#eureka

アルゴリズム #02

Shintaro Kaneko
iOS/Android/Web Developer

© 2014 Shintaro Kaneko

今後の流れ

データ構造はプログラミングで重要
その構造をイメージできるようにまずは訓練する
アルゴリズムの本題はまだまだかも知れないし、次かもしれない

アルゴリズムやりたいこと
- 文字列検索
- インデクシングの理論
- 暗号理論、署名
- 誤り訂正符号
- パターン認識

Agenda

C言語

小手調べ

連結リスト（単方向）

C言語

C言語

ポインタ

typedef

構造体

構造体 with ポインタ

メモリアサイン

※型、制御構文などの超基礎はやりません

ポインタ

```
void swap(int *a, int *b) {
    int c = *a;
    *a = *b;
    *b = c;
}
```

```
int main(int argc, const char * argv[]) {
    int a = 10, b = 20;
    int *x = &a;
    swap(&a, &b); printf("a = %d, b = %d\n", a, b);
    *x = 30;      printf("a = %d\n", a);
    return 0;
}
```

typedef

```c
typedef int integer;
typedef char *string;

int main(int argc, const char * argv[]) {
    integer i = 0; // int i = 0;
    string str = "foo"; // char *str = "foo";
    return 0;
}
```

構造体

```
struct person_tag {
    char *nickname;
    int age;
};
typedef struct person_tag person_t;
```

```
int main(int argc, const char * argv[]) {
    struct person_tag person = {"kaneshin", 26};
    // person_t person = {"kaneshin", 26};
    printf("%s is %d years old.\n",
           person.nickname, person.age);
    return 0;
}
```

構造体 with ポインタ

```c
struct person_tag {
    char *nickname;
    int age;
};
typedef struct person_tag person_t;
```

```c
int main(int argc, const char * argv[]) {
    person_t person = {"kaneshin", 26};
    person_t *person_ptr = &person;
    printf("%s is %d years old.\n",
           person_ptr->nickname, person_ptr->age);
    return 0;
}
```

メモリアサイン

```c
#include <stdlib.h>
struct person_tag {
    char *nickname;
    int age;
};
typedef struct person_tag person_t;
int main(int argc, const char * argv[]) {
    person_t *person_ptr = malloc(sizeof(person_t));
    person_ptr->nickname = "kaneshin";
    person_ptr->age = 26;
    printf("%s is %d years old.\n",
           person_ptr->nickname, person_ptr->age);
    free(person_ptr);
    return 0;
}
```

メモリアサイン

ポインタ・メモリ管理を理解しているなら飛ばします

http://www.chokkan.org/lectures/2012c/p2-3_pointer.pdf

```
struct foo_tag *foo = malloc(sizeof(struct foo_tag));
struct foo_tag *bar;
bar = /* fooをコピーしたい */
```

小手調べ

スライドパズル

http://www.afsgames.com/15puzzle.htm

フィボナッチ数列

$F_{n+2} = F_{n+1} + F_n \ (\forall n \geqq 0, n \in N \cup \{0\})$
$F_1 = 1, F_0 = 0$

Question:
Is the value of the Fibonacci sequence where n is 15.

