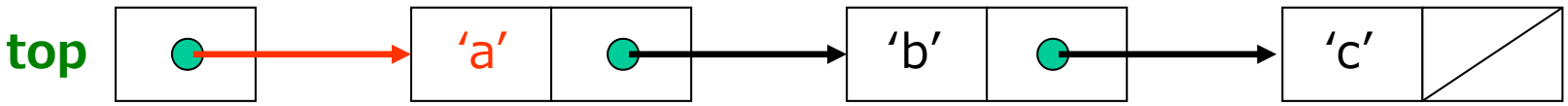
第5週
リスト

連結リスト

- 「リスト」ともいう
- ポインタでデータをつなぎあわせたもの
  - 各データは構造体で表現
  - 構造体のメンバに次のデータへのポインタを含む
- 線状リスト(線形リスト)、循環リストがある

連結リストの種類

●片方向線状リスト
- データをポインタで線状につないだもの
- 各データは、次のデータへのポインタを持つ
  (最後のデータの次は NULL)

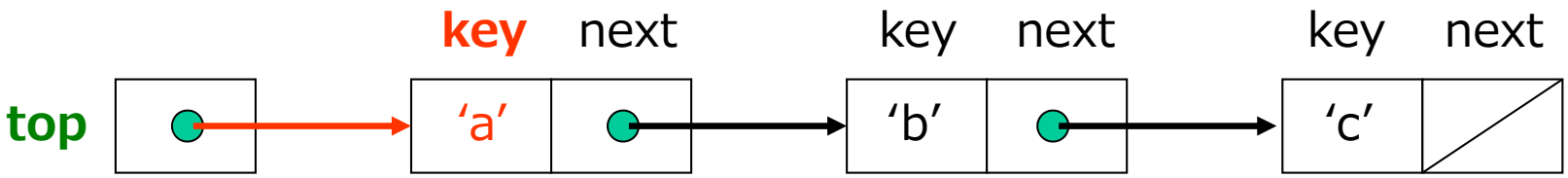


cf.「データ構造とアルゴリズム」資料

連結リストの長所・短所

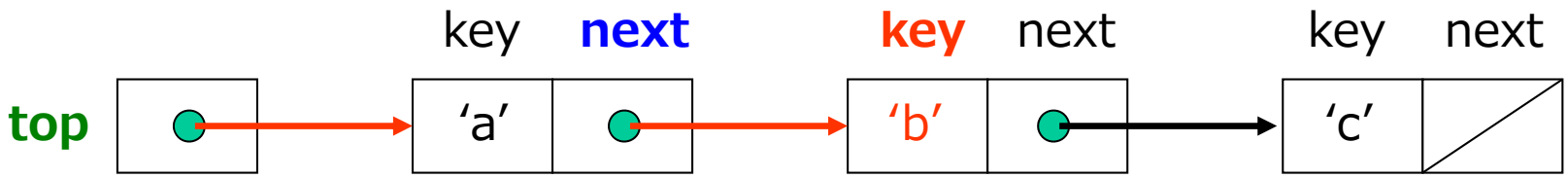
- 長所
  - データの数が可変
    - データが予想より増えても対応可能
    - あらかじめ余分にメモリ確保する必要がない
  - データの挿入や削除が高速
  - ポインタの繋ぎ方を柔軟に変更可能
- 短所
  - データへのアクセスは配列より遅い
  - ポインタの分、メモリを余分に使う

必須課題5-1 (1/3)



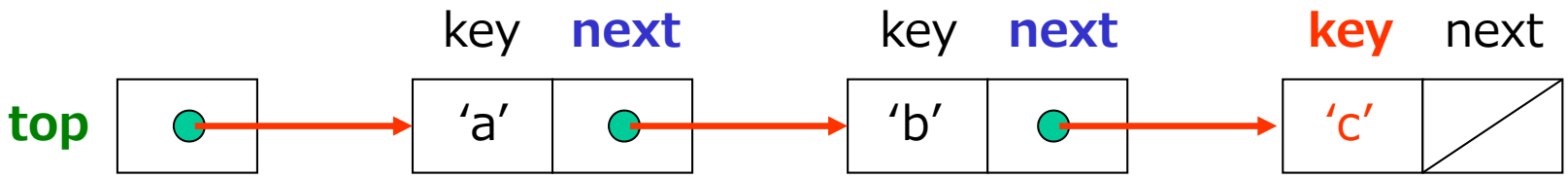

```
printf("%c¥n",top->key);
printf("%c¥n",top->next->key);
printf("%c¥n",top->next->next->key);
```

