アルゴリズムとデータ構造

第3回基本的なデータ構造　その1
（スタック、キュー）

第3回基本データ構造

- 今日の内容:
  - 抽象データ型としてのリスト
    - 変更演算 insertとdelete
  - スタック:配列(と構造体)による実装
  - キュー :配列(と構造体)による実装
- 次回
  - リストの実装:双方向連結リストによる実装(ポインタ)
    - スタックとキューのリストによる別の実装
- ポイント
  - 抽象データ型とその実装方法(プログラム)

第5回「二分探索木」でポインタ構造を再び学ぶ予定

復習（第1回）：

- アルゴリズム
  - 入力データから正しい出力を計算するために，一連の手順を記述したもの.
  - 正しさと効率が重要.
- データ構造
  - 計算のために，記憶領域に効率良くデータを格納するための配置法.
  - 時間と記憶領域を効率化.

https://ja.wikipedia.org/

Niklaus Wirth(1934-)はスイスの計算機科学者。コンピュータ言語Algol-W, Pascal, Modula-2の設計者。1984年にACMチューリング賞受賞。

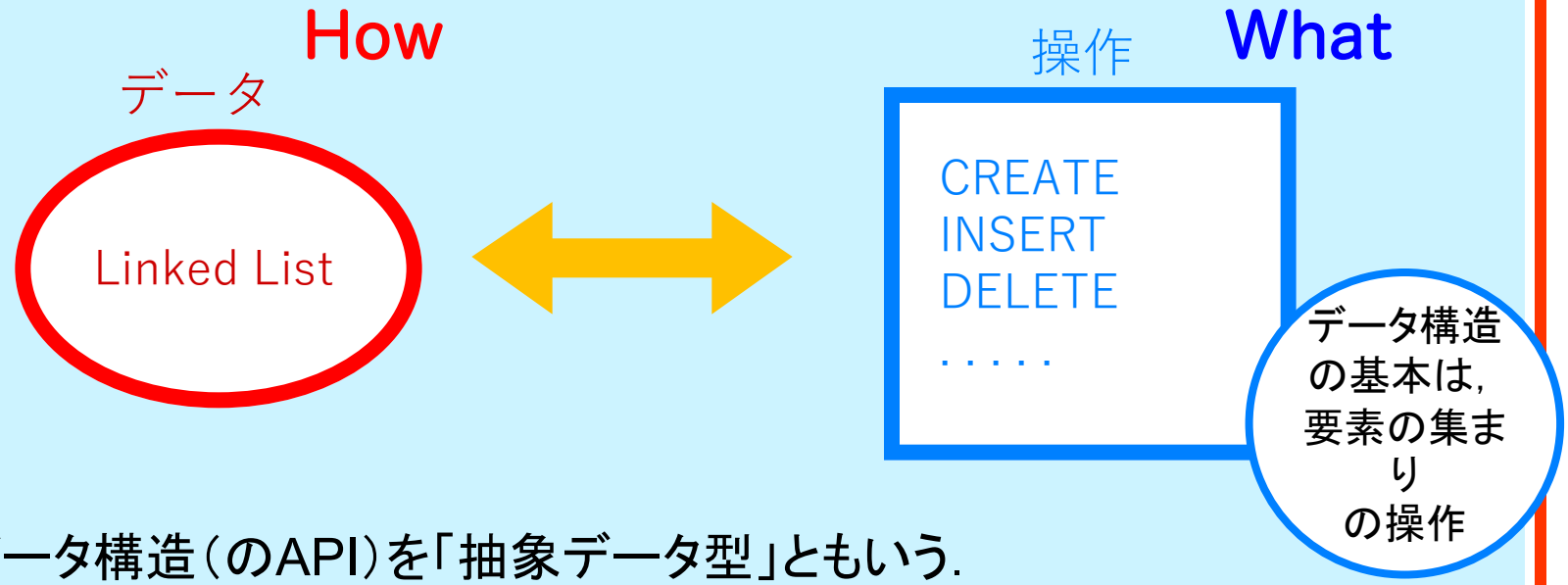
N. Wirth 1976

“アルゴリズム ＋ データ構造 ＝ プログラム”

Niklaus Wirth（ニクラウス・ビルト）が1976年に書いた本の題名

抽象データ型（Abstract Data Type）とは？

データ型を，それに適用される一組の操作で抽象的に定めたもの．



- データ構造（のAPI）を「抽象データ型」ともいう．
- データ構造には，「それは何か（What）」と「それをどのように実現するか？（How）」の二つの面がある．
- 現代的なプログラム言語やライブラリーはこの考え方に基づく．（例：C++，Java，Ruby, Python などなど）

