

8章： 事実の観察

ANDREAS ZELLER 著 中田秀基 監訳

デバッグの理論と実践

-なぜプログラムはうまく動かないのか-

学籍番号 60206299

氏名 吉富 楓雅

自動デバッグテクニック(再掲)

今後の章の紹介

- ・入力の単純化
- ・プログラムスライス
- ・状態の観察
- ・状態の監視
- ・アサーション
- ・異常
- ・原因と結果の連鎖
- ・ミスから学ぶ



目次

1. 状態の観察と出力
2. printfデバッグを克服する
3. デバッガの使用
4. 状態の可視化
5. まとめ

今回は
「状態の観察・監視」
についてです

状態の観察と出力



プログラムの観察

- ・演繹は何か起こりうるかという予測しかできない

実際... 事実を見る



何が起きているのかを観察し
値が感染しているかどうかを判断
しなければならない

観察の原則

- ・ 干渉しない

プログラム実行の影響だけを観察すべき

- ・ 何をいつ観察するかを把握

プログラム実行とは膨大なプログラム状態の長い連続

- ・ 系統的に進める

やみくもに値を観察するのではなく科学的手法に従って調べる

printfデバッグ

```
#include <stdio.h>
int main(void)
{
    int i;
    int num;
    printf("Input Base number : ");
    scanf("%d", &num);
    for (i=0; i<19; i++) {
        printf("DEBUG : i = %d\n", i);
        printf("%d\n", num * i);
    }
    return (0);
}
```

- ・printデバッグと呼ばれる有名な観察テクニック

- ・デバッグログ:

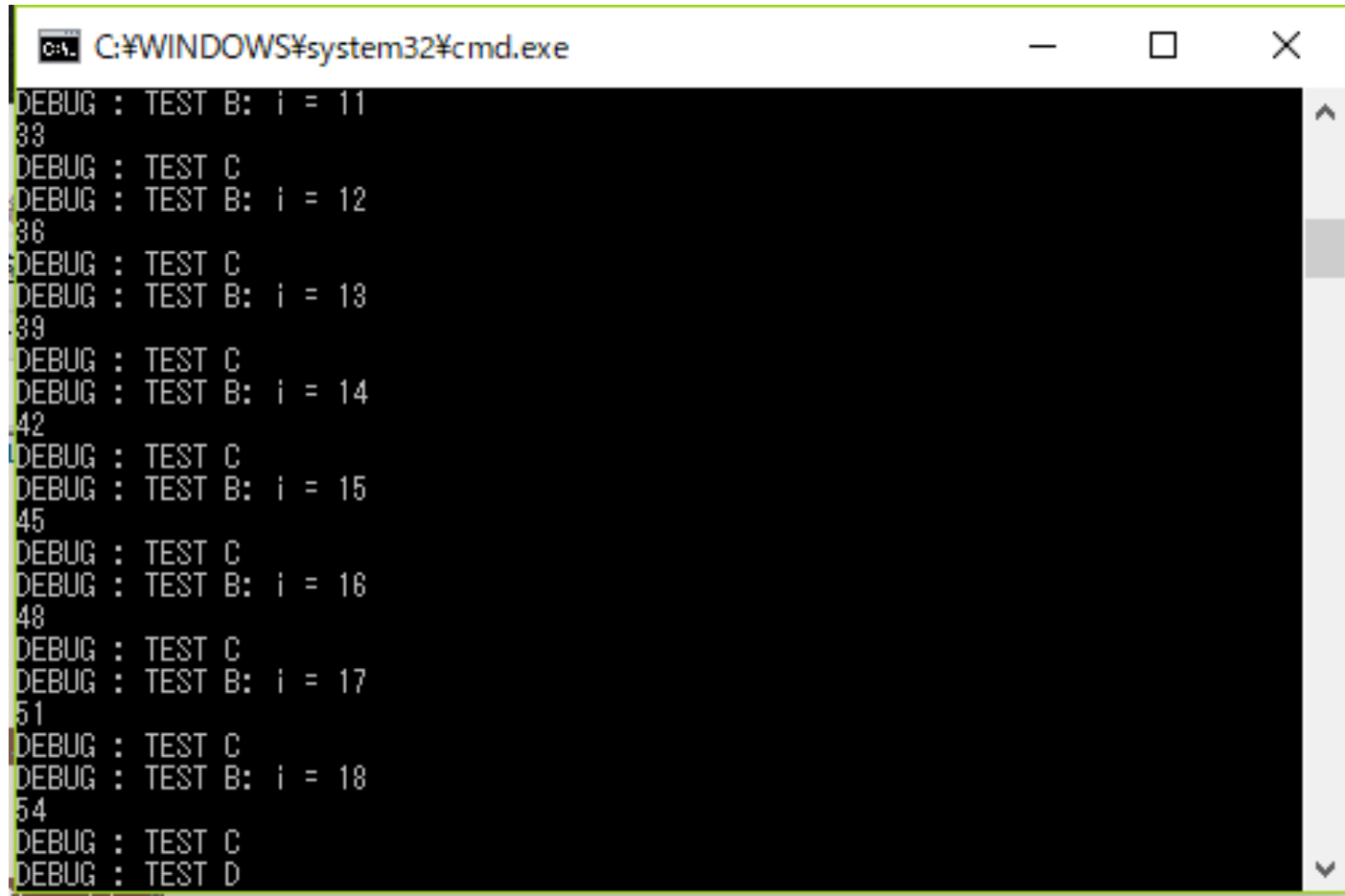
出力が複数まとまったもの

- ・特別な仕組みを必要としない

printfデバッグは手軽でも...

```
#include <stdio.h>
int  main(void)
{
    int i;
    int num;
    printf("Input Base number : \n");
    printf("DEBUG : TEST A\n");
    scanf("%d", &num);
    for (i=0; i<19; i++) {
        printf("DEBUG : TEST B: i = %d\n", i);
        printf("%d\n", num * i);
        printf("DEBUG : TEST C\n");
    }
    printf("DEBUG : TEST D\n");
    return (0);
}
```

実行結果



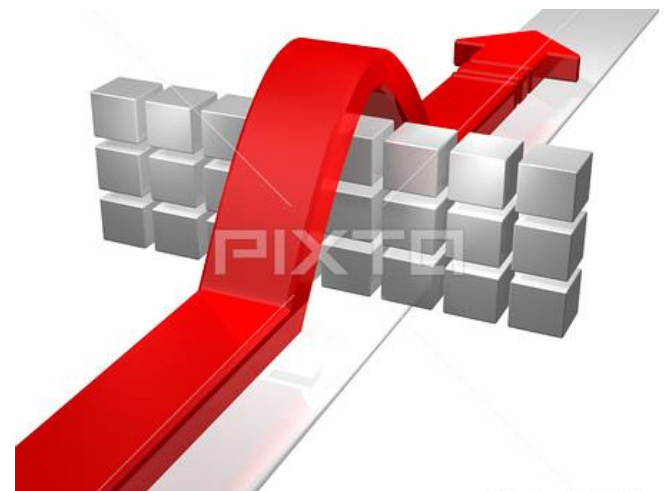
A screenshot of a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The window has a black background with white text. It displays a series of debug messages in a loop. The messages are: "DEBUG : TEST B: i = 11", "33", "DEBUG : TEST C", "DEBUG : TEST B: i = 12", "36", "DEBUG : TEST C", "DEBUG : TEST B: i = 13", "39", "DEBUG : TEST C", "DEBUG : TEST B: i = 14", "42", "DEBUG : TEST C", "DEBUG : TEST B: i = 15", "45", "DEBUG : TEST C", "DEBUG : TEST B: i = 16", "48", "DEBUG : TEST C", "DEBUG : TEST B: i = 17", "51", "DEBUG : TEST C", "DEBUG : TEST B: i = 18", "54", "DEBUG : TEST C", and "DEBUG : TEST D". The window has a standard Windows title bar with minimize, maximize, and close buttons.

```
C:\WINDOWS\system32\cmd.exe
DEBUG : TEST B: i = 11
33
DEBUG : TEST C
DEBUG : TEST B: i = 12
36
DEBUG : TEST C
DEBUG : TEST B: i = 13
39
DEBUG : TEST C
DEBUG : TEST B: i = 14
42
DEBUG : TEST C
DEBUG : TEST B: i = 15
45
DEBUG : TEST C
DEBUG : TEST B: i = 16
48
DEBUG : TEST C
DEBUG : TEST B: i = 17
51
DEBUG : TEST C
DEBUG : TEST B: i = 18
54
DEBUG : TEST C
DEBUG : TEST D
```

