第三回

直行性

互いに影響を及ぼさず、独立にコントロールできるということ。

PICの例

PICで以下のようなサウンドポートのHigh/Lowのきっりかえを一定の回数行う関数を組んだとする

```
void ring(int t) {
  for(i = 0; i < 1000; i++) {
    BIT_SET(*PORTA, 3);
    delay_us(t);
    BIT_CLEAR(*PORTA, 3);
    delay_us(t);
  }
}
```

ソフトウェアを組む時`ring()`関数のような関数が基本的な部品として働いている。このような部品のことをコンポーネントという。

この場合音程によってこの関数を呼び出す関数を調整しなければならないため、直行しているとは言えない。直行しているシステムにするためには、サウンドポートをHighとLowに切り替える回数を音程に合わせて調整するような関数にするべきである。

こうすることによって音程を高くする機能や低くする機能がお互いに影響を及ぼさない。

直行度

- 結合度が低いこと。
- 凝集度が高い(部品の中はつながりあっている)

直行度を高くすると、

- 生産性の向上
- リスクの低減

の二つのメリットを享受できる。

またシステム間の結合が疎であればあるほど再編成、再作成が可能

直行しているコンポーネント同士を組み合わせることで最小限の努力でより多くの機能が手に入る。

直行原則を適用する方法

直行性のあるソフトウェアは依存が少ないため

- コードの結合を最小化(ほかの実装をあてにしない)

- オブジェクトの変更はオブジェクト自身にさせる
- グローバル変数を避ける
    - 前に何をして後で何をするかを明示的に引き渡す
- 類似機能を避ける
    - 関数化を理由してコードの二重化を避ける。

直行性との付き合い方

- DRY原則と密接にかかわる
- DRY原則と直行性は組み合わせる

直行性のないソフトウェアに遭遇したら極力直行性を損なわないようにコードを書く(割れ窓理論)

リスクの削減

- コードの問題発生部分の隔離
- デバッグの容易性が高まる。

曳光弾

1. **仮説**に基づいてアプリケーションのフレームワークを作成
2. フレームワークに肉付け
3. うまくいかなければ必要に応じて仮説を修正。1に戻る
4. アプリケーションソフトウェアをそのまま運用

プロトタイピングとの違いは、初めに造ったものを捨てるか捨てないか。 力技で組んだプロトタイプは、仮説を立てたら捨てるもの

プロトタイプを作ることを言われたときは、「曳光弾」か「プロトタイピング」かということを聞く。知らなかったら**絶対に説明するべき。** 世の中のソフトウェアの炎上はこれで起こっている。

アジャイルソフトウェア開発は仮説がたってからである。試行錯誤はプロトタイピングでやるべし。

実験Ⅰ　改良

改良前のプログラムは以下。

```
void main()
{
    INT_IO();    // Call INT_IO() subrutines for INITIALIZE PORTs.
    int pre0 = 0 , pre1 = 0 , pre2 = 0 ,pre4 = 0, pre5 = 0,pre6 = 0;
    int trig = 0;//0:Time setting 1:count time
    int count = 0 ,minute = 0;
    //time setting
    while(1)
    {
        OUTPUT_B(count);
        //minute lighter
        if(minute == 1){BIT_SET(*PORTB,6);}
        if(minute == 2){BIT_SET(*PORTB,7);}
        if(minute == 3)
```

```
        {
            BIT_SET(*PORTB,6);
            BIT_SET(*PORTB,7);
        }
        while(trig == 1)
        {
            //counting
            OUTPUT_B(count);
            //minute lighter
            if(minute == 1){BIT_SET(*PORTB,6);}
            if(minute == 2){BIT_SET(*PORTB,7);}
            if(minute == 3)
            {
                BIT_SET(*PORTB,6);
                BIT_SET(*PORTB,7);
            }
            //stop button
            if(BIT_TEST(*PORTA,3) == 0){trig = 0;}
            //minute -1 or end of timer
            if(count == -1)//count -1 to avoid timer end first
            {
                //end of timer
                minute = minute - 1;
                if(minute == -1)
                {
                    //minute = -1 to avoid end without second = 0;
                    OUTPUT_B(255);
                    delay_ms(1000);
                    OUTPUT_B(0);
                    delay_ms(1000);
                    count = 0;
                }
                //1 minute -> 59 second
                else{count = 59;}
            }
            //Prepare of next
            delay_ms(1000);
            count = count - 1;
            //To The Next
        }
        //start button
        if (BIT_TEST(*PORTA,0)== 0)
        {
            if (pre0 == 0)
            {
                if (trig == 0){trig = 1;}
                else {trig = 0;}
            }
            pre[0] = 1;
        }
        else{pre0 = 0;}
            //time setting
            //count+1 button(second + 1)
        if(BIT_TEST(*PORTA,1)== 0)
```

```
        {
            if(pre1== 0){
                count = count + 1;
                trig = 0;
                }
            pre1 = 1;
        }
        else{pre1 = 0;}
        //count-1 button (second + 1)
        if(BIT_TEST(*PORTA,2)==0)
        {
            if(pre2 == 0)
            {
                trig = 0;
                count = count - 1;
            }
            pre2 = 1;
        }
        else{pre2 = 0;}
        //minute +1 button
        if(BIT_TEST(*PORTA,4)==0)
        {
            if(pre4 == 0)
            {
                trig  = 0;
                minute = minute + 1;
            }
            pre4 = 1;
        }
        else{pre4 = 0;}
        //minute -1 button
        if(BIT_TEST(*PORTA,5)==0)
        {
            if(pre5== 0)
            {
                trig  = 0;
                minute = minute - 1;
            }
            pre5 = 1;
        }
        else{pre5 = 0;}
        //reset button
        if(BIT_TEST(*PORTA,6)==0)
        {
            if(pre6 == 0)
            {
                trig = 0;
                minute = 0;
                count = 0
            }
            pre6 = 1;
        }
        else{pre6 = 0;}
        //minute + 1
```

```
        if(count == 60)
        {
            minute = minute + 1;
            count = 0;
        }
        //regulate minutes
        if(minute == 4){minute = 3;}
    }
}
```