10進数 → 2進数

10進数の「109」を2進数にしてください
※再帰関数を定義すること

連結リスト（単方向）

連結リスト（単方向）

単方向リスト

実践

Gitリスト

単方向リスト

単方向リスト

データとポインタの構造体（ノード）を単一方向に接続したリスト

ノード 

最初のノードを参照するノード

単方向リスト

メリット

ノードの追加・挿入・削除が容易

必要なときにメモリ領域を確保

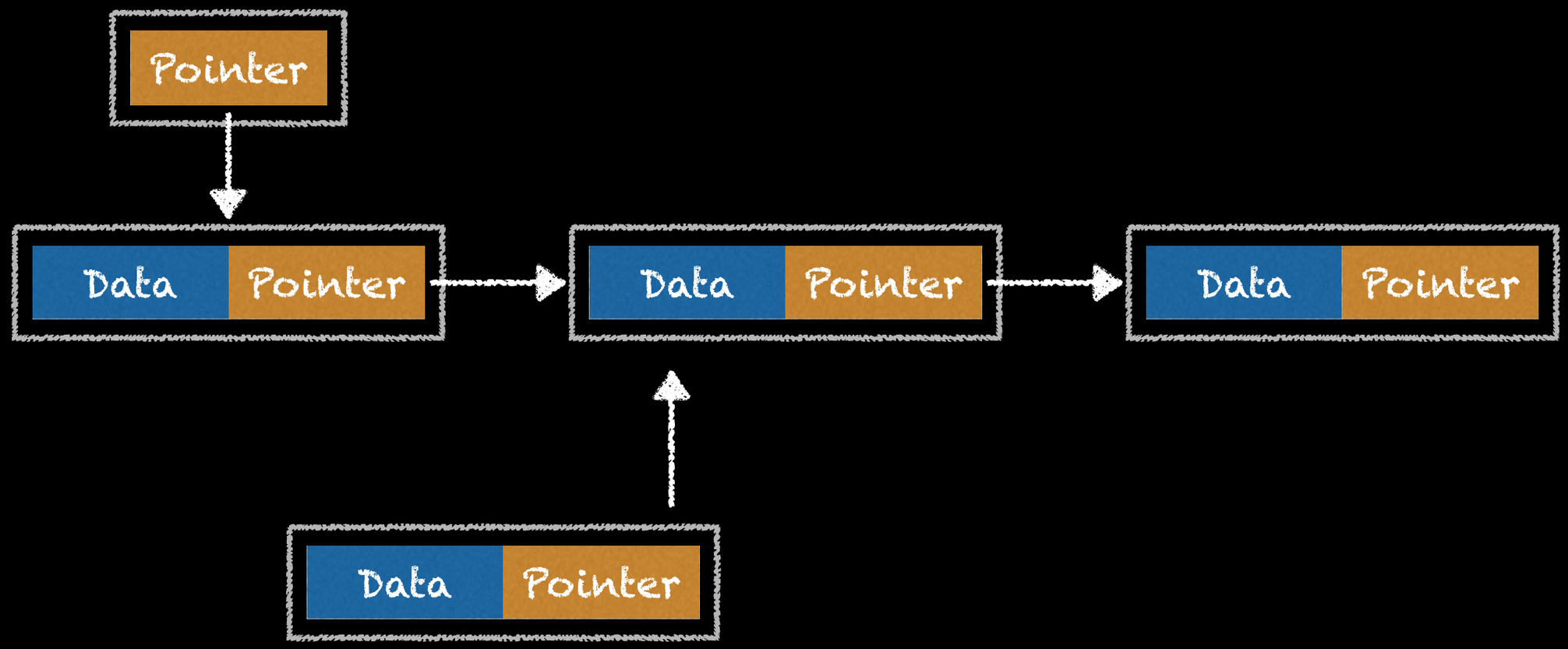
配列のようにメモリ上に連続していないため

デメリット

各ノードへのアクセスは先頭から順に辿る必要がある

単方向リスト

単方向リスト


Pointer
Data
Pointer
Data
Pointer
Data
Pointer
Data
Pointer

実践

ノード構造体

データとポインタの構造体（ノード）を単一方向に接続したリスト

ノード 

このノードを構造体（Struct）で表してください。

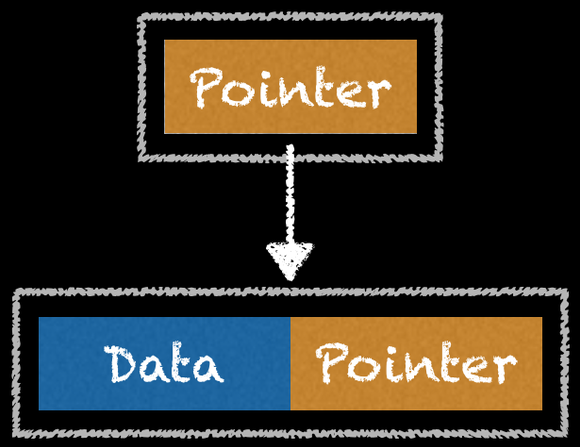
※Dataは整数型

ノード構造体

```
typedef struct node_tag {
    int data;
    struct node_tag *next;
} node_t;
```

