

Soft Loop

タイムラインを活用した、プログラムの動作を可視化する学習システム

プログラムの動作・ロジック・アルゴリズムの実行を可視化

技育博202405 資料

丸山拓真 / しおのあそびば X:@ShioPy0101



Sechack365 ポスター

1

プロジェクト概要

こんなことありませんか . . . ?

何が原因かよくわからない論理エラーに苦しむ **気合**でprintデバッグ

突然出てくる**Segmentation fault** の特定に苦慮

過去の自分が書いたソースコードと格闘する **昨日の自分は赤の他人**

教科書・技術本に乗っている不思議なアルゴリズム **まるで魔法で動いていそう**

眠気と格闘しながら頑張って書いたプログラム
全然動かない (そんな時に限って締め切りが近い)

数時間かけて原因調査
結局原因は添え字の **i** と **j** の**書き間違い**

1

プロジェクト概要

プログラム可視化システム
解析対象ソースコード: test/bubble_sort.txt

```
int tmp = 0 ;
int i = 0 ;
int j = 0 ;
while ( i < size ) {
    j = i + 1 ;
    while ( j < size ) {
        if ( num [ i ] > num [ j ] )
        {
            tmp = num [ i ] ;
            num [ i ] = num [ j ] ;
            num [ j ] = tmp ;
        }
        j = j + 1 ;
    }
    i = i + 1 ;
}
```

タイムライン進行操作
> < < > < > < >
さいせい もどる すすむ

Line	Column	Operation	Visual
99	15	5 比較	○
100	15	2 while 条件分岐	○
101	16	8 比較	○
102	16	2 if条件分岐	○
103	22	5 加算	○
104	22	2 変数代入	○
105	15	5 比較	○
106	15	2 while 条件分岐	○
107	16	8 比較	○
108	16	2 if条件分岐	○
109	22	5 加算	○
110	22	2 変数代入	○
111	15	5 比較	○
112	15	2 while 条件分岐	○
113	16	8 比較	○
114	16	2 if条件分岐	○
115	22	5 加算	○
116	22	2 変数代入	○
117	15	5 比較	○
118	15	2 while 条件分岐	○
119	16	8 比較	○
120	16	2 if条件分岐	○
121	22	5 加算	○
122	22	2 変数代入	○
123	15	5 比較	○
124	15	2 while 条件分岐	○
125	16	8 比較	○
126	16	2 if条件分岐	○
127	22	5 加算	○
128	22	2 変数代入	○
129	15	5 比較	○

