再帰関数

山本昌志*

2007 年 5 月 22 日

概 要

再帰関数について学習する．関数の定義に自分自身を使っている場合を再帰関数呼ぶ．ここでは，再帰関数の考え方とプログラムを記述する方法について，学習する．再帰関数は編入学試験問題に出題されることが多いので，十分理解しなくてはならない．編入学試験の問題は，ここで学ぶものよりもさらに難しい．しかし，本日の講義では編入学試験に対応できるところまでは教えない．今のところ，諸君の数学の能力では編入学試験の対応までは不可能であるからである．4〜5 年生になったならば，各自，さらに高度な問題を解く訓練をしなくてはならない．

1 前回の復習と本日の学習内容

1.1 復習

前回は gcc が実行ファイルを作る順序とプリプロセッサについて学習した．

- C 言語のソースファイルから機械語の実行ファイルができるまでのプロセスは，図 1 のとおりである．
- ソースファイルはプリプロセッサーにより書き換えられる．プリプロセッサには様々な機能があるが，`#include` と `#define` について学んだ．
  - `#include` は，指定したファイルをそこに挿入する．
  - `#define` は，文字列を置き換える．

1.2 本日の学習内容

本日は，再帰関数について学習する．本日のゴールは，以下のとおりである．

- 再帰関数の意味が分かる．
- 再帰関数を使ったプログラムを書くことができる．

教科書 [1] の pp.70–79 が本日の範囲である．ただし，この範囲のうち非局所的ジャンプは説明をしない．この機能は滅多に使わないからである．

*独立行政法人　秋田工業高等専門学校　電気情報工学科

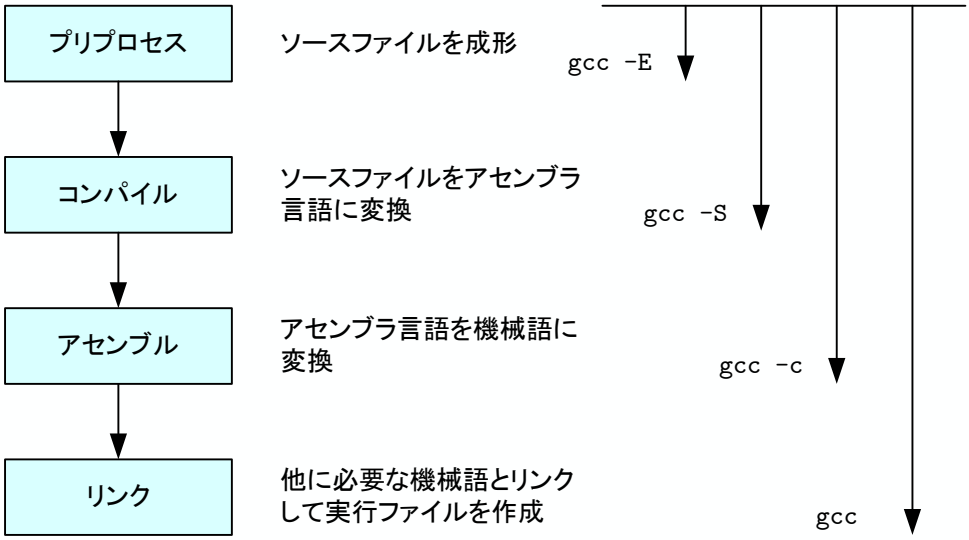

図 1: gcc が実行ファイルをつくるプロセス．左側に示すようにオプション付きで処理をすると，途中までの結果が得られる．

2 再帰関数

2.1 再帰関数の例

自分自身を呼び出す関数を**再帰関数** (recursive function) という．また，自分自身を呼び出すことを**再帰呼出し** (recursive call) という．具体的な例をもって説明するとわかりやすいだろう．

階乗 ここでは，数学で使う階乗の計算を再帰関数で説明する．諸君は，階乗についてまだ学習していないと思われるので，先に階乗について説明しておく必要があるだろう．例えば，6 の階乗は 6! と書き，次のように計算する．

$$6! = 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 720 \tag{1}$$

ようするに整数の値を 1 ずつ減らして，1 になるまで乗算しているのである．これから, 正の整数を n として，階乗には次の性質があることが分かる[1]．

$$n! = n \times (n-1)! \qquad (\text{ただし}, 2 \leqq n) \tag{2}$$

$$1! = 1 \tag{3}$$

数学では式 (2) を漸化式と呼ぶ．もう少し経てば，諸君は数列とともに詳しく学習するだろう．

[1] 本当は $0! = 1$ とすべきだが，諸君にはイメージできないので $1! = 1$ としている．

キーボードから整数を入力して，その階乗を表示するプログラムを 2 つ作成する．ひとつは再帰関数を使わないで繰り返し文で計算するプログラム，もうひとつは再帰関数を使ったプログラムである．[**注意**] これらの二つのプログラムは 12! までしか計算できない．13! 以上は，整数型の変数のサイズを越えるからである．

