gomicollector
ガベコレ自作で辿る自作の森

セキュリティ・キャンプフォーラム2023 (2023/3/11)

speed(@strayer_13)

自己紹介

speed

Twitter: @strayer_13

- 大阪大学B3
- セキュキャン2022全国大会修了生
- 研究室選びに迷い中

目次

input part

- ガベージコレクションについて
- マーク･アンド･スイープ法

output part

- Rustによるガベコレ実装

- なぜ「自作」するのか

ガベージコレクションとは?

ガベージコレクション(英: garbage collection、GC)とは、

コンピュータプログラムが動的に確保したメモリ領域のうち、

不要になった領域を自動的に解放する機能である。

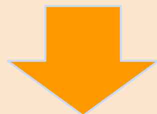
(Wikipedia ガベージコレクション より引用)

…….?

以下詳しく解説します!

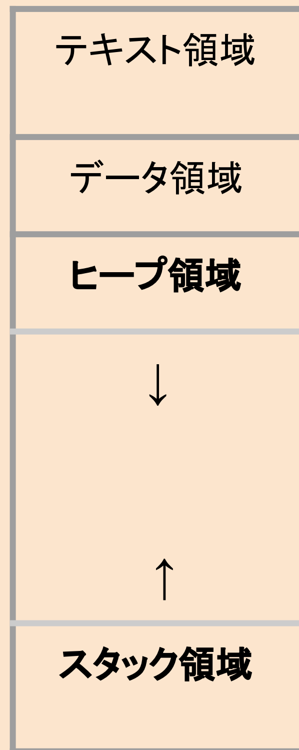
プロセスのメモリ構成―スタックとヒープ

プロセスが利用できるメモリは有限



伸縮する二つの領域を活用

- ヒープ領域
- スタック領域



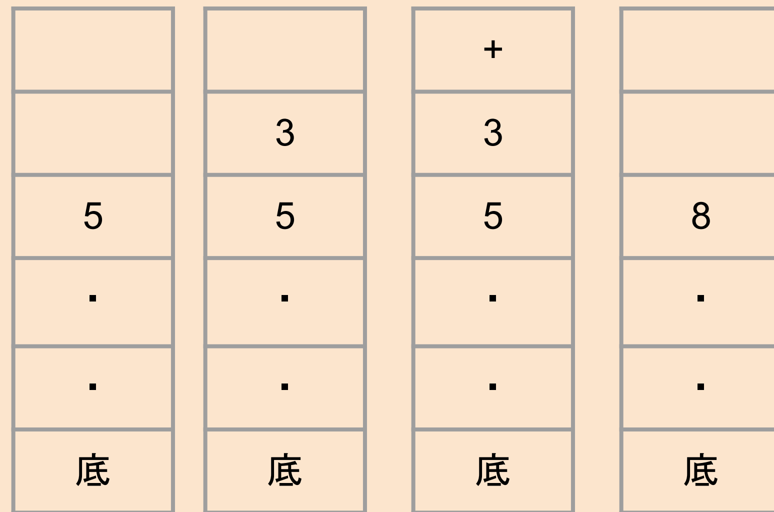
スタック領域

コンパイル時にサイズが決まるデータ

を格納

例

- 式の計算
  - 5+3
  - 関数呼び出しに必要な情報(駆動レコード)