このプログラムに改善を加え、

- 関数化やSwitch構文の応用で二重化を撤廃
- 関数に明確な役割を与えることによる直交性の向上

を実現した。またこれによって変更容易性が格段に向上した。それが以下である。

```
void LED_blink(value,delay_time){
    OUTPUT_B(value);
    delay_ms(delay_time);
    OUTPUT_B(0);
    delay_ms(delay_time);
}

void end_control(int *minute, int *count){
    if(*count == -1){
        //end_of_timer
        *minute -= 1;
        if(*minute == -1)
        {
            LED_blink(255,1000);
            *count = 0;
        }
        //1 minute -> 59 second
        else{*count = 59;}
    }
    //prepare of next
    delay_ms(1000);
    count -= 1;
}

void button_control(int *pre,int pre_length,int *trig,int *minute, int *count)
{
    for(int i = 0; i < pre_length, i++){
    //pre3は使用しないため
    if(i == 3){break;}
    if(BIT_TEST(*PORTA,i) == 0){
        if(pre[i] == 0){
            if(i != 0){*trig = 0}
            switch(i){
                case 0:
                    if (*trig == 0){*trig = 1;}
```

```
                    else {*trig = 0;}
                case 1:
                    *count++;
                case 2:
                    *count--;
                case 4:
                    *minute++;
                case 5:
                    *minute--;
                case 6:
                    //reset
                    *minute = 0;
                    *count = 0;
                default : break;
            }
        }
        pre[i] = 1;
    }
    else {pre[i] = 0}
    if(*count == 60)
    {
        *minute++;
        *count = 0;
    }
    //regulate minutes
    if(*minute == 4){*minute = 3;}
    }
}

void main(){
    INT_IO();    // Call INT_IO() subrutines for INITIALIZE PORTs.
    int pre[] = [0,0,0,0,0,0,0]
    int trig = 0;//0:Time setting 1:count time
    int count = 0 ,minute = 0;
    {
        Program_Start:
            OUTPUT_B(minutes * 64 + count)
            while(trig == 1)
            {
                OUTPUT_B(minutes * 64 + count);
                end_control(&minute,&count)
            }
        //Button_control
        int pre_length = size(pre) / size(pre[0])
        button_control(pre,pre_length,&trig,&minute,&count);

        goto Program_Start
    }
}
```

予想問題

Q1. K君は将棋ソフトウェアを開発しようと考えています。が、右も左も分からず見通しが立たない状況です。どのようなアプローチをとるべきでしょうか？

A1. コンポーネント単位でプロトタイピングを行い、仮説を立てた後でフレームワーク設計を行う。途中で詰まった場合はフレームワーク設計に戻る。部品の設計が終わったらテストを行う。その際は直行性テストも重要である。

Q2. K君は将棋ソフトウェアのプロトタイプを完成させたものの、プロトタイプのコードがめちゃくちゃです。またバグも残っています。最適なアプローチを提案しなさい。

A2. プロトタイプの機能を仮説をもとに分け、どのオブジェクトにどの機能を割り振るかを再度検討する。その上でそのプロトタイプは廃棄し、一からソフトウェアを作り直す。