繰り返し文による階乗の計算プログラム　リスト 1 のようにすれば階乗の計算ができる．このプログラムの動作は，容易に想像がつくだろう．そして，作ることもできるだろう．

リスト 1: 繰り返し文を使った階乗の計算.

```
1  #include <stdio.h>
2
3  int kaijyo(int n);                    // プロトタイプ宣言
4
5  //========== メイン関数 ================================
6  int main(void)
7  {
8    int nx, result;
9
10   scanf("%d", &nx);                   // 整数入力
11   result=kaijyo(nx);                  // 関数呼出し
12   printf("%d!=%d\n", nx, result);     // 計算結果表示
13
14   return 0;
15 }
16
17 //========== 階乗を計算する関数(繰り返し文) ===============
18 int kaijyo(int n)
19 {
20   int i, p=1;
21
22   for(i=n; 1<=i; i--){
23     p=p*i;
24   }
25
26   return p;
27 }
```

実行結果

```
12
12!=479001600
```

再帰関数による階乗の計算プログラム　リスト 2 のようにしても階乗の計算ができる．このプログラムの動作は，なかなか想像がつかないと思うが，式 (2) と式 (3) の通りにプログラムしていることに気がついてほしい．これらの式のうち 2 をプログラムすると，関数の中で自分自身を呼び出すことになる．このように自分自身を呼び出す関数を再帰関数と呼ぶ.

リスト 2: 繰り返し文を使った階乗の計算.

```
1  #include <stdio.h>
2
```

```
3  int kaijyo(int n);                    // プロトタイプ宣言
4
5  //========== メイン関数 ================================
6  int main(void)
7  {
8    int nx, result;
9
10   scanf("%d", &nx);                   // 整数入力
11   result=kaijyo(nx);                  // 関数呼出し
12   printf("%d!=%d\n", nx, result);     // 計算結果表示
13
14   return 0;
15 }
16
17 //========== 階乗を計算する関数(再帰呼出し)===============
18 int kaijyo(int n)
19 {
20
21   if(n==1){
22     return 1;
23   }else{
24     return n*kaijyo(n-1);
25   }
26
27 }
```

実行結果

```
12
12!=479001600
```

2.2 再帰関数の書き方

再帰関数を使ったプログラムは，アルゴリズムさえ分かれば比較的簡単に記述できる．式 (3) に示したように，漸化式の通りにプログラムを書けば良いのである．それでは，漸化式とは何か?—と問いたくなる．漸化式とは「隣同士の関係により数列の性質を表す式」といえるだろう．ちょっと難しいかなー．例えば，先ほど示した階乗の数列は，

$$1!, 2!, 3!, 4!, 5!, 6!, 7!, 8!, \cdots, (n-1)!, n!, \cdots \tag{4}$$

である．隣との関係—すなわち漸化式—は，

$$n! = n \times (n-1)! \tag{5}$$

である．この通りプログラムを書けば再帰関数を使うことになる．

分割統治法 [2] というアルゴリズムから再帰関数を考えることもできる．問題を 2 つ以上の部分に分けて，それぞれを解く．そしてそれを合わせたもの[2]が元の問題の解となる場合である．分けられた問題は元の問

[2]問題を合わせるのであって，和になるとは限らない.

題よりも小さくなり，さらに分割できる．そして，自明な解になるまで分割する方法である．先ほど示した $n!$ の階乗の計算では，n の計算と $(n-1)!$ の計算に問題を分けることができる．すなわち

$$n! = n \times (n-1)! \tag{6}$$

である．このように問題を分けることにより再帰関数が表れる．この通りにプログラムを書くのである．

再帰関数を使うときに，再帰を終了させる条件を絶対に忘れてはならない．先ほどの階乗の計算では，

```
if(n==1){
  return 1;
}
```

が終了条件となっている．この文により，n が 1 になれば関数の呼び出しを行わない．この終了の動作の記述を忘れると，プログラムは止まらない．そしてメモリーを使いつづけるので，いずれはクラッシュして止まる．

再帰関数は難しい概念であるが，非常に便利な場合がある．問題のときかたのアルゴリズムは分からないが，分割統治法あるいは漸化式が分かっている場合，解を求めるプログラムができてしまう．このような代表例が「ハノイの塔」と呼ばれるパズルである．これについては，ちょっとばかり難しいのでこの授業では取り上げないが，興味のある者は私の web[3]でも見よ．

2.3 再帰関数を使うべきか?