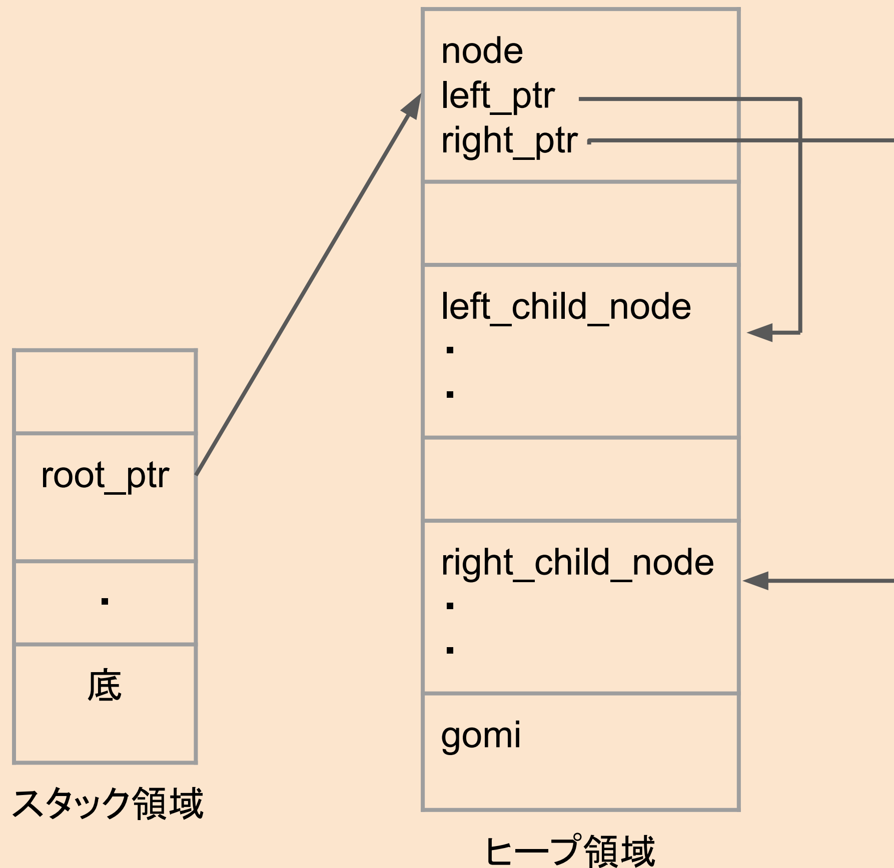
ヒープ領域

実行時に動的にサイズが変わるデータ

を格納

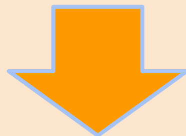
例

- 木(Tree)
- 連結リスト(LinkedList)



ヒープ領域のメモリ管理の難しさ

- 不要メモリの解放忘れ(メモリリーク)
- ぶら下がりポインタ(ダングリングポインタ)
- メモリの二重解放
- etc

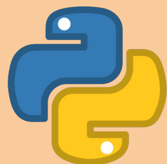


ガベージコレクションがまるっと解決!

