ゲームアルゴリズム資料 Vol.2

佐藤　陽悦[*1]

令和 3 年 6 月 21 日 Ver.0.6a

[*1] ゲームエンジニア科

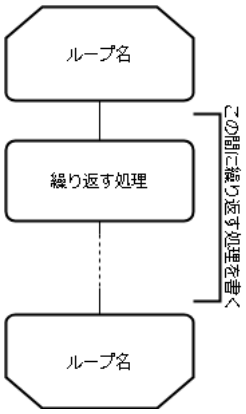
1　本日のお題：アルゴリズムとは

1.1　フローチャートの続き

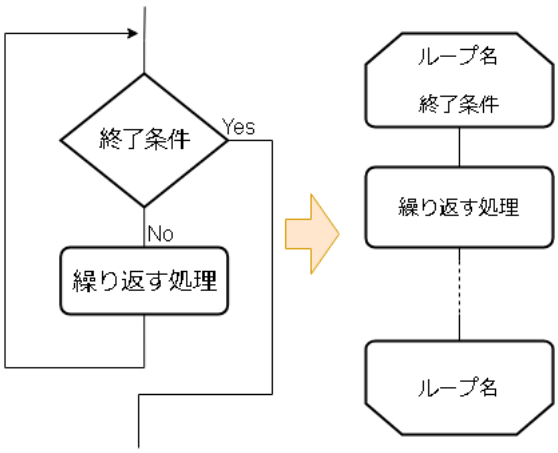
これまで、プログラムを作る上での 3 つの基本構造と、フローチャートで基本構造を使って処理内容を表現する方法を学んできた。ここからは、すこし特殊な構造や処理について説明する。

ループ端記号

今までは繰り返し構造を表現するために、判断記号と、ループを表す線記号を使って繰り返しの処理を書いていた。この煩わしい書き方を少しだけ軽減する記号がある。その記号が、いわゆる**ループ端**である。ループ端は、繰り返しの入り口と出口を、ループ名と終了条件の書いた記号ではさむことで表現する。いかにループ記号と、その使用例を示す。



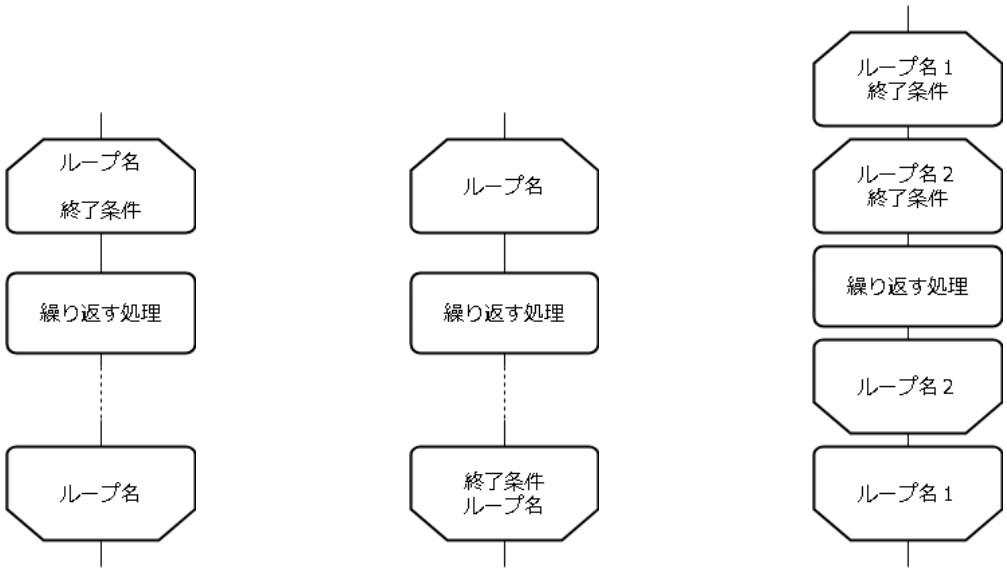
(a) ループ端



(b) 判断記号を使った繰り返し処理とループ端を使った繰り返し処理

図 1: ループ端を使った繰り返し処理

当然、前判定、後判定、多重ループも同様に表現できる。しかも判断記号を使った繰り返しよりもシンプルに表現できる場合が多い。処理の内容によって、Yes,No の表記が明示的に示されているほうがわかりやすい時もあるので、判断記号を使った繰り返しも捨てたものではないので、場合によって描き分けられるのが望ましい。以下に、前判定、後判定、多重ループの記述例を示す。