再帰関数と繰り返し文のどちらを使ってもプログラムがかける場合，どちらを使うべきだろうか? 両方使える場合は繰り替えし文を使うべきである．計算速度も早いし，メモリーの消費も少ないからである．

単純な問題であれば繰り替えし文の方が良いが，複雑な問題となると再帰関数で記述はできるが，繰り替えし文ではプログラムが大変困難な場合がある．先ほど少し述べた「ハノイの塔」や後で学習する「クイックソート」の場合である．このような時には再帰関数を使い，問題をあざやかに解くことができる．

3 再帰関数の例

3.1 2 進数

入力された正の整数を 2 進数で表示するプログラムを書く．これは再帰関数を使って，記述することができる．10 進数から 2 進数に変換するとき，2 で割ってあまりを書く---という作業を繰り返したことを思い出してほしい．同じことを繰り返したはずである．

このプログラムを書くためには，2 で割った商とあまりの演算が必要である．商の演算には「/」を，あまりの演算には「%」をつかう．これは整数同士の演算でしばしば使われるので憶えておく必要がある．

準備はできた．再帰関数を使った 2 進数への変換プログラムは，リスト 3 のようになる．

[3] http://www.akita-nct.jp/ yamamoto/lecture/2005/2E/24th/html/index.html

リスト 3: 正の 10 進数を 2 進数になおすプログラム．

```
#include <stdio.h>

void print_binary(int n);                    // プロトタイプ宣言

//========== メイン関数 ================================
int main(void)
{
  int nx;

  scanf("%d", &nx);                   // 整数入力
  print_binary(nx);                  // 関数呼出し
  printf("\n");

  return 0;
}

//========== 2進数を表示する関数(再帰呼出し)===============
void print_binary(int n)
{

  if(n==0){
    return;
  }else{
    print_binary(n/2);
    printf("%d",n%2);
  }

}
```

実行結果

```
1234
10011010010
```

3.2 フィボナッチ数列

フィボナッチ数列も再帰関数の代表的な問題である．この数列は次のようなものである．

> 成熟した 1 つがいのウサギは，1ヶ月後に 1 つがいのウサギを生むとする．そして，生まれたウサギは 1ヶ月かけて成熟して次の月から毎月 1 つがいのウサギを生む．全てのウサギはこの規則に従うとし，死ぬことは無いとする．nヶ月後にウサギはつがいの数は?

この数列は，

$$1, 1, 2, 3, 5, 8, 13, 21, 34, \cdots \tag{7}$$

となる．漸化式で表すと

$$F_n = F_{n-1} + F_{n-2} \tag{8}$$

$$F_0 = 1 \tag{9}$$

$$F_1 = 1 \tag{10}$$

である．リスト 4 にこの数列の計算プログラムを示す．

リスト 4: フィボナッチ数列 (ウサギの問題).

```
#include <stdio.h>

int fibonacci(int n);                   // プロトタイプ宣言

//========== メイン関数 ================================
int main(void)
{
  int month;

  printf("何ヶ月後?\t");
  scanf("%d", &month);                          // 整数入力
  printf("つがいの数 = %d\n", fibonacci(month));  // 関数呼出し

  return 0;
}

//========== 2進数を表示する関数(再帰呼出し)===============
int fibonacci(int n)
{

  if(n==0 || n==1){
    return 1;
  }else{
    return fibonacci(n-1)+fibonacci(n-2);
  }

}
```

実行結果

```
何ヶ月後?       8
つがいの数 = 34
```

4 プログラム作成の練習

[練習 1] 整数を入力して，8 進数にして表示するプログラムを作成しなさい．再帰関数を使うこと．

[練習 2] 次の数列を計算するプログラムを作成しなさい．再帰関数を使うこと．

$$a_n = 3a_{n-1} - 2a_{n-2} \tag{11}$$

$$a_0 = 1 \tag{12}$$

$$a_1 = 2 \tag{13}$$

[練習 3] 整数を入力して，16 進数にして表示するプログラムを作成しなさい．再帰関数を使うこと．

[練習 4] (発展的問題) ハノイの塔を解くプログラムを作成しなさい．

5　課題

5.1　内容

以下の課題を実施し，レポートとして提出すること．

[**問** 1]　(復予) 教科書 [1] の第 3 章を 3 回読め．レポートには「3 回読んだ」と書け．

[**問** 2]　(復) 本日配布したプリントを 2 回読め．レポートには「2 回読んだ」と書け．

[**問** 3]　(復) 次の数列を計算するプログラムを作成しなさい．再帰関数を使うこと．

$$a_n = 4a_{n-1} - 3a_{n-2} \tag{14}$$

$$a_0 = 0 \tag{15}$$

$$a_1 = 2 \tag{16}$$

[**問** 4]　(復) ここでの学習内容でわからないところがあれば，具体的に記述せよ．

提出要領はいつものとおり．ただし，期限は 5 月 29 日 (火)AM 8:45，課題名は「課題　再帰関数」とする．

参考文献

[1] 内田智史監修，(株) システム計画研究所編．C 言語によるプログラミング　応用編　第 2 版．(株) オーム社，2006．

[2] 石畑清．アルゴリズムとデータ構造，岩波講座　ソフトウェアー科学，第 3 巻．岩波書店，2004．