授業で学ぶデータ構造の範囲

「プログラミング」で学ぶところ

機械語のデータ型
- ◆レジスタ値とその番地

機械語
（型がない）

基本データ型
- ◆char, int, large int, double,

昔の言語も持っている型
（C，Pascal）

構造データ型
- ◆配列（array），構造体（struct）

最近の言語（C++，Java, etc.）

基本的データ構造
- ◆スタック（stack），待ち行列（queue），リスト（list）

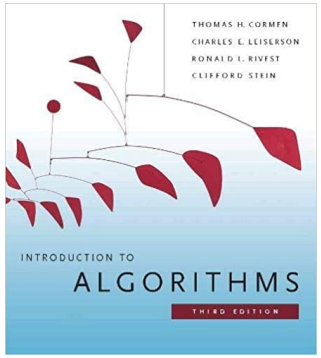
今日のトピック

先進的データ構造
- ◆二分探索木（binary search tree），平衡探索木（balanced search tree），ハッシュ表（hash table）

「アルゴリズムとデータ構造」の範囲

今日のネタ本

今日は，次の教科書にしたがって
基本データ構造を紹介します．



- Cormen, Leiserson, Rivest, and Stein
  - III部. データ構造，10章. 基本データ構造（日本語第1巻）
  - Introduction to Algorithms, 3rd Edition, MIT Press, 2009.
  - 現在，最も定評がある教科書．企業や競プロなどでの勉強会の定番．
    （日本語訳有，3巻組，近代科学社，第1巻4000円）

- 次も参考にしました：Aho, Hopcroft, Ullman
  - The Design and Analysis of Computer Algorithms, Addison Wesley, 1974.（最初のデータ構造の教科書の一つ）

今日のあらすじ

授業のはじめに，抽象データ型としてのリストを紹介する

それから特殊なリストであるスタックとキューを紹介する

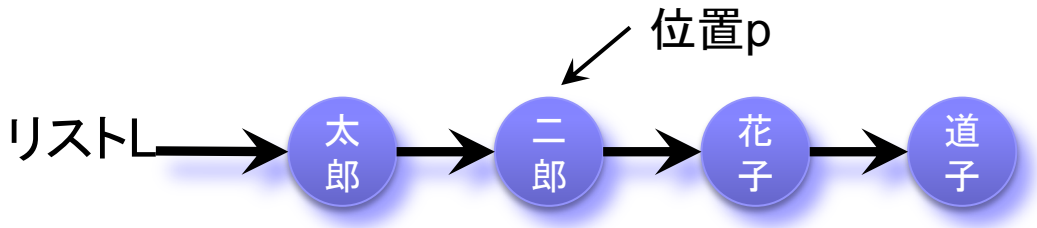
最後に，ふたたび，リストの実装に戻る

抽象データ型としてのリスト

基本

リストとは？（抽象データ型として）

0個以上の要素を一列にならべたもの



要素に順序があるところが，集合との違い

用語

- 空リスト：要素を含まないリストのこと
- リストの長さ：要素数 n
- A_i: 最初から i 番目の要素（$0 \leqq i \leqq n-1$）
- リスト中の場所を指示するための位置 p をもつ
  （要素へのポインタ）

基本

抽象データ型としての「リスト」に対する操作

- List L = create () : 空のリストを返す.

変更操作

- insert(L, p, x) : リストLの位置pの次に要素xを**挿入する**
- delete(L, p) : リストLの位置pの要素を**削除する**
- insert(L, x) : リストLの先頭位置に要素xを**挿入する**

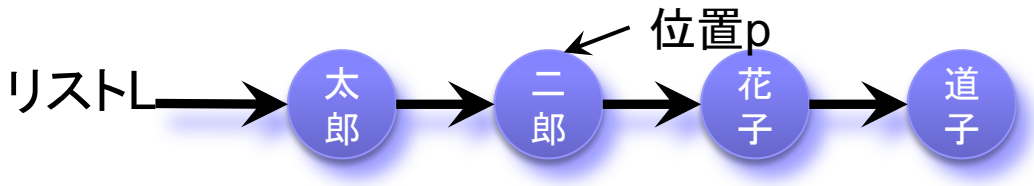
変更操作がある
データ構造を
「動的データ
構造」という
↔「静的データ構造」

探索操作

- search (L, x) : リストLに要素xが含まれてるかを1と0で返す

アクセス操作

- find(L, i) : リストLのi番目のセルの内容を返す（ランダムアクセス）
- last(L) : リストLの最後のセルの位置を返す
- next(L, p) : 位置pの1つ次のセルの位置を返す
- previous(L, p) : リストLにおいて、位置pの1つ前のセルの位置を返す