必須課題5-1 (2/3)



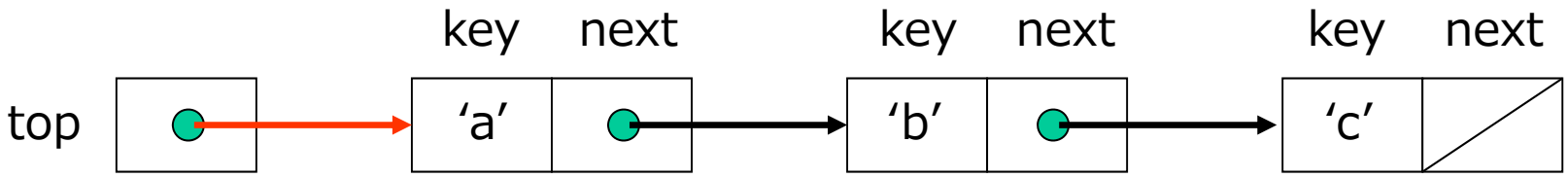

```
printf("%c¥n",top->key);
printf("%c¥n",top->next->key);
printf("%c¥n",top->next->next->key);
```

必須課題5-1 (3/3)




```
printf("%c¥n",top->key);
printf("%c¥n",top->next->key);
printf("%c¥n",top->next->next->key);
```

必須課題5-1：参考プログラム（リストの初期化部分のみ）

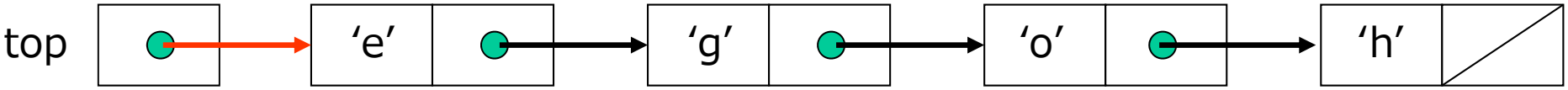


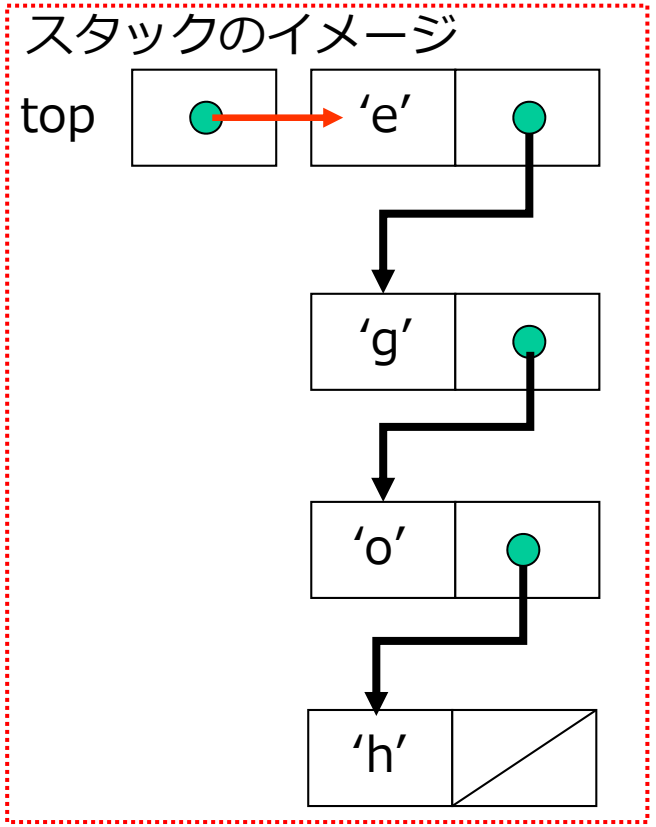

```
struct data *top=NULL;
top = 動的メモリ確保
top->key = ‘a’;
top->next = NULL;

top->next = 動的メモリ確保
top->next->key =‘b’;
top->next->next = NULL;

top->next->next = 動的メモリ確保
top->next->next->key =‘c’;
top->next->next->next=NULL;
```