num: 1 2 10 4 2 6 1 2 9 3

size: 10

n: 10

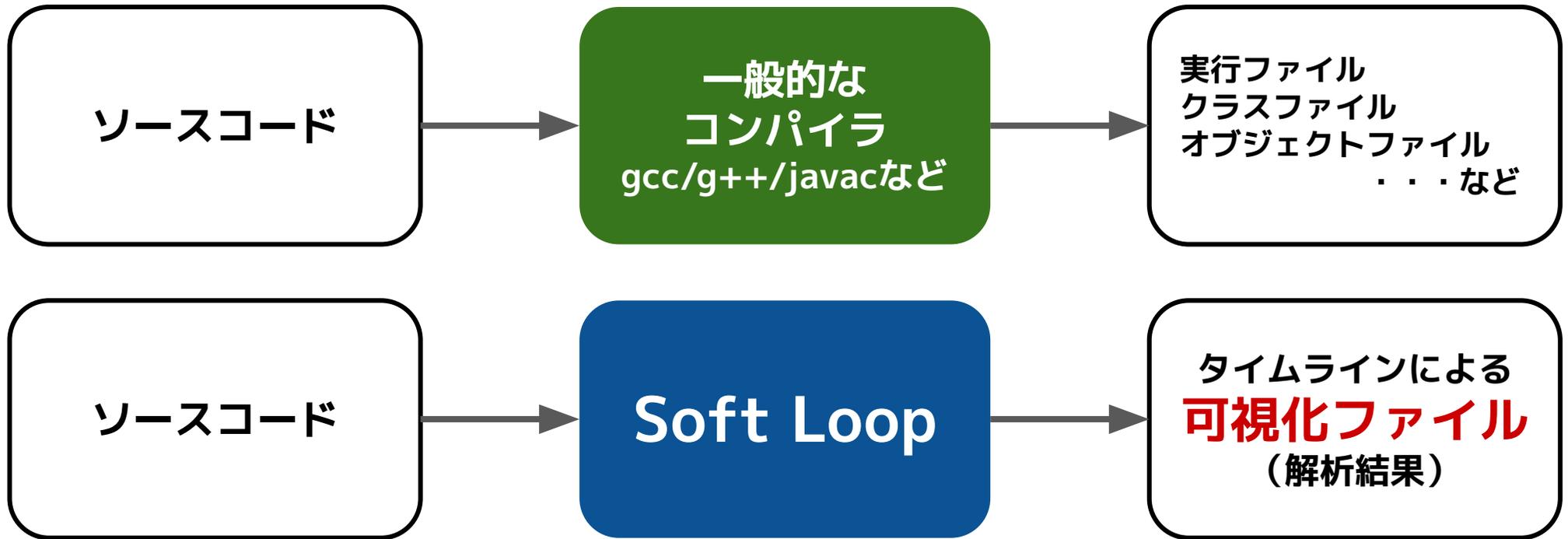
min: 1

max:

Soft Loop

プログラミング教育での活用、および、強力なデバッガとして開発

1 プロジェクト概要



対象言語は、CやC++、Javaなどの言語の
基本データ構造に関する機能を参考にした**独自言語**

現時点でも、再帰関数や多次元配列に対応しているため、
一定の高度なアルゴリズムを可視化できる体制が整っている

1

プロジェクト概要

タイムライン部

実行進行操作

さしせい もどる すすむ

130	14	2	変数代入
137	15	5	比較
138	15	2	while 条件分岐
139	16	8	比較

タイムラインを主軸にした可視化

```
int i = 0 ;
int j = 0 ;
while ( i < size ) {
    j = i + 1 ;
    while ( j < size ) {
        if ( num [ i ] > num [ j ] )
        {
            tmp = num [ j ] ;
            num [ i ] = num [ j ] ;
            num [ j ] = tmp ;
        }
    }
}
```

ソースコード表示部

変数・配列表示部

num	1	2	10	4
size	10			
n	10			

実行箇所のハイライト表示

2

特徴 (1/7)

自由に試したいソースコードを可視化できる



自分が書いたプログラムを動かせる！
作って遊ぶことができる！楽しい！

プログラム可視化システム

解析対象ソースコード：test/bubble_sort.txt

```
int tmp = 0 ;  
int i = 0 ;  
int j = 0 ;  
while ( i < size ) {  
    j = i + 1 ;  
    while ( i < size ) {
```



2

特徴 (2/7)

セキュリティ教育をアルゴリズム面からサポート

セキュリティで重要な暗号理論・符号理論には様々なアルゴリズムが使用されている！

本システムでは、解析器、解析木走査、文法定義が許す限り**符号理論を含めたさまざまなアルゴリズムの可視化が可能**

すでにランレングス圧縮(連長圧縮)の可視化に成功！

ランレングス圧縮可視化ファイルを、バブルソート可視化ファイルと共にスライド掲載場所に掲示しておきますので、ぜひ遊んでみてください

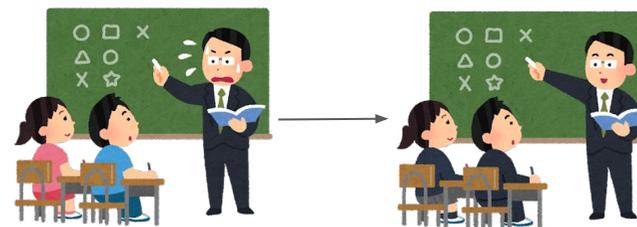
2 特徴 (3/7)

小回りが利く可視化ファイル

ソースコード

Soft Loop

可視化
ファイル



がんばってパワーポイントを使って
動作の説明をする資料を作らなくてよい
このシステムで一発！

Google Classroomに可視化ファイルを添付して
資料として配布することもできる

2

特徴 (4/7)

実行エラーの強調表示によるエラー対処の練習

```
int [ ] array = new int [ 10 ] ;  
int a = array [ 11 ] ;
```