今日のあらすじ

授業のはじめに，抽象データ型
としてのリストを紹介する

それから特殊なリストである
スタックとキューを紹介する

最後に，ふたたび，リストの実装に戻る

スタック

基本

スタック(stack)

スタックとは

要素の挿入、削除がいつも先頭からなされるリスト

LIFO(last-in-first-out)

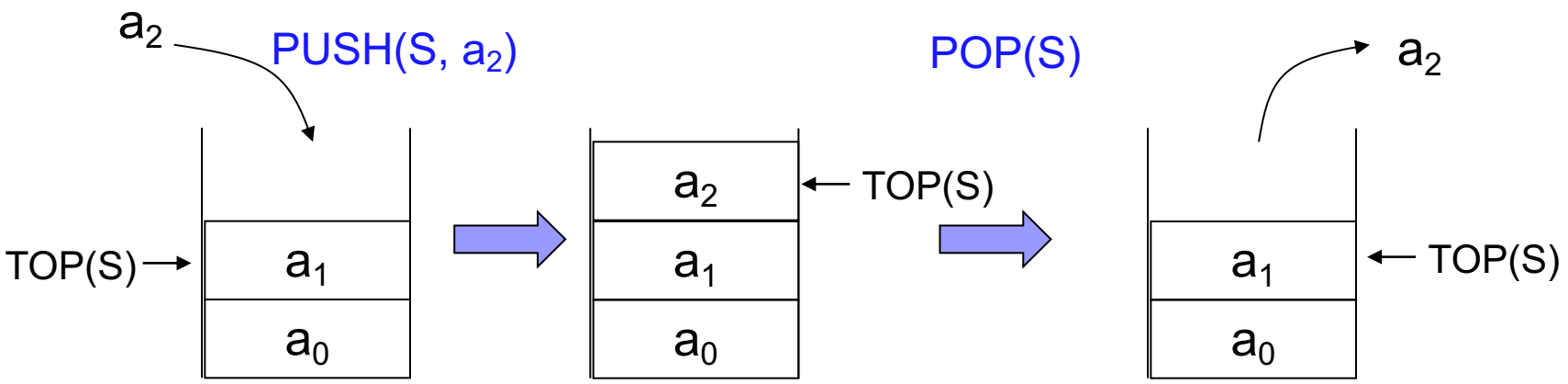
[基本操作]

TOP(S)　　スタックSの先頭の位置を返す

POP(S)　　スタックSの先頭の要素を削除

PUSH(S, x)　スタックSの先頭に要素xを挿入

理解のヒント：ポインタTOP(S)がどう変化するか動きを追ってみよう!