(a) 前判定型ループ　**(b)** 後判定型ループ　**(c)** 多重ループ

図 2: ループ端のバリエーション（1）

また、カウンタ変数を使った繰り返しのような、一定回数行う繰り返しも単純化して書くことができる。カウンタに対する、初期値、継続・終了条件、カウンタの増減値をループ端に記述する。前判定の場合は、上部のループ端に、後判定の時は下部のループ端に記述する

表 1: カウンタ型の繰り返し

初期化式	繰り返し処理の前に、カウンタ変数の値の**初期値**を決定する処理。カウンタの開始する値が決まる
継続条件式	繰り返しを継続する条件を書く
終了条件式	繰り返しを終了する条件を書く
再初期化式	カウンタの値を増減させる式を書く（インクリメント、デクリメント）

練習問題

以下に示す処理のフローチャートをループ端を使って書け。型、変数名、ループ名などは、適宜適切なものを考える。（以前の問題と同じなので、判断記号を使った繰り返しと比べながら書く）

1. １～１０までの整数が奇数か偶数かを判別し表示する。
   例：1 →奇数、2 →偶数、3 →奇数....
2. ０～５０までの奇数の合計を求め表示する。
3. １～９の整数を入力して、その段の九九を表示する処理。
   例：３を入力→ 3, 6, 9,12,15,18,21,24,27
4. １～９の段までの九九表を表示する処理。
   例：

```
 1  2  3  4  5  6  7  8  9
 2  4  6  8 10 12 14 16 18
 3  6  9 12 15 18 21 24 27
 4  8 12 16 20 24 28 32 36
 5 10 15 20 25 30 35 40 45
 6 12 18 24 30 36 42 48 54
 7 14 21 28 35 42 49 56 63
 8 16 24 32 40 48 56 64 72
 9 18 27 36 45 54 63 72 81
```

1.2　疑似コード表現（Pseudo Code）

疑似コードは、アルゴリズムを人間の言語になるべく近い疑似的なプログラミング言語で表したものである。いわば、なんちゃってプログラミング言語で書かれた、似非コードのようなものである。疑似コードにも様々な流派があり、教科書には、基本情報技術者試験の教科書基準の疑似コードの書き方が載っている。

```
1   ○整数型関数：Init(整数型:S[],整数型: N,整数型: K)
2   ○整数型: L
3   ▲ 1 ≦ K and K ≦ N
4   | ■ L: 1, L ≦ N, 1
5   | | ▲ L ≦ K
6   | | | ・ S[L] ← 1
7   | | +---------------
8   | | | ・ S[L] ← 0
9   | | ▼
10  | ■
11  | ・  return 0
12  +-------------
13  | ・  return 1
14  ▼
```

ソースコード 1: 基本情報技術者試験基準の疑似コード

Algorithm 1 Algorithm for IEEE papers

Input: in

Output: out

Initialisation :

1: first statement

LOOP Process

2: **for** $i = l - 2$ to 0 **do**

3:　statements..

4:　**if** $(i \neq 0)$ **then**

5:　　statement..

6:　**end if**

7: **end for**

8: **return** P

練習問題

以下に示す処理を疑似言語使って書け。型、変数名、処理名などは、適宜適切なものを自由に考えて使って良い。データの入出力は、以下の関数を用いること

○入力 *(data)*

data にキーボードから値を入力

○出力 *(data)*

data を画面に出力する。（自動的に型によって適切な形で出力する機能がある。改行はしない）

○改行 *()*

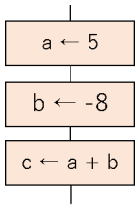
改行文字を出力する。

1. １～１０までの整数が奇数か偶数かを判別し表示する。
   例：1 →奇数、2 →偶数、3 →奇数....
2. ０～５０までの奇数の合計を求め表示する。
3. １～９の整数を入力して、その段の九九を表示する処理。
   例：３を入力→ 3, 6, 9,12,15,18,21,24,27
4. １～９の段までの九九表を表示する処理。
   例：

```
01 02 03 04 05 06 07 08 09
02 04 06 08 10 12 14 16 18
03 06 09 12 15 18 21 24 27
04 08 12 16 20 24 28 32 36
05 10 15 20 25 30 35 40 45
06 12 18 24 30 36 42 48 54
07 14 21 28 35 42 49 56 63
08 16 24 32 40 48 56 64 72
09 18 27 36 45 54 63 72 81
```