Garbage Collection is Everywhere

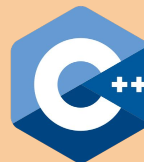
多くの言語がガベージコレクションを採用

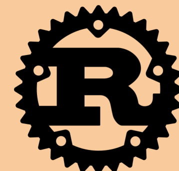
Python, Golang, Java, JavaScript, etc



ガベージコレクションを採用していない言語

C, C++, Rust, etc



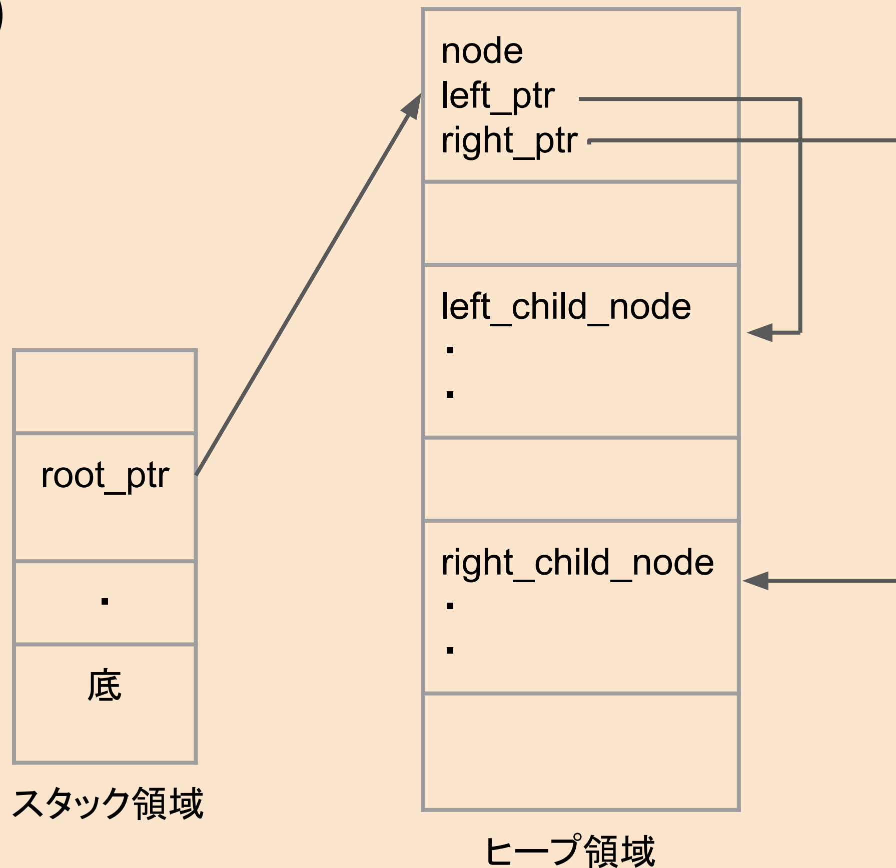


ガベージコレクションのアルゴリズム各種

- **マーク･アンド･スイープ法**(←gomicollectorで採用)
- 参照カウント法
- コピーGC
- 世代別GC
- etc

マーク・アンド・スイープ法(1/4)

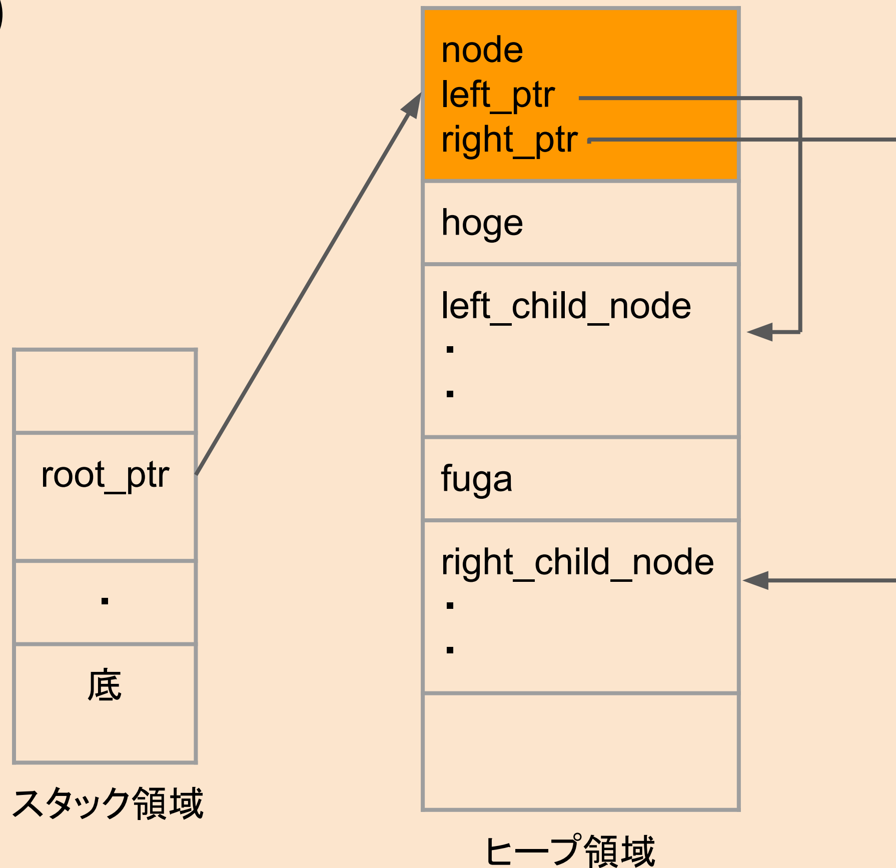
1. mark: ヒープ領域のうち到達可能な領域を全てマーク
2. sweep: マークされていない領域はゴミとみなして解放