配列を用いたスタックの実現法

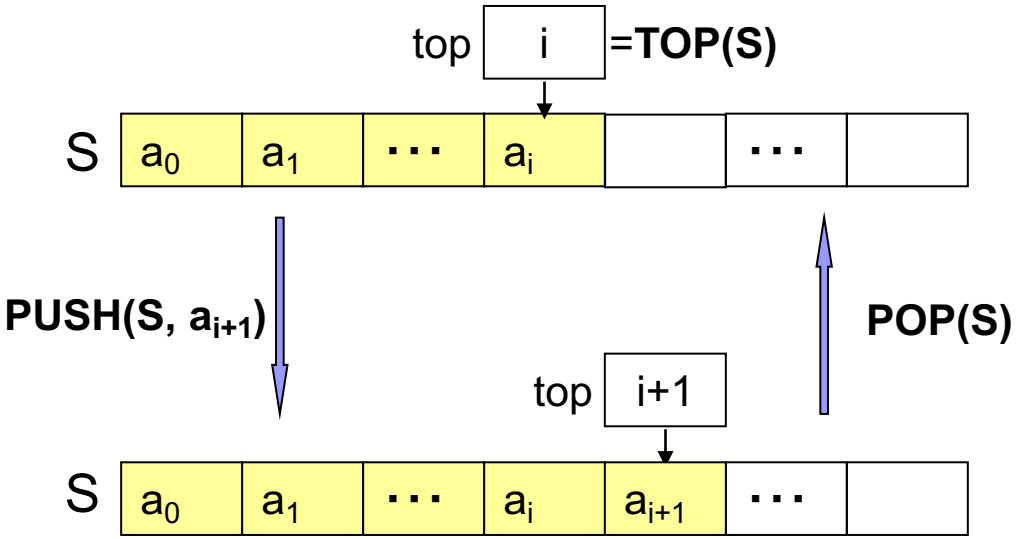
基本　配列による実現

- 次を用いる
  - スタックの頂上(トップ)を表す添字: int top
  - 要素を保持する配列: int S[MAXSTACK]
  - 配列の長さは定数 MAXSTACK に保持
- 次の演算を実現する
  - S = CREATE()
  - PUSH(S, x)
  - x = POP(S)

すべての操作の
時間計算量はΘ(1)

長所:実装が容易，効率良い
短所:固定長のスタックのみ

スタックの応用
- 関数呼び出し
- 根付き木/グラフの巡回
- 数式の評価(電卓)
- 構文解析



参考：配列を用いたスタックの実装（C言語）

Cコード：Windows/Mac/Linuxのコンソール/シェル上のふつうのcc/gccコンパイラで実行できます

```
/* スタックの定義 */
#define MAXSTACK 128 /*最大長さ*/
int top;
int S[MAXSTACK];

/* スタックの初期化 */
void create() {
  top = -1;  /* 空にする */
}

/* スタックへの要素xのプッシュ */
void push(int x) {
  if (top >= MAXSTACK - 1) {
    /* スタックが満員？ */
    printf("オーバーフロー!¥n");
    exit(1);
  }
  top++;
  S[top] = x;
}
```

エラー処理 (*1)

(*1) 2022.4.21の授業時に間違っていましたので，訂正しました．以前は，配列Sがfullの際に書き込みをしていました．ご指摘感謝します．

```
/* スタックからの要素のポップ */
int pop() {
  int x;
  if (top == -1) { /* スタックが空か */
    printf("アンダーフロー!¥n");
    exit(1);
  }
  x = S[top];
  top--;
  return x;
}

/* 主プログラム */
int main() {
  create();
  push(1); push(2); push(3);
  pop();
  push(4); push(5);
  print();
}
```

エラー処理

実装メモ：一つのスタックだけの実装です．演算引数のスタックSは省略しています．

出力関数print()を書いてみましょう

上級編として，ヘッダーファイルを付けたり (C言語)，クラスのオブジェクトにして (C++)，ライブラリ化してみましょう．

発展

スタックの応用例1: 関数呼び出し

プログラム(階乗n!の計算)

```
int factorial(int n) {
  if(n==1) return 1;
  else      return n*factorial(n-1);
}
```