フローチャート、疑似言語とプログラミング言語

以下に、継続条件と終了条件のフローチャート記述例を示す



順次構造

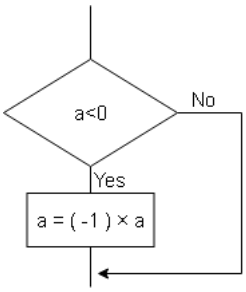
```
1    int main()
2    {
3      int a,b,c;
4      a = 5;
5      b = -8;
6      c = a + b;
7    }
```

ソースコード 2: 対応する C++ ソースコード

図 3: 順次構造のフローチャートと C++ ソースコード

選択構造（条件駆動）

ある条件が成立（不成立）の場合にだけ処理を実行したい時の選択構造。if 文で真、又は偽のときの処理のみ書くことにより実現する。



選択構造１（Yes（No）の時のみ実行

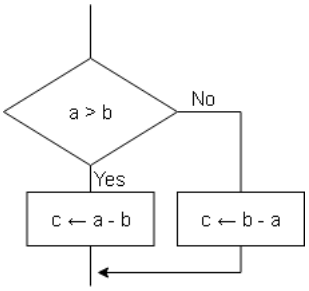
```
1    if(a < 0)
2    {
3        a = (-1) * a;
4        //a = -a; //これでもいいよ
5    }
```

ソースコード 3: 対応する C++ ソースコード

図 4: 選択構造のフローチャートと C++ ソースコードその１

選択構造（２分岐）

選択構造の、条件判断で真の時の処理、偽の時の処理に処理が分岐する構造。if else 文により実現する。



選択構造２（Yes、No で別の処理を実行）

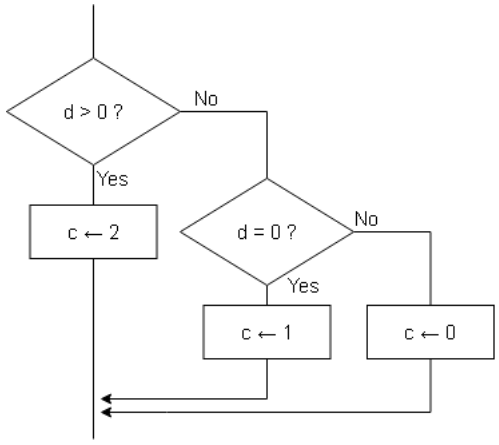
```
1    if(a > b)
2    {
3        c = a - b;
4    }
5    else
6    {
7        c = b - a;
8    }
```

ソースコード 4: 対応する C++ ソースコード

図 5: 選択構造のフローチャートと C++ ソースコードその２

条件判断の入れ子構造

選択構造の分岐に、また、選択構造が現れる構造。C++ では、if else if 文で表すことができる。または、if 文の中にまた if 文を書く。



選択構造３（No の時にさらに選択があるパターン（else if パターン））

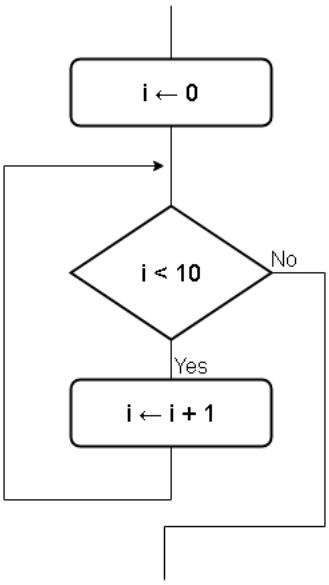
```
1   //if~else if文
2     if(d > 0)
3     {
4         c = 2;
5     }
6     else if(d == 0)
7     {
8         c =  1;
9     }
10    else
11    {
12      c = 0;
13    }
14  //if文の中にif文を書く（入れ子）
15    if(d > 0){
16      c = 2;
17    }else{
18      if(d == 0){
19        c = 1;
20      }else{
21        c = 0;
22      }
23    }
```