マーク・アンド・スイープ法(2/4)

1. mark: ヒープ領域のうち到達可能な領域を全てマーク
2. sweep: マークされていない領域はゴミとみなして解放

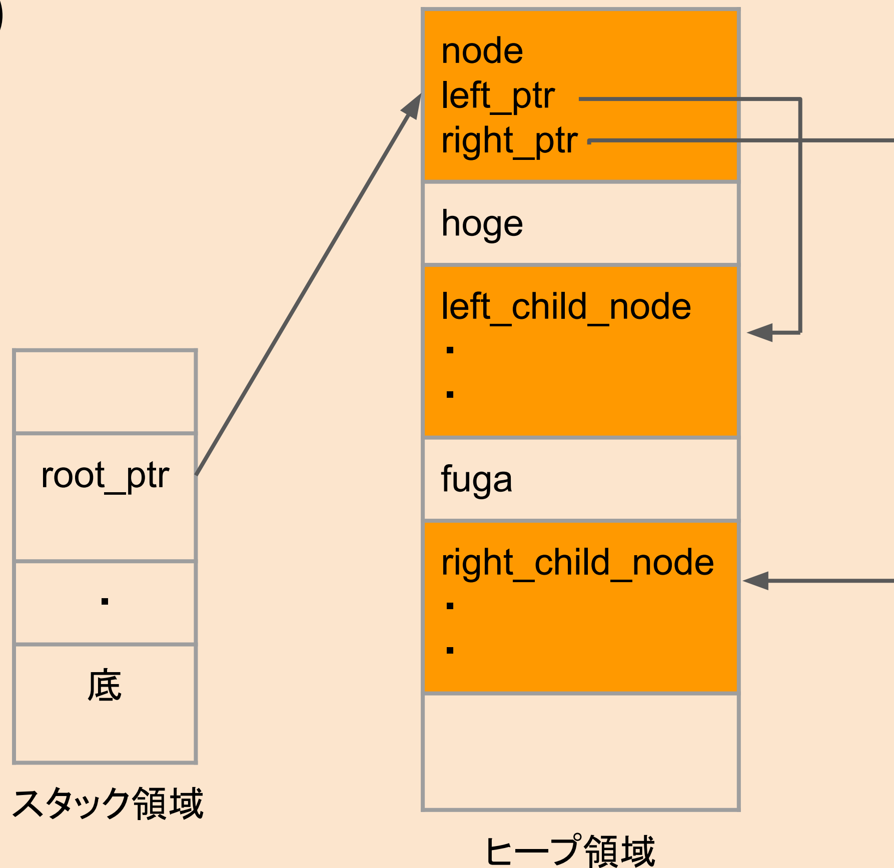
スタック領域から辿れる領域をマーク



マーク・アンド・スイープ法(3/4)