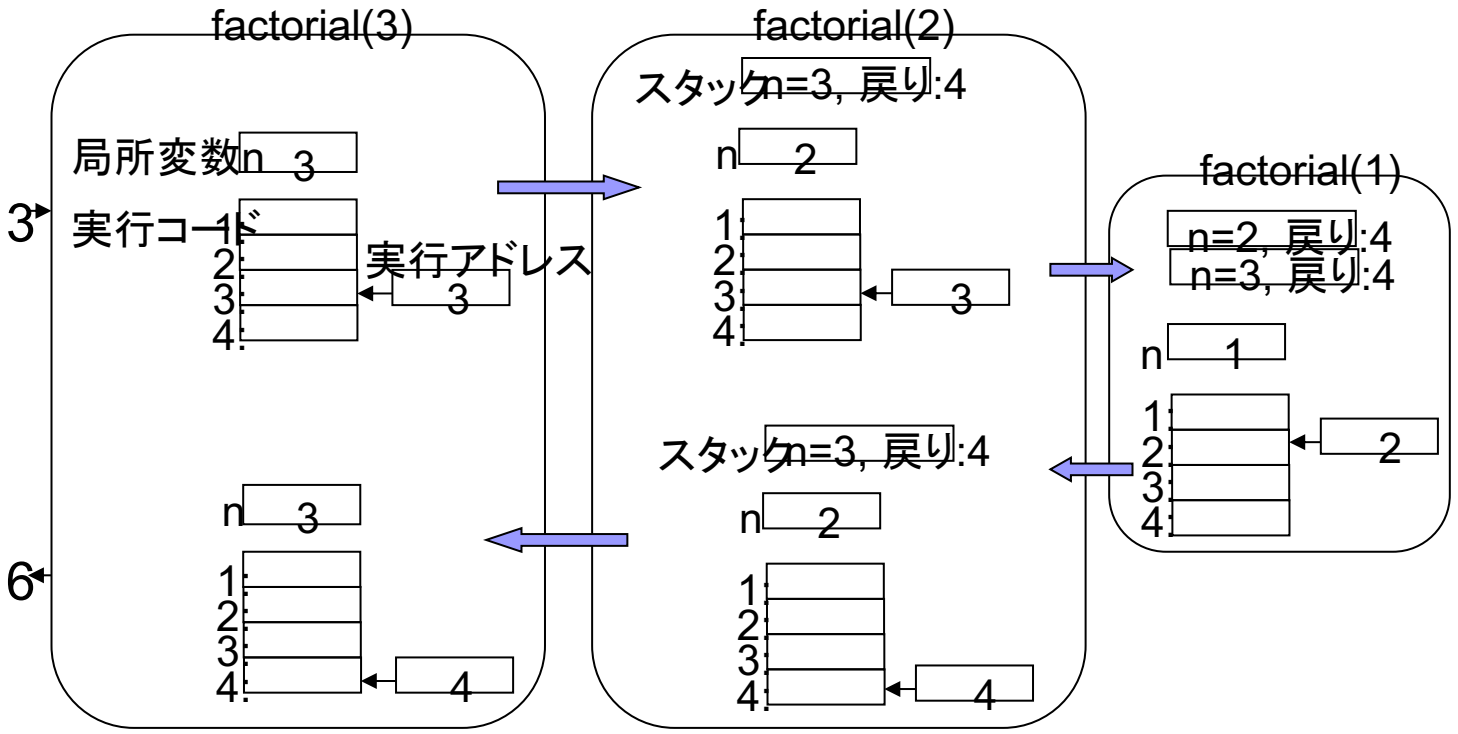
実行コードシークエンス

```
1: if n≠1 then goto 3
2: return 1
3: r=factorial(n-1)
4: return n*r
```

関数呼び出しの前後で，引数と返り値をスタックに積む

ここでスタックにpush

ここでスタックをpop



☞ 付録:スタックの応用例2:逆ポーランド記法電卓(演習課題2の問4*)

今日のあらすじ

授業のはじめに，抽象データ型としてのリストを紹介する

それから特殊なリストであるスタックとキューを紹介する

最後に，ふたたび，リストの実装に戻る

キュー（待ち行列）

基本

待ち行列（キュー）とは？

要素の挿入は最後尾、削除は先頭からなされるリスト

FIFO(first-in-first-out)ともいう

理解のヒント：銀行の窓口の「順番待ちの行列」はキュー

[基本操作]

- Q = CREATE()　　新しいキューを生成する
- ENQUEUE(Q, x)　　要素xをキューQの最後尾に入れる
- DEQUEUE(Q)　　先頭の要素をキューQから除く

「エンキュー」、「デキュー」と読む

基本

待ち行列(キュー, queue)