printfデバッグの欠点

- ・コードがごちゃごちゃする
- ・出力がごちゃごちゃする
- ・処理速度が低下する
- ・データの損失

printfデバッグの克服



pixta.jp - 1292544

printfデバッグにおける欠点 の克服法

- ・**アスペクト**を使ったログ出力

アスペクト指向を用いてごちゃごちゃを解決する

- ・**バイナリレベル**でのログ出力

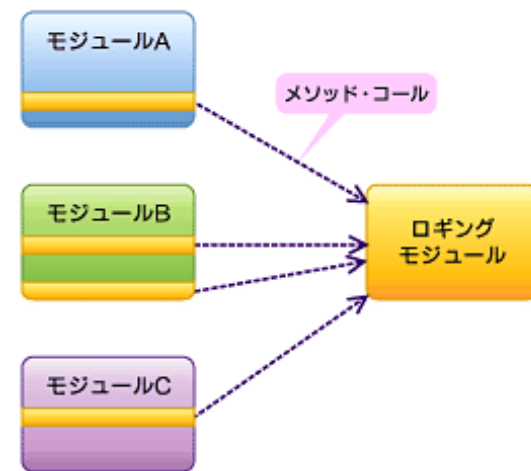
アスペクトが使えない場合にも対応できる

参考：アスペクト指向とは

アスペクト指向：Aspect Oriented Programming

- ・オブジェクト指向の補完版
- ・ソフトウェアに存在する「横断的関心事」を分離させることが目的
- ・散在した「横断的関心事」はソフトウェアの保守性を妨げる

今後発展する可能性がある！



横断的関心事

Javaにおけるログ出力

今回ログを取りたいコード

```
class Article {  
    public void buy() {  
        // コード  
    }  
}
```

Javaにおけるログ出力

```
class Article {  
    public void buy() {  
        System.out.println("Entering Article.buy()");  
        original_buy();  
        System.out.println("Leaving Article.buy()");  
    }  
    public void original_buy() {  
        // Article.buy() の元コード  
    }  
}
```

アスペクトを使ったログ出力

今回ログを取りたいコード

```
class Article {  
    public void buy() {  
        // コード  
    }  
}
```

AspectJにおけるログ出力

```
public aspect Logby{  
    pointcut buyMethod() :  
        call(public void Article.buy());  
    before(): buyMethod() {  
        System.out.println("Entering Article.buy()")  
    }  
    after(): buyMethod() {  
        System.out.println("Leaving Article.buy()")  
    }  
}
```