1. mark: ヒープ領域のうち到達可能な領域を全てマーク
2. sweep: マークされていない領域はゴミとみなして解放

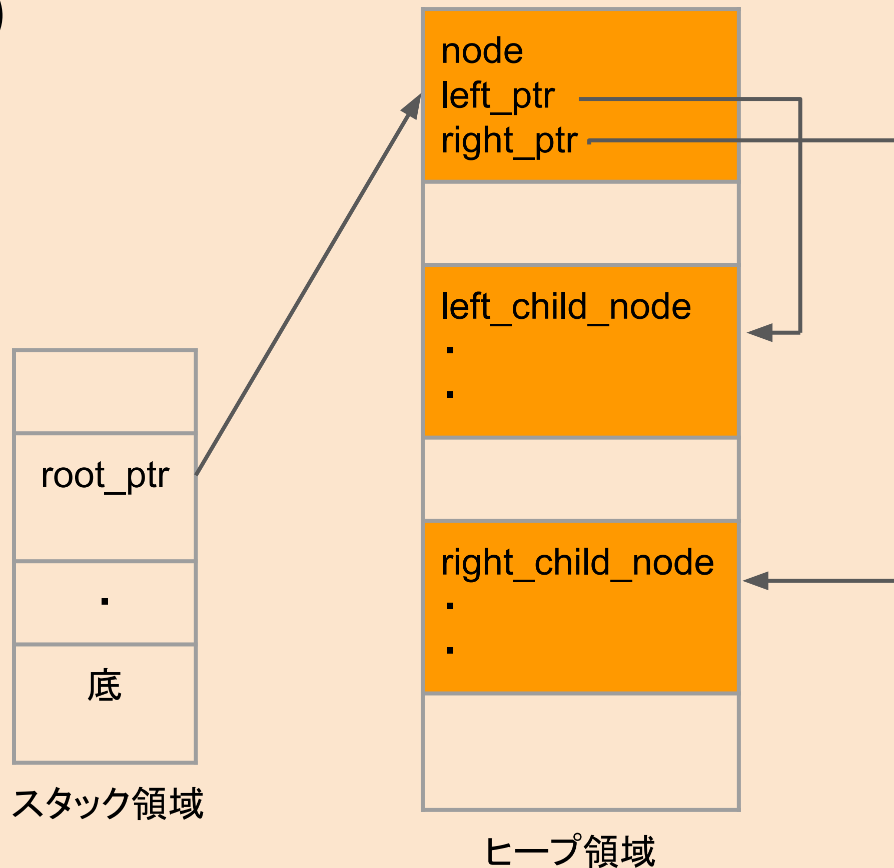
マークした領域から再帰的に辿れる領域をマーク



マーク･アンド･スイープ法(4/4)

1. mark: ヒープ領域のうち到達可能な領域を全てマーク
2. sweep: マークされていない領域はゴミとみなして解放

マークされていない
領域を解放



Rustによるガベコレ実装方法

root_set: set	{obj1}

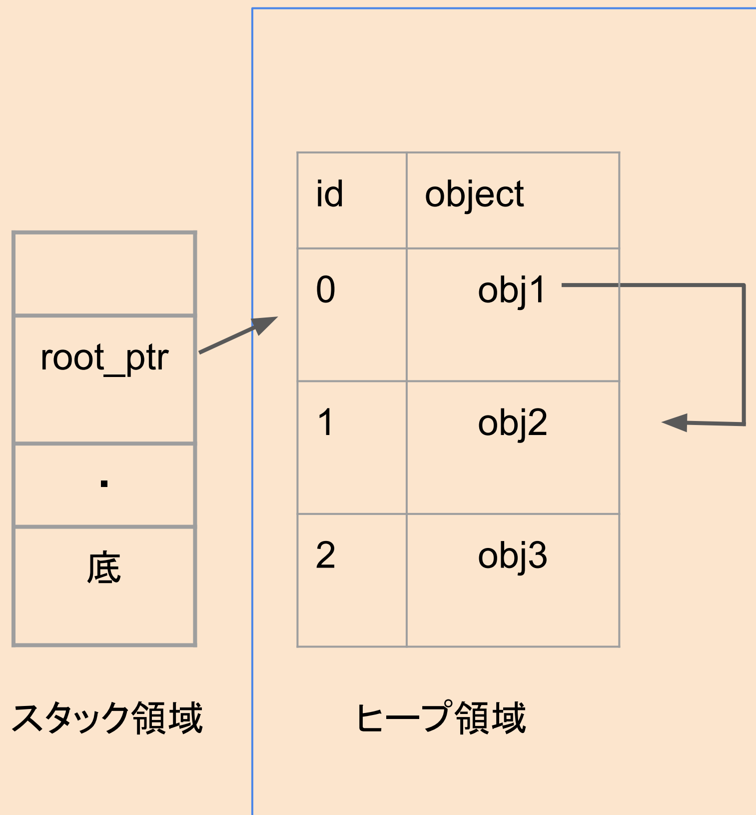
markの開始地点

free_list: Vec<int>	[]

ヒープの利用できる領域

id: int	marked: bool	object名: str	points to: int	data: T
0	false	obj1	obj2	“hello”
1	false	obj2		“こんにちは”
2	false	obj3		“你好”