ノードの追加

ポインタがノードを指す



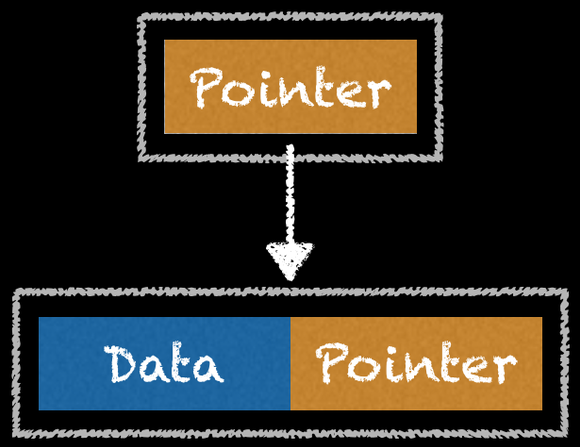
さらに別のノードを指す

ノードの追加

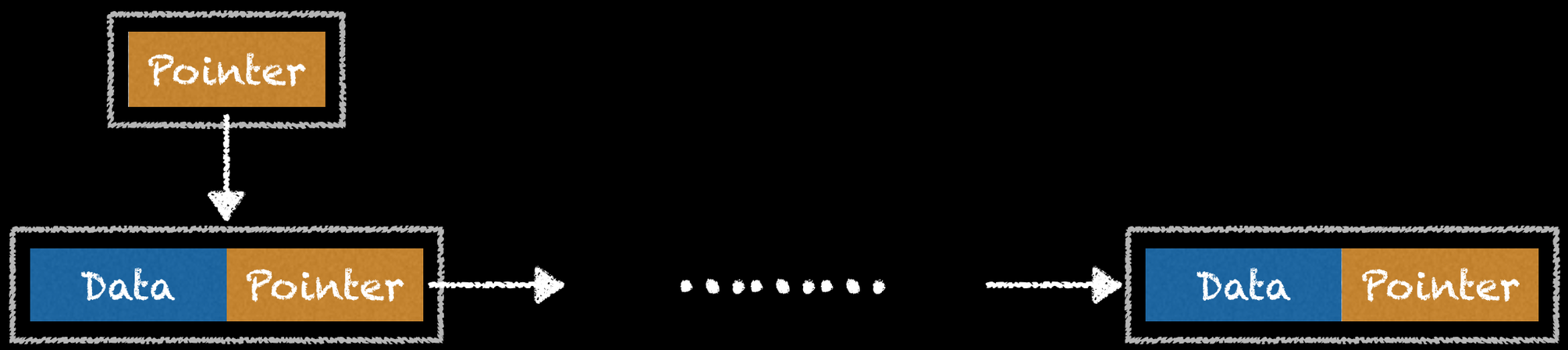
```c
int main(void) {
    node_t *head = malloc(sizeof(node_t));
    node_t *n1 = malloc(sizeof(node_t));
    n1->data = 1;
    head->next = n1;
    node_t *n2 = malloc(sizeof(node_t));
    n2->data = 2;
    n1->next = n2;
    node_t *n3 = malloc(sizeof(node_t));
    n3->data = 3;
    n2->next = n3;
}
```

ノードの追加

ポインタがノードを指す



さらに別のノードを指す



ノードの追加（末尾）

最後のノードまで辿り、参照するポインタを作成したものを指す

ノードの追加（末尾）

```c
typedef struct node_tag {
    int data;
    struct node_tag *next;
} node_t;
```

```c
void node_append_to_tail(node_t *head, int data);
```

ノードの追加（末尾）

```
void node_append_to_tail(node_t *head, int data) {
    node_t *node = malloc(sizeof(node_t));
    node->data = data;
    while (head->next != NULL) {
        head = head->next;
    }
    head->next = node;
}
```

ノードの追加（先頭）

先頭のノードが指すポインタを変更する

ノードの追加（先頭）

```
typedef struct node_tag {
    int data;
    struct node_tag *next;
} node_t;
```

```
void node_append_to_head(node_t *head, int data);
```

ノードの追加（先頭）

```
void node_append_to_head(node_t *head, int data) {
    node_t *node = malloc(sizeof(node_t));
    node->data = data;
    node->next = head->next;
    head->next = node;
}
```

Gitリスト

Gitリスト

実践のものを応用すれば簡単にコミットを出力するものは作成可能

Gitリストの仕組み

ホワイトボード

今後の流れ

データ構造はプログラミングで重要
その構造をイメージできるようにまずは訓練する
アルゴリズムの本題はまだまだかも知れないし、次かもしれない

アルゴリズムやりたいこと
- 文字列検索
- インデクシングの理論
- 暗号理論、署名
- 誤り訂正符号
- パターン認識

終わりに

アルゴリズムやりたいこと
- 文字列検索
- インデクシングの理論
- 暗号理論、署名
- 誤り訂正符号
- パターン認識

金子の中で「これやらないとベースが不安定になる」と思ったところをやっています。
特にアルゴリズムに縛られず、やってほしいことがあれば言ってください。
Goはいつかやるので安心してください。
数学やりたい人も言ってください。