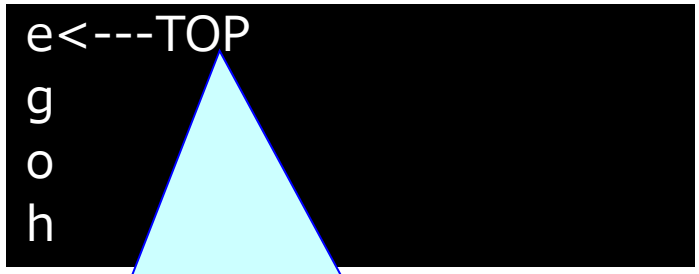
構造体struct dataの
動的メモリ確保を行う
必須課題3-3を復習しよう。
エラー処理も記述が必要

必須課題5-２：出力例

リスト版スタックの状態を出力する関数
void print_stack_list(struct data *top)

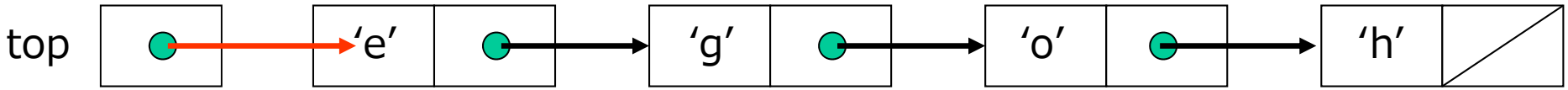






必須課題５-1とは
　TOPの表示位置が違うことに注意

必須課題5-２：print_stack_list

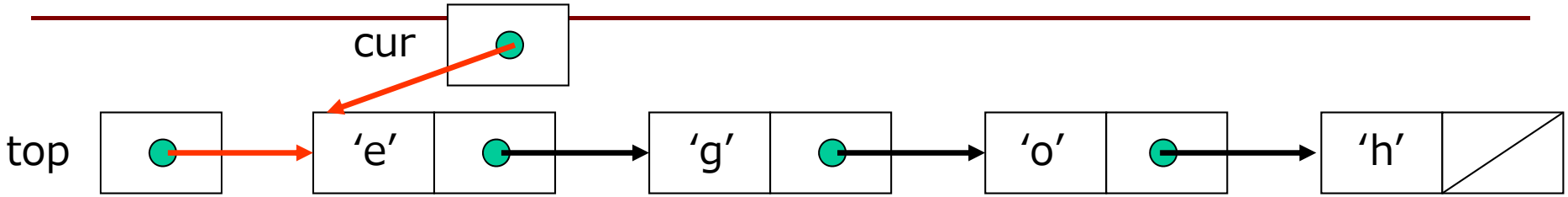


●ループを利用し、出力例以外にも対応できること

- struct data *型の一時変数（仮にcurとする）を用意する
- 一時変数に先頭（top）を代入する
- 一時変数を、一時変数自体のnextで更新しながら、リストを辿る

  例：cur = cur->next ;
- リスト辿りながら、リストの各要素（key）を出力する
- ループの終了判定はnextがNULLかどうかで判定する

必須課題5- 2 ：print_stack_list

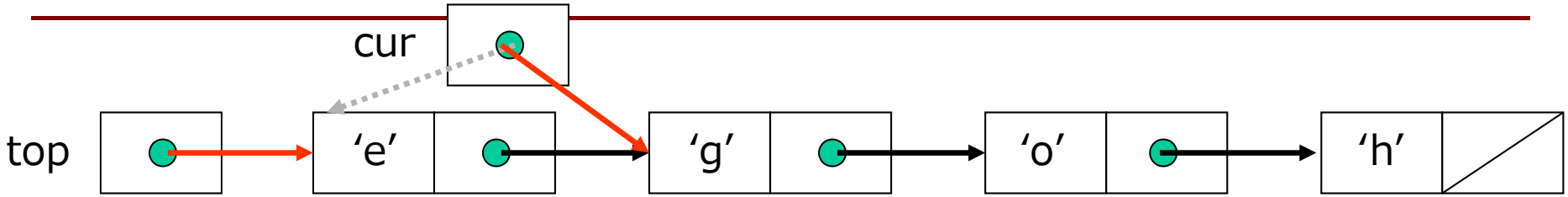


●ループを利用し、出力例以外にも対応できること

- struct data *型の一時変数（仮にcurとする）を用意する
- 一時変数に先頭（top）を代入する
- 一時変数を、一時変数自体のnextで更新しながら、リストを辿る

