整理術としてのオブジェクト指向入門 その２

～どうしてオブジェクト指向なのか？～

森　博之（ひろえむ）

Microsoft MVP for Visual Developer – Visual C#

-ひろえむの日々是勉強-

http://blogs.wankuma.com/hirom/

Agenda

- 前回のおさらい。
- どうしてオブジェクト指向なんだろう？
- 次回予告

前回のおさらい

- オブジェクト指向って難しい？
  →否！　わかってしまえば難しくない！
- オブジェクト指向の教え方が難しい
  →「チャーハンの食べ方を教えて？」
- オブジェクト指向を誤解していない？
  →ソフトウェアで実現したいことが
  　なくなるワケじゃない！

どうしてオブジェクト指向が必要になったの？

・ソフトウェアってなんだろう？

→コンピュータを制御するプログラム

→物理的装置であるハードと対比。

コンピュータって？

- ５つの装置

入力装置（キーボード・マウス）
出力装置（ディスプレイ・プリンタ）
記憶装置（ハードディスク・メモリ）
制御装置（マザーボード）
演算装置（ＣＰＵ）

機械語

Ex) C000 F0 CB AF ED 3E

- 命令とパラメータの組み合わせ
- 命令は１６進数の数値に割り当て
- パラメータも１６進数の数値

つまり・・・

１６進数だらけ！

アセンブリ言語・ニーモニック

Ex)

```
MOV AX, X
MOV DX, Y
ADD AX, DX
MOV Z, AX
```

- 機械語に比べれば若干・・・
- 単純で簡単。
- 高速かつ無駄が少なくなる。
- 命令の種類は少ない・制限も多い

高級言語

Ex)FORTRAN , COBOL, C, BASIC

FORTRAN) Z = X + Y

より自然言語に近い

演算・制御も方程式などに近い！

わかりやすい！

→コンピュータを動作させるためのハードルが下がる！

ハードルが下がることで・・

1．プログラミングが楽になる
2．低次元で考えていたことに時間を割かなくてよくなる。
3．より一層、問題に注力できる。

→そうすると新たな問題が・・・。

高級言語で問題が！？

- 高級言語で表現してもわかりづらい！

→機械語にくらべればわかりやすいはずなのに・・・。

- プログラムが複雑になってきた！！

→いわゆるgotoを使って、構造が複雑に！

複雑になったプログラム

```
10 PRINT “W”
13 I = 0
15 GOTO 40
20 PRINT “N”
25 I = 1
25 PRINT “K”
30 PRINT “U”
35 PRINT “M”
40 PRINT “A”
45 IF I = 0 THEN GOTO 25 ELSE GOTO 50
50 END
```

構造化プログラミングの登場

- プログラムを複雑化しているのは

ＧＯＴＯ命令！！！

いわゆるスパゲッティプログラム

→これらのコードは

順次進行・条件分岐・繰り返し

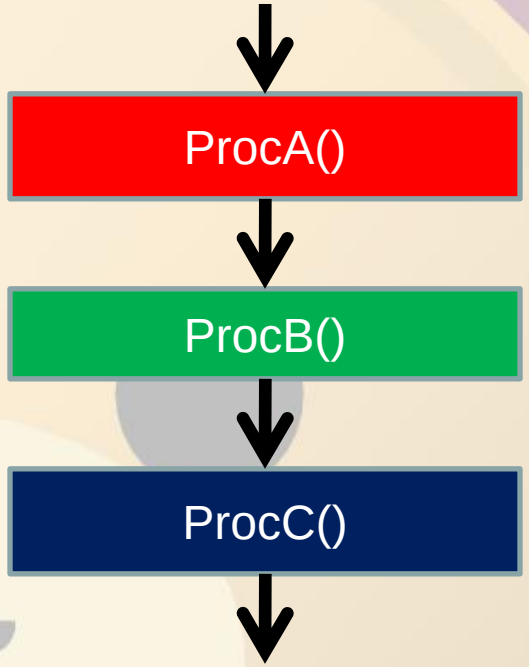
(基本３構造)