バイナリレベルでのログ出力

- ・全ての言語でアスペクトが使えるわけではない
- ・コードをソースレベルではなくバイナリレベルで追加する

バイナリコード: コンピュータに処理を依頼する命令をCPUが理解できるように2進数で表したコード

- ・結果例

```
$ head itrace.out
```

```
0x40000c20
```

```
0x40000c22
```

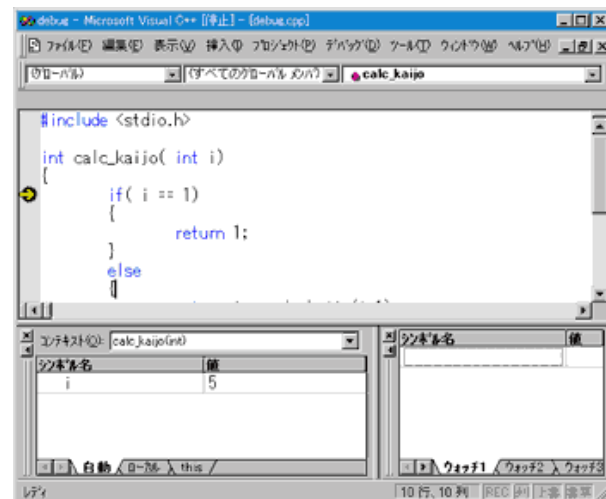
```
0x40000c70
```

デバッグの使用



デバッガとは

- ・外部の観察ツール
- ・元のプログラムコードを何も変更せず実行中のプログラムと接続する
- ・特定の瞬間の状態を観察する



デバッガの利点

- ・すぐに開始できる

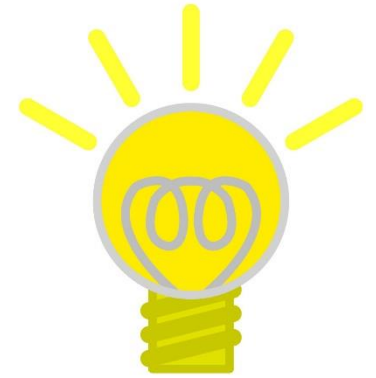
ソースコードの変更やリコンパイルが必要ない

- ・柔軟な観察ができる

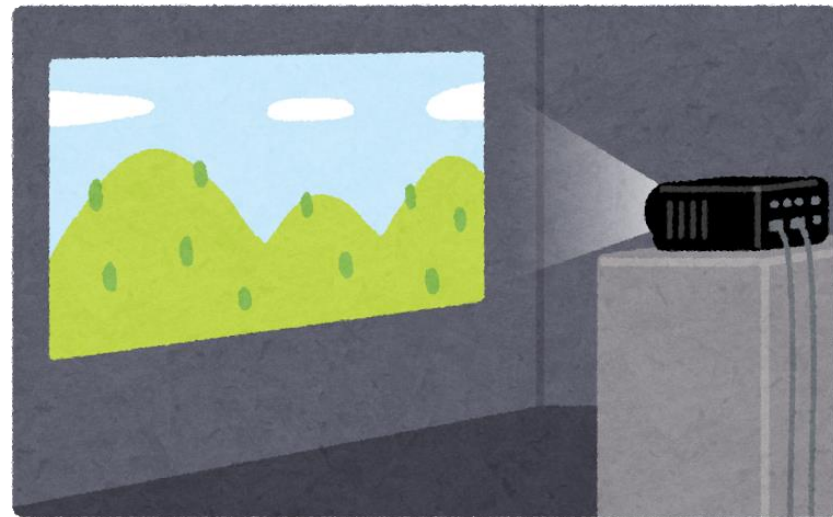
任意の観点からプログラム実行の観察が可能

- ・一時的なセッション

外部観察ツールは素早く結果が出せる対話型で1回限りのデバッグに向いている



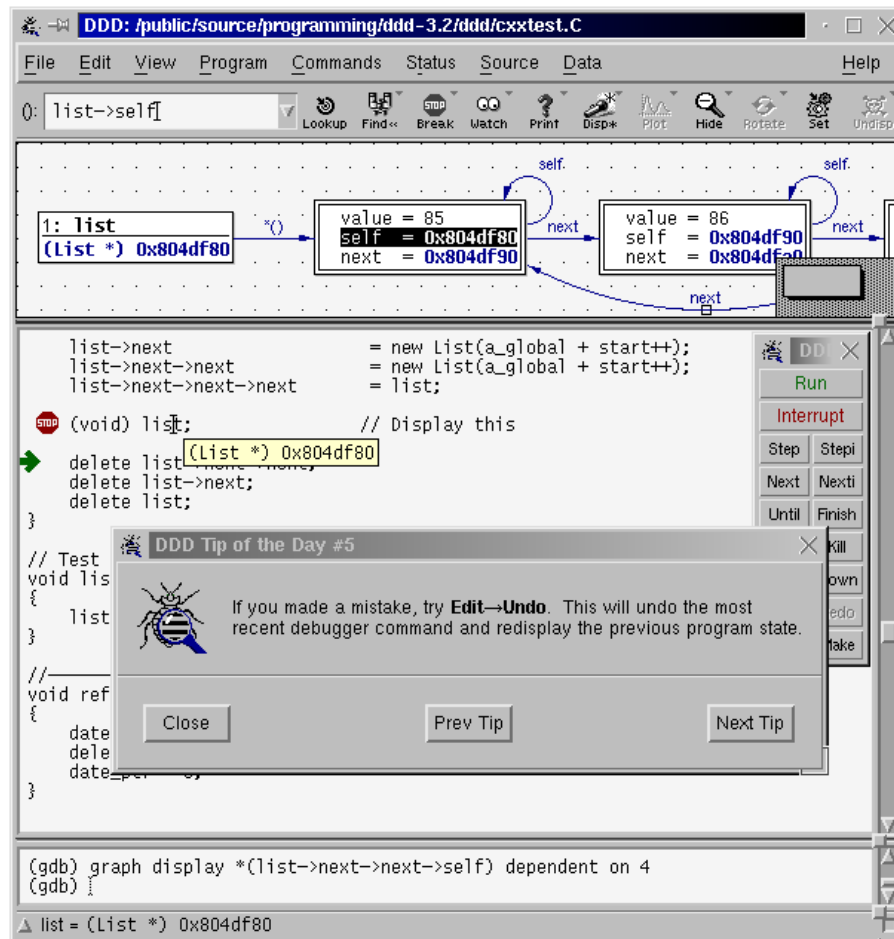
状態の可視化



状態を可視化するデバッガ

- ・データ構造を深く理解できる
- ・関係ないデータを見ることなく特定の箇所に注目できる
- ・DDD (GNU Data Display Debugger):
グラフィカルに可視化を提供するツール
数値をグラフとして描画することも可能

DDDデバッガの画面



まとめ



まとめ

- プログラムを観察する際の原則を守る必要がある
- printfデバッグ法とそれを克服した観察法
- さまざまなデバッガも存在する