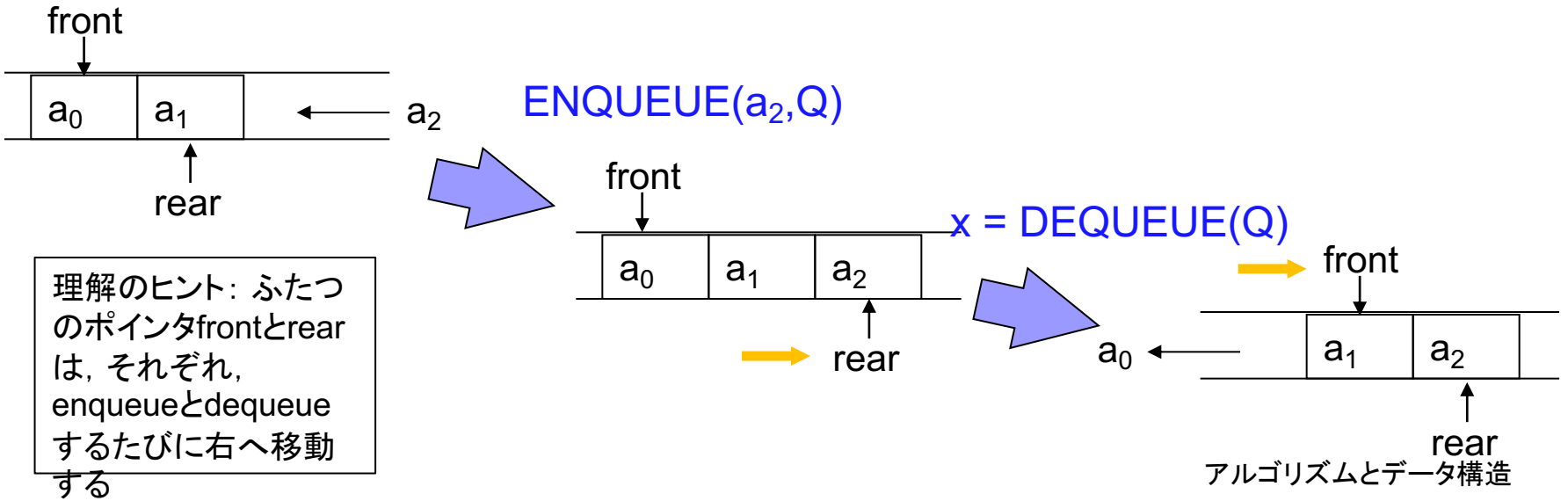
要素の挿入は最後尾、削除は先頭からなされるリスト

FIFO(first-in-first-out)ともいう

先頭(front)と末尾(rear)の二つのポインタをもつ

[基本操作]
ENQUEUE(Q, x) 要素xをキューQの最後尾に入れる
DEQUEUE(Q) 先頭の要素をキューQから除く

キューの応用
- 通信バッファ
- 根付き木/グラフの巡回



理解のヒント: ふたつのポインタfrontとrearは, それぞれ, enqueueとdequeueするたびに右へ移動する

配列によるキューの実現

基本

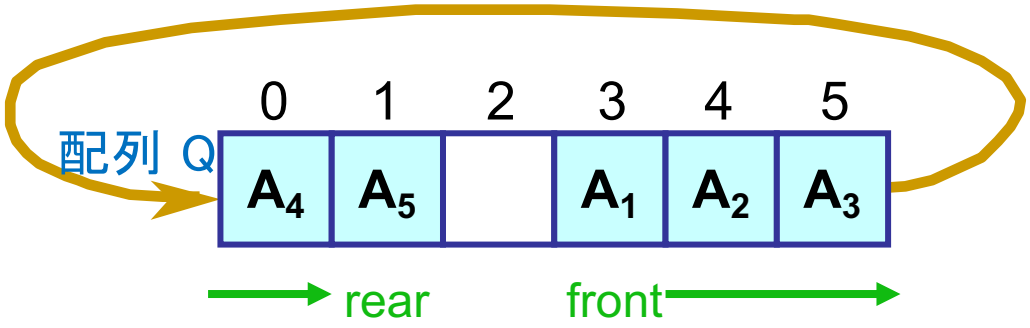
基本アイディア：配列で巡回リストを表わす．

- キューの長さ n が限定された場合
- 先頭と末尾がつながって，輪になったリスト
- 要素の位置を，n の剰余演算 (mod n)で定める(*1)．

- この方法は，たいへん上手いアイディア．簡単で効率良いので，覚えておくと便利．
- OSや通信機器では，有限バッファの実装として広く使われている

$$front \text{ から } i \text{ 番目の要素} = Q[\ (front + i) \bmod n\ ]$$

「リングバッファ」とも呼ばれる



実装メモ (*1): 実際のプログラムでは，mod nの剰余演算の代わりに，enqueueとdequeue演算のたびに，if文を用いてfrontとrearを正しく更新する．

例：長さ$n = 6$ でfrontが$f = 3$ にあるとき，その4マス先の位置であるrear r は，$r = (f + 4) \bmod 6 = (3 + 4) \bmod 6 = 7 \bmod 6 = 1$なので，rearの位置は$r = 1$.