  例：cur = cur->next ;
- リスト辿りながら、リストの各要素（key）を出力する
- ループの終了判定はnextがNULLかどうかで判定する

必須課題5- 2 ：print_stack_list

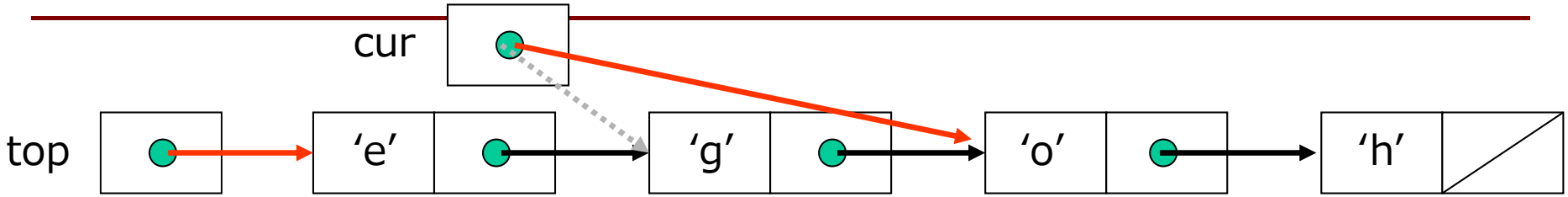


●ループを利用し、出力例以外にも対応できること

- struct data *型の一時変数（仮にcurとする）を用意する
- 一時変数に先頭（top）を代入する
- 一時変数を、一時変数自体のnextで更新しながら、リストを辿る

  例：cur = cur->next ;
- リスト辿りながら、リストの各要素（key）を出力する
- ループの終了判定はnextがNULLかどうかで判定する

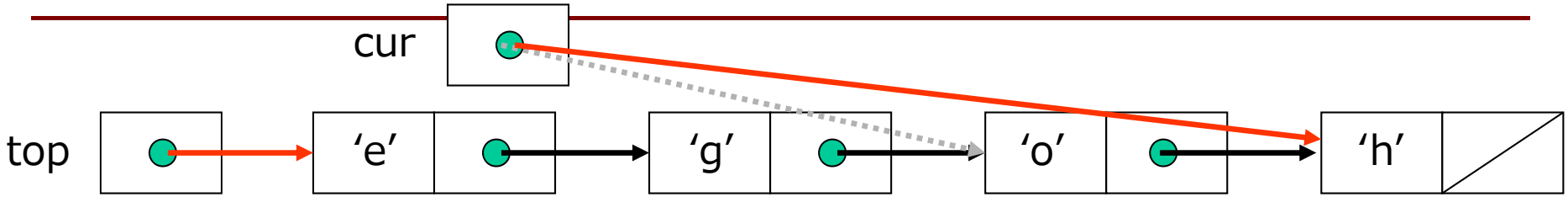
必須課題5- 2 ：print_stack_list



- ループを利用し、出力例以外にも対応できること
  - struct data *型の一時変数（仮にcurとする）を用意する
  - 一時変数に先頭（top）を代入する
  - 一時変数を、一時変数自体のnextで更新しながら、リストを辿る

    例：cur = cur->next ;
  - リスト辿りながら、リストの各要素（key）を出力する
  - ループの終了判定はnextがNULLかどうかで判定する