ソースコード 5: 対応する C++ ソースコード

図 6: 選択構造のフローチャートと C++ ソースコードその３

前判断型繰り返し構造

前判断型繰り返し構造は、while 文、又は for 文で実現できる。カウンタを使う場合は、for 文を使った方がすっきりと書ける場合が多い。



繰り返し構造１ (前判定)

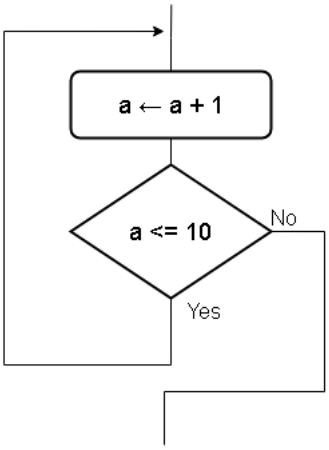
```
1   //while文バージョン
2   i = 0; //カウンタ変数と初期化
3   while(i < 10)
4   {
5     //ここに繰り返す処理を書く
6     i++;
7     //i = i + 1;
8   }
9   //for文バージョン
10  for(int i = 0; i < 10; i++)
11  {
12    //ここに繰り返す処理を書く
13  }
```

ソースコード 6: 対応する C++ ソースコード

図 7: 繰り返し構造のフローチャートと C++ ソースコードその１

後判断型繰り返し構造

後判断型繰り返し構造は、do while 文を使って実現できる。後判断型では、繰り返したい実行文→条件の判定、の順に処理が行われるため、必ず１度は繰り返したい実行文が処理される。（前判断型では、条件次第で１度も繰り返したい実行文を処理せずループを抜ける場合がある。



繰り返し構造２ (後判定)

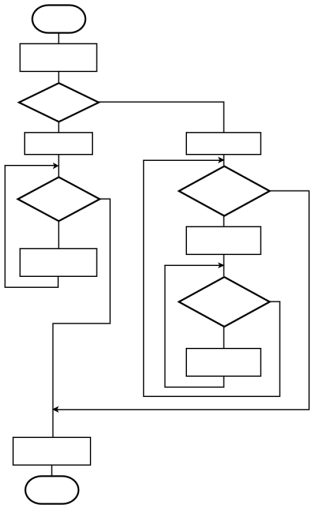
```
1
2   do
3   {
4     //ここに繰り返す処理を書く
5     a = a + 1;
6   }while(i <= 10);
```

ソースコード 7: 対応する C++ ソースコード

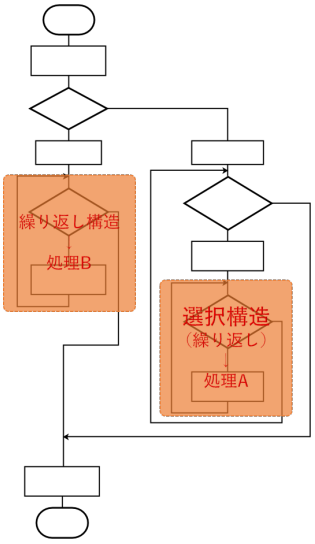
図 8: 繰り返し構造のフローチャートと C++ ソースコードその２

構造化プログラミング

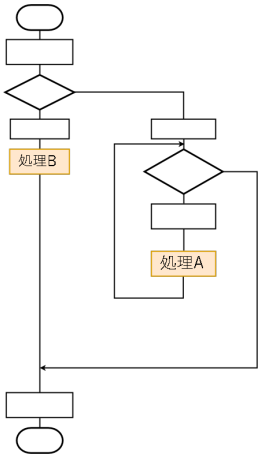
順次、選択、繰り返し構造を組み合わせて、複数の処理をまとまった処理としてまとめ、複雑な処理を分割して構成してゆく手法のことを構造化プログラミングと呼ぶ。逆に言うとどんな複雑な処理もこの３つの処理の組み合わせて表現できると言うことが言える。（いえない時もあるよ）