プログラムは思った通りには動かない！
書いた通りに動く！

タイムライン進行操作				
さいせい				
もどる				
すすむ				
進行	行	列	実行	スコープ状況
0	1	5	配列配置	○
1	2	8	配列範囲外アクセス(要素)	○
2	2	9	言語処理系異常検出	○

実行エラーの強調表示、
このシステムを開発しているときに欲しかった（小声）

2

特徴 (5/7)

柔軟な構文解析・文法定義が行える仕組みの開発

```
<S> ::= <EL>
<EL> ::= <E> *
<E> ::= <left> ";" <right> ";" | <left> "=" <right> ";" | <ifexpr> | <while> | <function> | <return> ";"
<left> ::= <value_definition> | <array_definition> | <value_name> | <array_name>
<right> ::= <comparison_equal> | <function_message_passing> | <new_array>
<comparison_equal> ::= <comparison_equal> "=" <comparison> | <comparison>
<comparison> ::= <comparison> "<" <expr> | <comparison> "<=" <expr> | <comparison> ">" <expr> | <comparison> ">=" <expr> | <expr>
<expr> ::= <expr> "+" <term> | <expr> "-" <term> | <term>
<term> ::= <term> "*" <factor> | <term> "/" <factor> | <term> "<" <factor> | <factor>
<factor> ::= <number> | "(" <right> ")" | <value_name> | <array_name> | <text>
<number> ::= NUM
<value_definition> ::= <type_name> <value_name>
<array_definition> ::= <type_name> <array_length> <value_name>
<type_name> ::= "int" | "string" | "void"
<value_name> ::= TEXT
<text> ::= "" TEXT "" | "" TEXT "" | "" NUM "" | "" NUM "" | "" "" "" | "" "" ""
<array_name> ::= <value_name> <array_length>
<array_length> ::= "[" <right> "]" <array_length> | "[" "]" <array_length> |
<new_array> ::= new <type_name> <array_length>
<ifexpr> ::= <if>
<if> ::= "if" <right> <block>
<while> ::= "while" <right> <block>
<function> ::= <value_definition> "(" <argument> ")" <block>
<function_message_passing> ::= <value_name> "(" <value_enumeration> ")"
<argument> ::= <value_definition> "," <argument> | <value_definition> |
<value_enumeration> ::= <right> ";" <value_enumeration> | <right> |
<return> ::= "return" | "return" <right>
<block> ::= "(" <EL> ")"
```

LR(1)構文解析法に基づいた解析器の実装

BNF準拠の構文定義ファイル、
および構文木走査モジュールを組み変えるだけで、
可視化の根拠となる言語仕様の変更が可能

可視化
システム

構文木走査
モジュール

構文定義
ファイル
(BNF準拠)

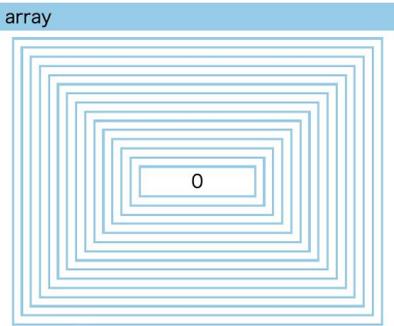
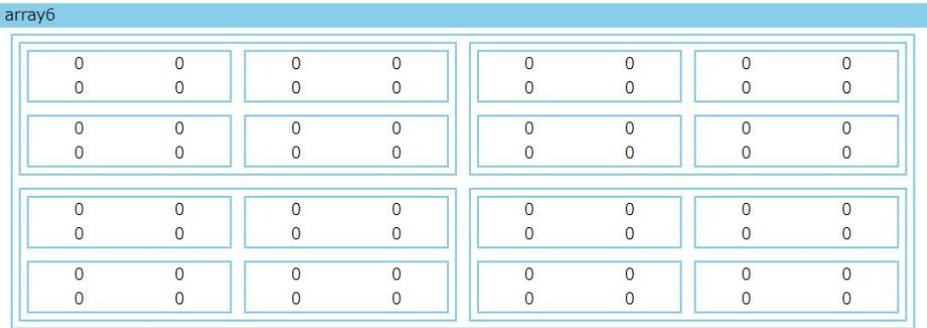
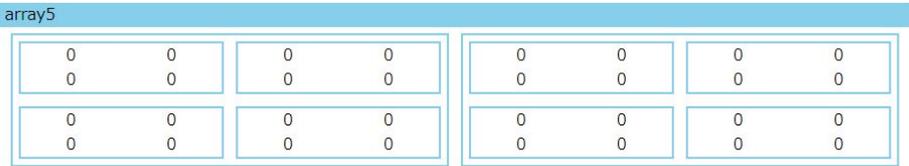
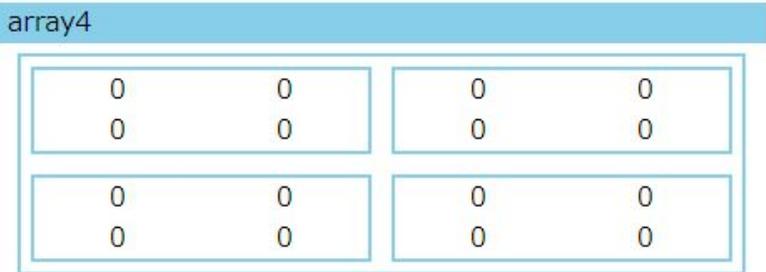
柔軟な変更・差し替え