の３つの構造で表現できる。

順次進行

C#

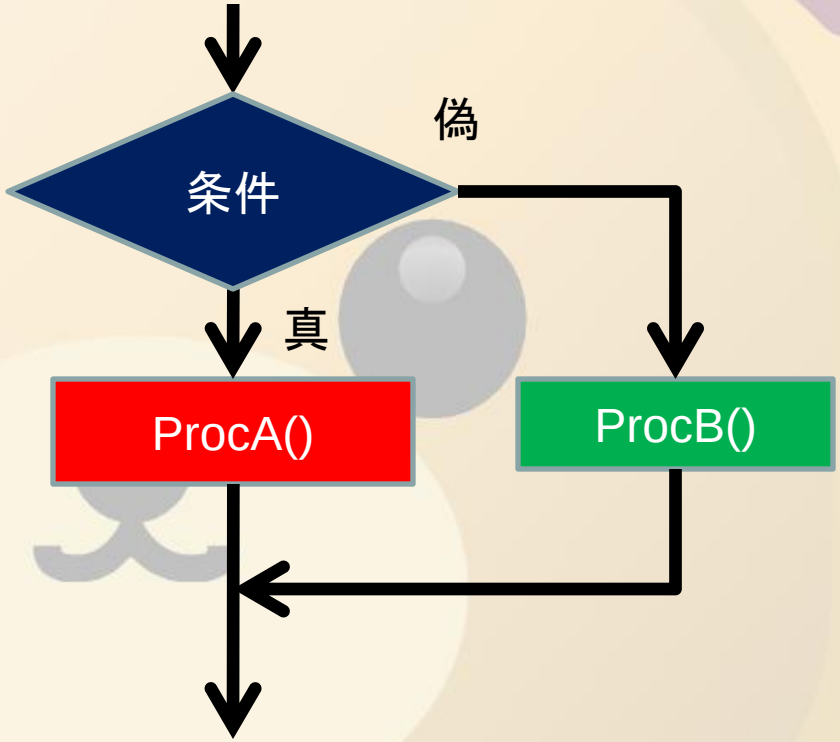
```
void main()
{
    ProcA();
    ProcB();
    ProcC();
}
```



条件分岐

C#

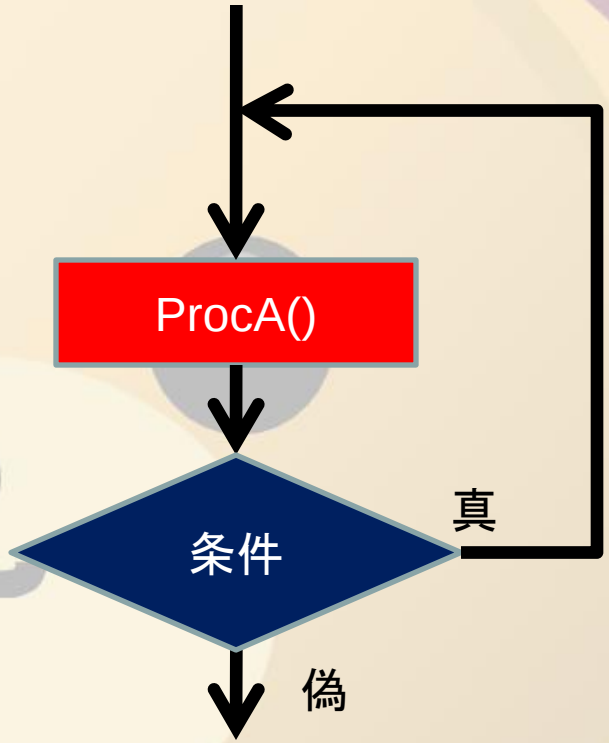
```
void main()
{
    if(条件) {
        ProcA();
    }
    else {
        ProcB();
    }
}
```