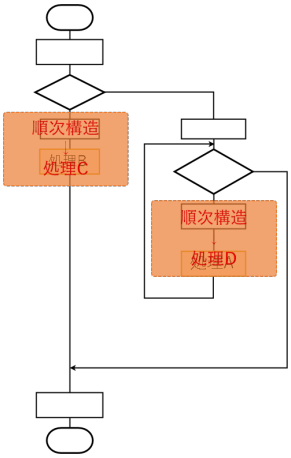
(a) 複雑なフローチャート：一見複雑なこのようなフローチャートが、実は順次構造、選択構造、繰り返し構造の組み合わせで、構成されていることを確認してみよう



(b) 処理の分割１：囲みの２つの繰り返し構造をまとまった処理 A、B としてまとめてみる

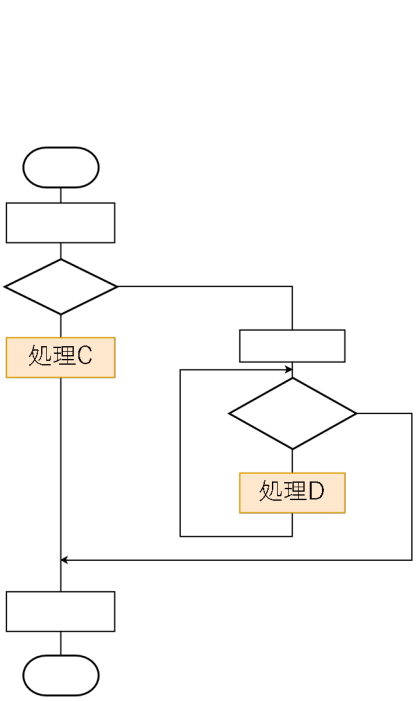


(c) 処理の分割２：処理 A、処理 B を一つの処理とみなすと、さらに順次構造としてまとめられそうな部分を見つけることができる。

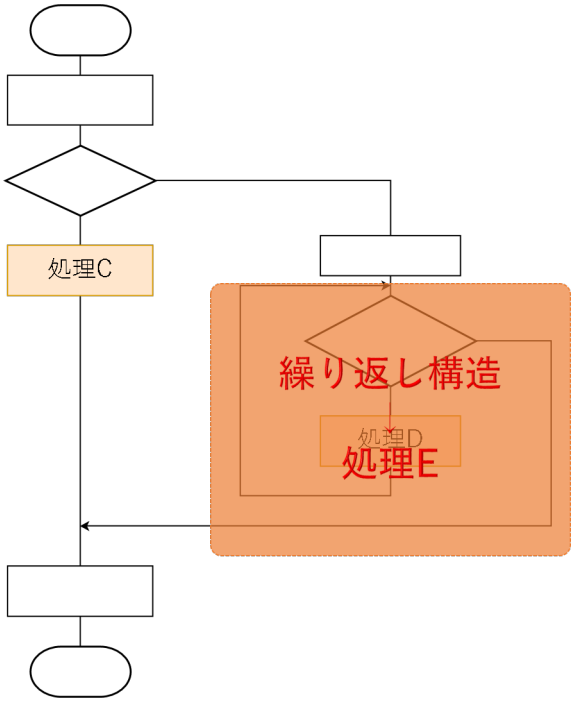


(d) 処理の分割３：処理 A、処理 B とその直前の処理を、一つの順次構造とみなし、処理 C、処理 D としてまとめてみる

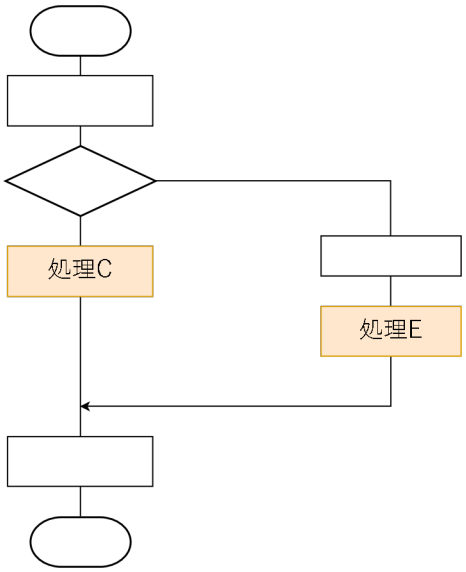
図 9: 構造化プログラミング（処理の分割）１



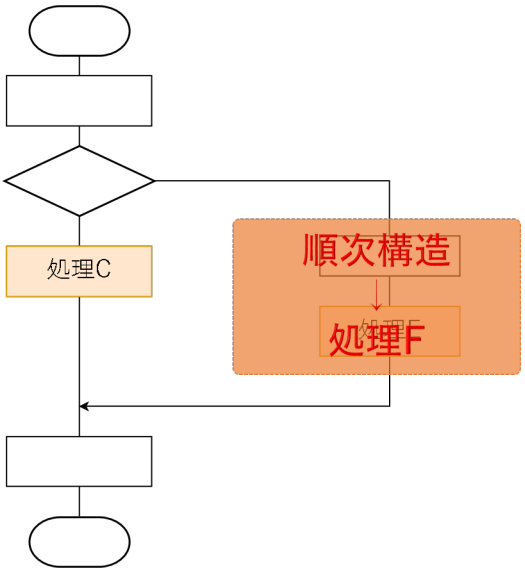
(a) 処理の分割 4：処理 C 付近、処理 D 付近に標準型の選択構造、繰り返し構造が現れる