Rustでの表現



実体

gomicollectorでのObject, Heapのデータ構造

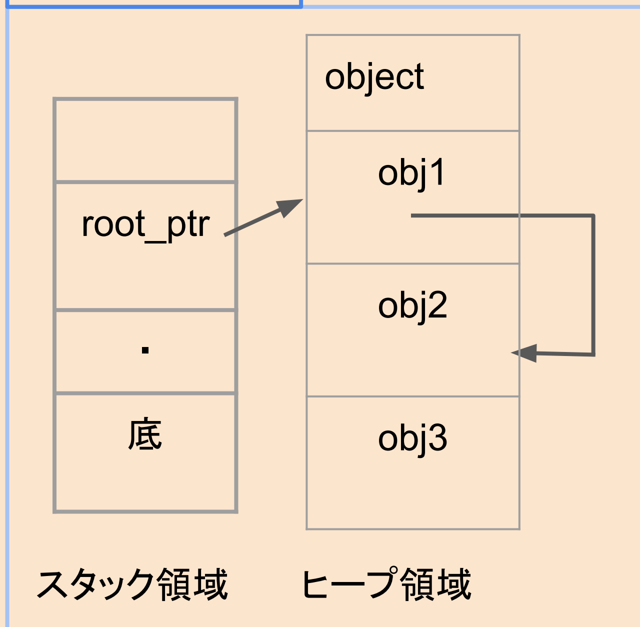
```rust
pub struct Object<T: Debug + Clone> {
    head: Option<usize>,
    tail: Option<usize>,
    marked: bool,
    id: usize,
    data: Option<T>,
}
```

```rust
pub struct Heap<T: Debug + Clone> {
    pub heap: Vec<Object<T>>,
    pub root_set: HashSet<usize>,
    size: usize,
    pub free_list: Vec<usize>,
}
```

https://github.com/speed1313/gomicollector より抜粋

実行結果

demo code

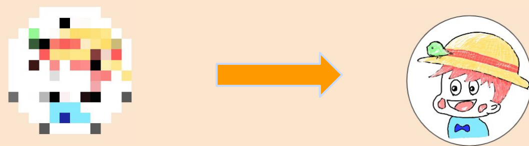


結果

```
$ cargo run main.rs
mark and sweep
obj1 allocated Object { head: None, tail: None, marked: false, id: 2, data: Some("obj1") }
obj2 allocated: Object { head: None, tail: None, marked: false, id: 1, data: Some("obj2") }
obj3 allocated: Object { head: None, tail: None, marked: false, id: 0, data: Some("Obj3") }
obj4 will be allocated
mark and sweep
droped "Obj3"
heap: Heap {
    heap: [
        Object {
            head: None,
            tail: None,
            marked: false,
            id: 0,
            data: Some(
                "obj4",
            ),
        },
        Object {
            head: None,
            tail: None,
            marked: true,
            id: 1,
            data: Some(
                "obj2",
            ),
        },
        Object {
            head: Some(
                1,
            ),
            tail: None,
            marked: true,
            id: 2,
            data: Some(
                "obj1",
            ),
        },
    ],
    root_set: {
        2,
    },
    size: 3,
    free_list: [],
}
reachable set: ["obj1", "obj2"]
```

なぜ「自作」するのか

- アルゴリズムの解像度が上がる!
  - なぜ木を辿るときにマークするの？→ 再訪問を防いで無限ループ回避！
  - ヒープ領域のレイアウトはどうするの？→ 固定長にしてみる！



- 自作の広がり
  - プロセスの動作→OS自作
  - スタックの使われ方→コンパイラ自作
- 楽しい!
- 作れたら怖いもの無し!

まとめ

- ヒープ領域の管理をガベコレが解決
- マーク・アンド・スイープ法のGCをRustで実装

Further Reading

- gomicollector コード: https://github.com/speed1313/gomicollector
- gomicollector 解説記事: https://speed1313.notion.site/Garbage-Collection-mark-sweep-b04f5cb763824b8b9cc3735c29fde545