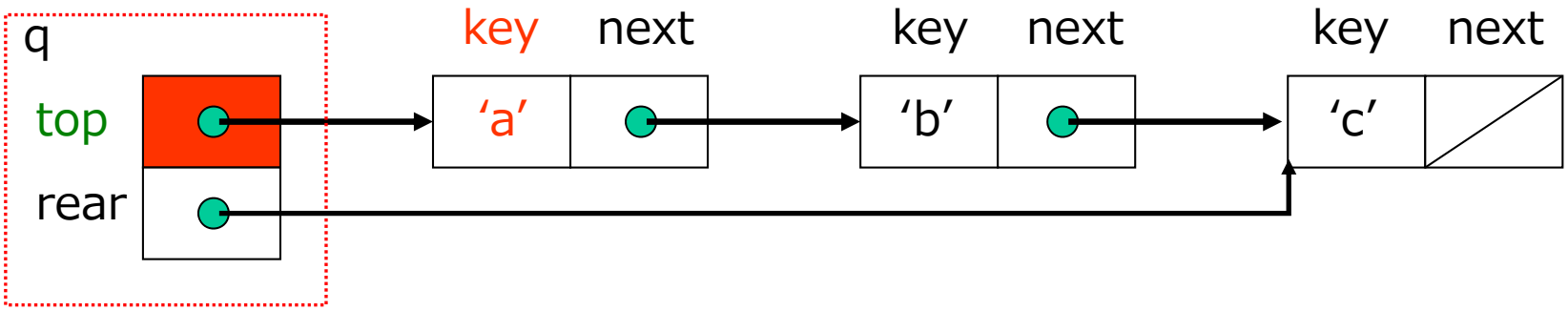
必須課題5- 2 ：print_stack_list



- ループを利用し、出力例以外にも対応できること
  - struct data *型の一時変数（仮にcurとする）を用意する
  - 一時変数に先頭（top）を代入する
  - 一時変数を、一時変数自体のnextで更新しながら、リストを辿る

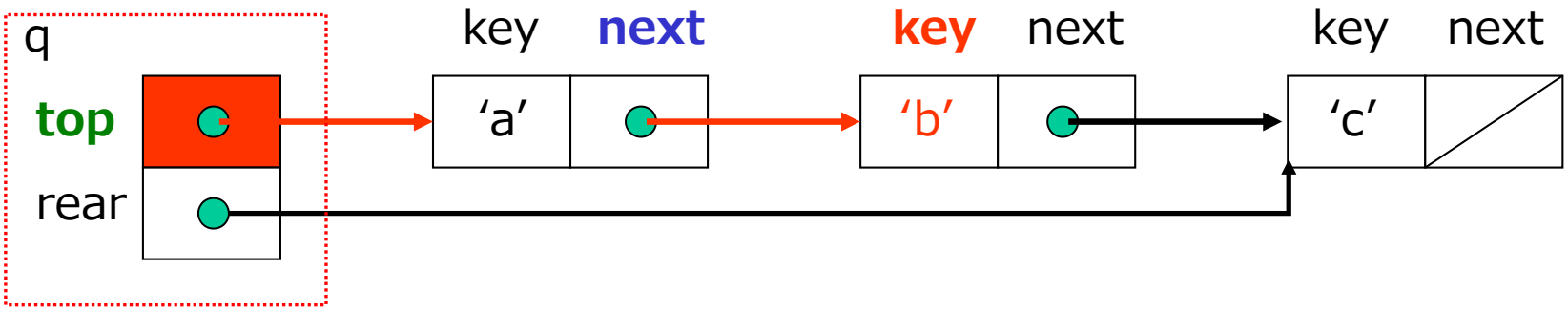
    例：cur = cur->next ;
  - リスト辿りながら、リストの各要素（key）を出力する
  - ループの終了判定はnextがNULLかどうかで判定する

必須課題5- 3（1/3)



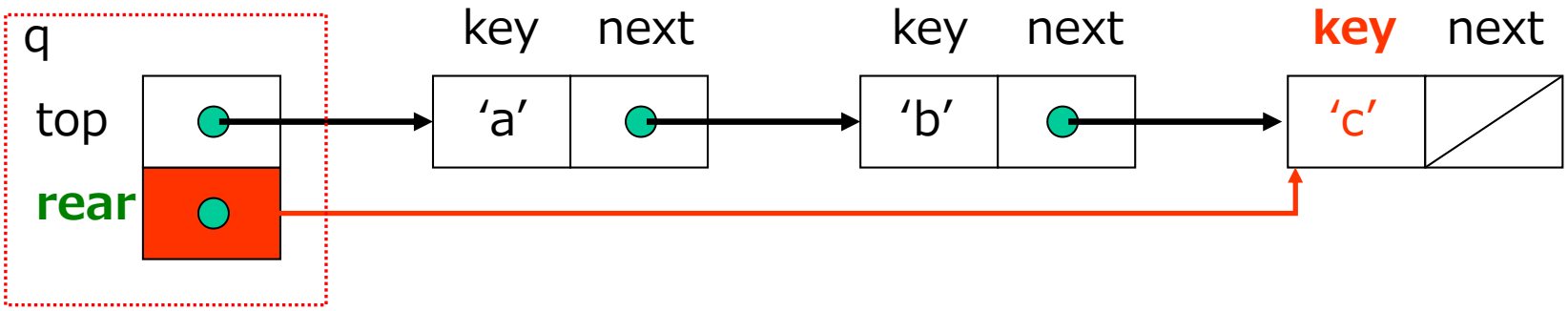

```
printf("%c¥n",q.top->key);
printf("%c¥n",q.top->next->key);
printf("%c¥n",q.rear->key);
```