(b) 処理の分割 5：処理 D 付近の繰り返し構造をまとめて処理 E としてフローチャートを書いてみる

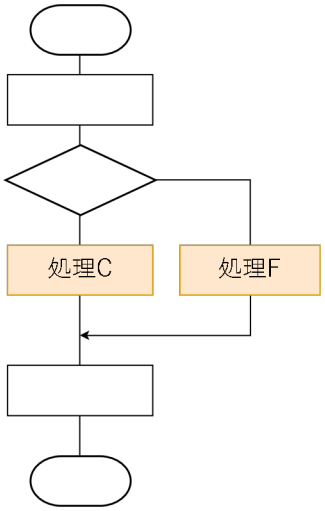


(c) 処理の分割 6：処理 E 付近を一つの順次構造とみなすことができそうな気がしてくる

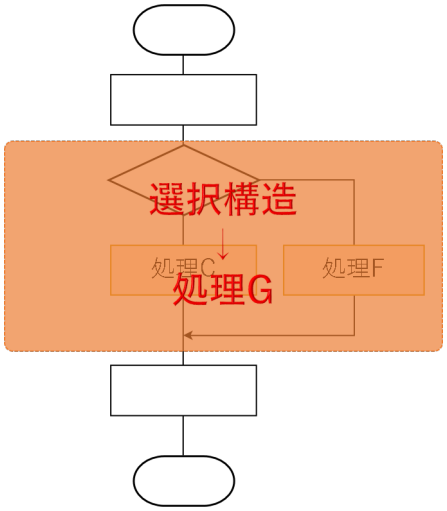


(d) 処理の分割 7：処理 D 付近を一つの順次構造（処理 F）としてまとめてみる

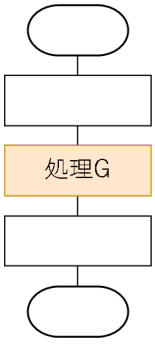
図 10: 構造化プログラミング（処理の分割） 2



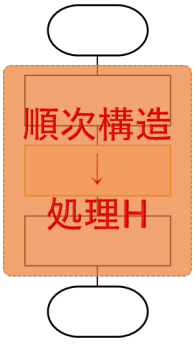
(a) 処理の分割８：処理 C、処理 F を含めて付近の選択構造を見てみると、単純な２分岐の選択構造が見つけられる



(b) 処理の分割９：処理 C、処理 F を含む選択構造を処理 G としてまとめてみる



(c) 処理の分割１０：全体が一つの順次構造として表すことができる



(d) 処理の分割１１：最終的にきれいに一つの順次構造としてあらわすことができるようになる

図 11: 構造化プログラミング（処理の分割） 3