2

特徴 (6/7)

多次元配列の可視化もばっちり

```
int [] array2 = new int[2][2];  
int [] array3 = new int[2][2][2];  
int [] array4 = new int[2][2][2][2];  
int [] array5 = new int[2][2][2][2][2];  
int [] array6 = new int[2][2][2][2][2][2];
```



何次元でも可視化が可能な表現機能

30次元配列の可視化 (一発芸)

2

特徴 (7/7)

再帰関数が理解しやすい可視化

```

int i = 0 ;
int function ( ) {
    if ( i < 3 ) {
        i = i + 1 ;
        function ( ) ;
    }
}
function ( ) ;

```

進行	行	列	実行	スコープ状況
0	2	3	変数定義	○
1	11	2	関数呼び出し	○
2	5	2	関数実行	○
3	5	5	比較	○
4	5	2	if条件分岐	○
5	6	5	加算	○
6	6	2	変数代入	○
7	7	2	関数呼び出し	○
8	5	2	関数実行	○
9	5	5	比較	○
10	5	2	if条件分岐	○
11	6	5	加算	○
12	6	2	変数代入	○
13	7	2	関数呼び出し	○
14	5	2	関数実行	○
15	5	5	比較	○
16	5	2	if条件分岐	○
17	6	5	加算	○
18	6	2	変数代入	○
19	7	2	関数呼び出し	○
20	5	2	関数実行	○
21	5	5	比較	○
22	5	2	if条件分岐	○
23	5	2	関数終了	○
24	5	2	関数終了	○
25	5	2	関数終了	○
26	5	2	関数終了	○

34	22	2	while条件分岐	○
35	24	5	比較	○
36	24	2	if条件分岐	○
37	29	8	比較	○
38	29	2	if条件分岐	○
39	31	5	加算	○
40	31	2	変数代入	○
41	33	8	比較	○

whileとifの組み合わせによる
ネスト

再帰関数の仕様を直感的に学ぶことができる 再帰関数に親しくなれる

**再帰的アルゴリズムが怖いとはもう言わせない！
再帰的アルゴリズムはともだち！！！！**

3 フィードバックの実施・評価（抜粋して紹介）

コンセプトについて

グラフィカルに表現され、視覚的に非常に面白く、
抽象的な概念に対する親しさを覚える

初心者から中級者以上まで幅広く役立てられそうなので、
このプロダクトの秘めるポテンシャルはかなり大きそう



3 フィードバックの実施・評価（抜粋して紹介）

可視化について

評価点

ifやwhile文の処理が色で分かる点も分かりやすい

コードの中で今どこが実行されていて、ループで戻る場所がわかる

タイムライン上の処理の説明と
ソースコード上のハイライトが同期しており、対応関係がわかりやすい

配列が実際にソートされている様子が反映されていたため、
どの数字がどこに移動するのか把握しやすい

3 フィードバックの実施・評価（抜粋して紹介）

可視化について

改善点

タイムラインやプログラム同様、変数や配列にも色が反映されると、どの部分が変更されたかもっと分かりやすくなりそう

配列のswapなど重要な処理に関して、アニメーションでその様子が強調されたら教材レベルになりそう

コードの中の変数にカーソルを合わせた時、変数の中身を見たい