必須課題5- 3 (2/3)



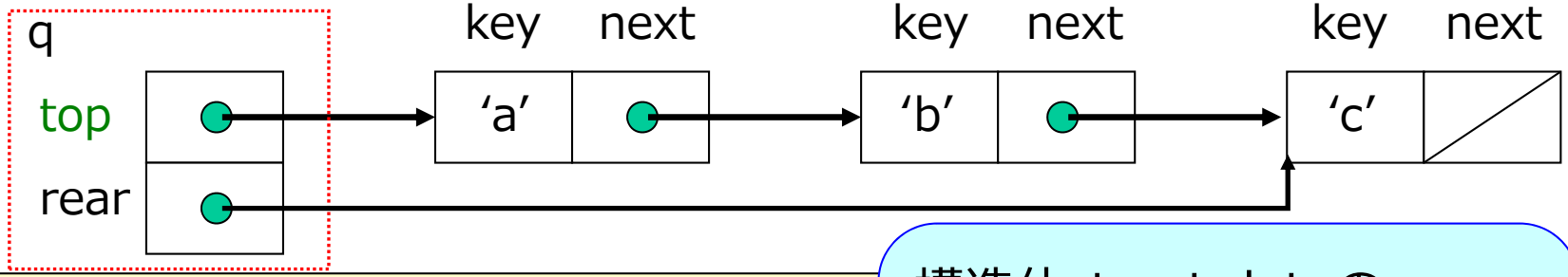

```
printf("%c¥n",q.top->key);
printf("%c¥n",q.top->next->key);
printf("%c¥n",q.rear->key);
```

必須課題5- 3 (3/3)




```
printf("%c¥n",q.top->key);
printf("%c¥n",q.top->next->key);
printf("%c¥n",q.rear->key);
```

必須課題5-3：参考プログラム（リストの初期化部分のみ）

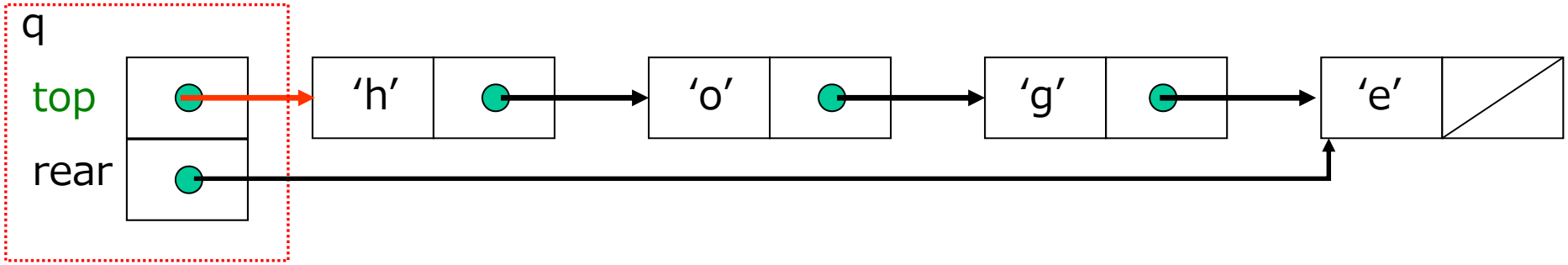



```
struct queue q;
q.top =  動的メモリ確保
q.top->key = 'a';
q.top->next = NULL;
q.rear = q.top;//rearの更新

q.rear->next =  動的メモリ確保
q.rear = q.rear->next; //rearの更新
q.rear->key ='b';
q.rear->next= NULL;

q.rear->next =  動的メモリ確保
q.rear = q.rear->next;
q.rear->key ='c';
q.rear->next=NULL;
```

