中の柱を使うことができますが、上の条件は満たしていなくてはなりません。
この問題の解き方としてまず、大小２枚の場合を考えてみましょう。これだったら、まず小さい盤を真ん中に移して、大きな盤を左に移し、最後に小さい盤を左に移せばよいことになります。では３枚ではどうでしょうか。この場合、まず、２枚の手順を使って、上２枚を真ん中に移します。それで、一番下にある盤を右の柱に移し、また２枚の手順を使って、真ん中から右に移せばいいということなります。これを一般化して、n 枚の場合には、

１、n-1 枚をあいているところに移す。
２、一番下の番を目的のところ（右の柱）に移す。
３、n-1 枚を目的のところに移す。

という手順でやればいいということになります。これの解答に関しては講義で詳しく解説しますが、n 枚の盤を移すという関数を定義して、その関数に再帰呼び出しを使うことで、エレガントにプログラミングすることができます。
もう、一つの問題を考えてみましょう。任意の整数を一文字の出力ルーチン(putchar)を使って、プリントアウトするという問題です。例えば、123 という数字の場合は”1”,”2”,”3”と出力するという問題です。一桁の場合であれば、単にその数字を'0'に加えることによって、出力します。では２桁はどうでしょう。この場合はまず、10 で割って、その商が２桁目なので、その数字を印刷します。で、あまりを１桁にしてプリントすればいいわけです。これを一般化して n 桁の数とすると、数 d に対し、

１、まず、一桁、つまり 10 よりも小さければ、putchar(d+'0')
２、最下位以外の n-1 桁について、プリントする。つまり、d/10 を印刷。
３、最下位の桁、d%10 を印刷。

とすればいいわけです。

```
void print_number(int d)
{
  if(d >=10) print_number(d/10); /*  上の桁を出力  */
  putchar('0'+d%10); /* 最下位の桁を出力  */
}
```

再帰の考え方は、ある n について問題を解くときに、n-1 についての解答でとくことができるときに使うことができます。プログラムを考えるときに、特定のケース、特定の数についてプログラムを考えるだけでなく、それを一般的な入力、一般的な n について考えることはプログラミングについての非常に重要な考え方です。