繰り返し

C#

```
void main()
{
    do {
        ProcA();
    } while(条件);
}
```



サブルーチン

- プログラムの複数場所に現れる同じ命令

→１カ所に集めちゃえ！

それだけでは・・・

→サブルーチンの独立性を高める必要がある

→メインルーチンとサブルーチンで共有する情報を少なくする

共有する情報

- 簡単にいうと変数
- 複数のサブルーチンで共有する情報

→グローバル変数

プログラム・ロジック

・順次追いかけると解読ができる変数

・どこで参照されているかひと目で判断することが難しい

グローバル変数がたくさん定義されると

- プログラムのメンテナンスが困難！

  →どこで参照されているか
  　どこに影響があるか調べるために
  　ソースコードをあちこち調べないと！

- そんなために考えられた仕組み、それが・・

ローカル変数・引数値渡し

- ローカル変数
  →サブルーチンの内部のみ有効な変数
  →サブルーチンに入った時に生成し
  　サブルーチンから出る時に消滅
- 引数値渡し
  →サブルーチンへ渡すパラメータを呼び出し元の値を利用せず、コピーしたものを渡す

構造化言語！

- ALGOL,Pascal,C言語など・・・
- If・case・while・forなどを利用して明確な制御構造が記述できるようになった！
- つまり、基本３構造が素直に表現できる言語が登場！
- ・・・でもGoTo文（もしくは類似命令）もあったけどね(^^;

C言語！

- 今となっては構造化言語の代名詞！

1．構造化プログラミング機能のサポート
2．アセンブリ言語のようなビット演算やメモリ領域を効率的に利用できるポインタなどの機能
3．プログラミングの機能を言語だけで提供せず、関数ライブラリでくみ上げるようにした！

言語の進化の方向性

手軽なプログラミングを・・・

機械語～アセンブラ

コンピュータにやらせたいことを
人間の親しみやすい方法で表現・・

高級言語

保守性を高めるための進化・・・

構造化プログラミング言語

機械語～アセンブラ

- ハードウェアではなく、ソフトウェアで。
- プログラミングにより自由でかつ複雑な計算を行える！

高級言語

- １６進数の羅列や制限のある命令群だけではなく、人間が親しみやすい言語へ
- おおよそ、目標は達成された！？

構造化プログラミング

- 保守性を高める必要がある！？
- 再利用されることが多い！
- プログラムの寿命が長い！
- ２０００年問題など！

進化の方向性

- 生産性向上
  →命令を簡単にする
- 保守性向上
  →プログラムをわかりやすくするため
- 品質向上
  →制約をつけて複雑さを避ける
- 再利用性促進
  →重複ロジックを排除し、独立したサブルーチンを再利用！

ここまでで解決できたこと・残された課題

- 解決！
  1．GOTO文の乱用によるスパゲッティの回避
  2．共通サブルーチンによる再利用

- 課題！
  1．グローバル変数！
  2．まだまだな再利用

グローバル変数

・ローカル変数ではサブルーチン内のみで利用できるけど・・・

→永続化しないといけない情報は少なくない！

・どうしても増えてしまうグローバル変数。

→保守性を落とす結果に！

再利用がまだまだなのはなぜ？

- サブルーチン同士の依存性の高さ
- 大きなサブルーチン
- 共通で利用する変数の多さ

そこで登場オブジェクト指向！

- これらの問題点をなんとか解決する方法はないものかしら！！
- グローバル変数・再利用性は・・・
- 解決するための３つの道具。

３つの道具

- クラス（抽象化・情報隠蔽）
- 継承
- 多態性（ポリモーフィズム）

３つの道具をどう使うか

- きちんと整理整頓
- 整理整頓されているから必要なロジックへの到達が簡単！
- 関連性の強いグローバル変数とロジックをまとめて１つにする仕組み

じゃぁ、具体的にどうすればいいの？

それは次回をお楽しみに！

ご静聴ありがとうございました。

長時間お疲れ様でした。