構造体struct dataの
動的メモリ確保を行う
必須課題3-3を復習しよう。
エラー処理も記述が必要

必須課題5-４：出力例

リスト版キューの状態を出力する関数
void print_queue_list(struct queue q)

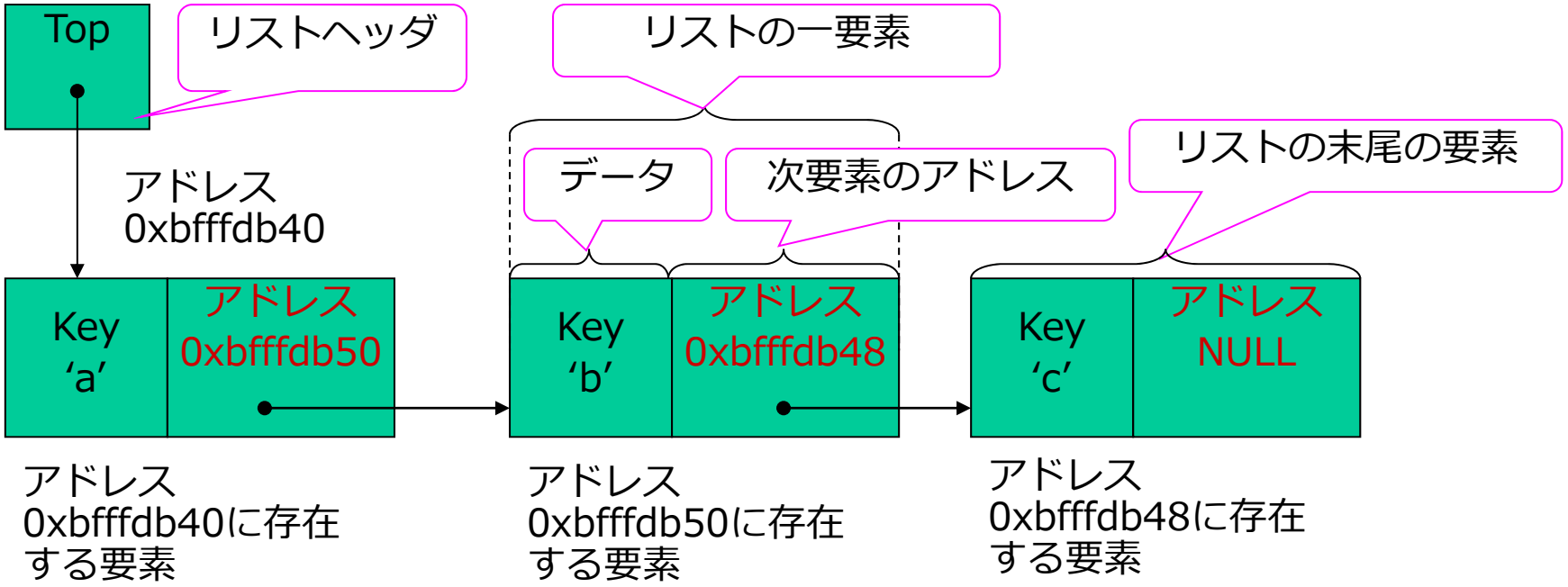



```
h<---TOP
o
g
e<---REAR
```

必須課題６-1とは
REARの表示位置が違うことに注意

補足：線状リスト

●複数のデータをひとまとまりとして扱うデータ構造
●配列との違い
- 型の違うデータを混在させられる (ただし本演習では同じ型のもののみ扱う)
- データが頻繁に追加・削除されるようなプログラムに向いている
- 物理的に非連続なメモリ領域群にデータを蓄えられる
  - ある1データに対応する要素に，次のデータが存在するメモリ領域の場所が含まれている
- あるデータには，先頭のデータを順にたどってしかアクセスできない



著者リスト

1. 安積 卓也（情報システム学科）
2. 大森 隆行（情報システム学科）
3. 原田 史子（情報システム学科）