キューの実現法(配列による実現)

基本

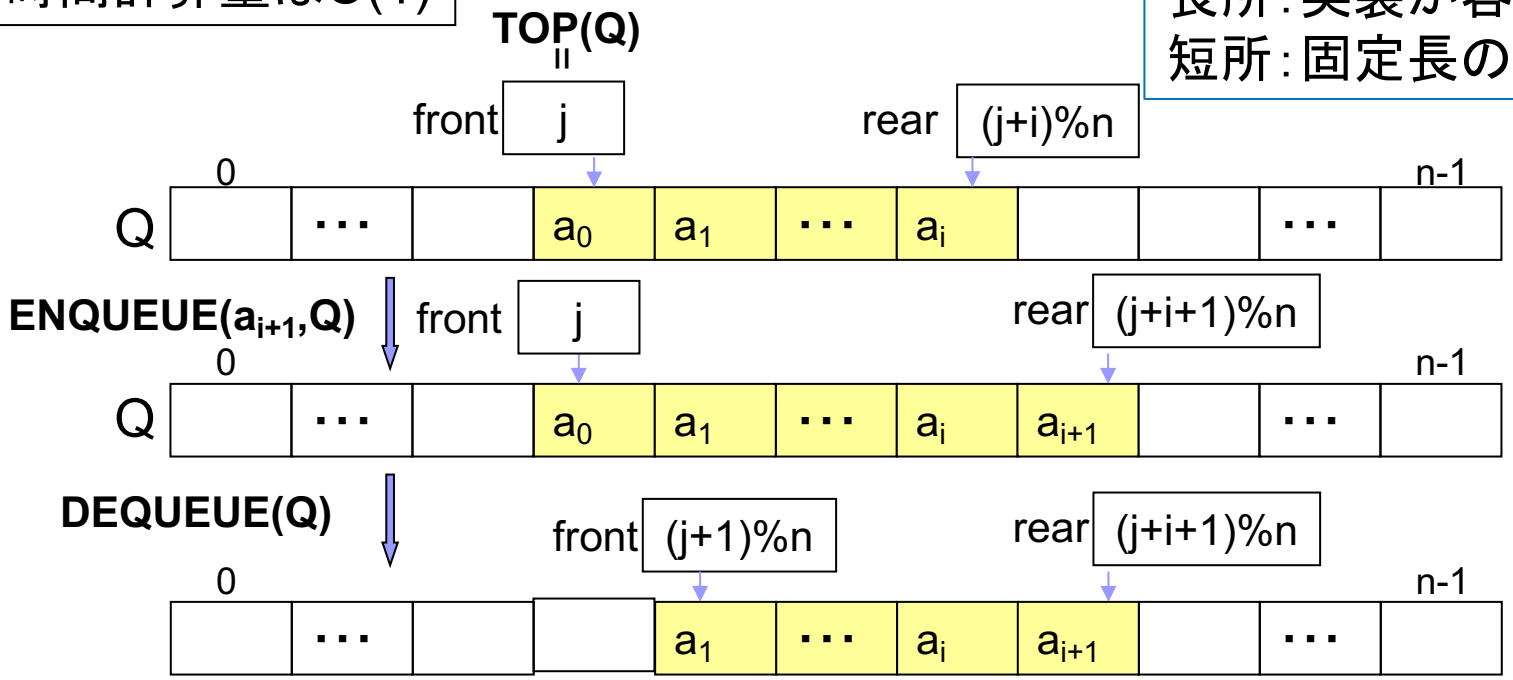
配列による実現

- 次の演算を実現する
  - TOP(Q)
  - ENQUEUE(Q, x)
  - DEQUEUE(Q)

すべての操作の時間計算量はΘ(1)

- 次のふたつを用いる
  - キューの先頭(front)と末尾(rear)を表す添字の対: int front, rear
  - 要素を保持する配列: int Q[MAXQUEUE]
  - 配列の長さは定数 MAXQUEUE に保持

長所:実装が容易,効率良い
短所:固定長のキューのみ



配列による実現

参考：配列を用いたキューの実装（C言語）

Cコード：Windows/Mac/Linuxのコンソール/シェル上のふつうのcc/gccコンパイラで実行できます

```
/* キューの定義 */
#define MAXQ 128 /*最大長さ*/
int front; /* 先頭 (フロント). 0..MAXQ-1の範囲 */
int rear;  /* 末尾（リア）. 0..MAXQ-1の範囲 */
int Q[MAXQ]; /* 要素の配列 */

/* キューの初期化 */
void create() { front = rear = 0; }

/* 要素xの挿入*/
void enqueue(int x) {
  if ((rear +1) % MAXQ  == front) {
    /* キューが満員か */
    printf("オーバーフロー!¥n");
    exit(1);
  }
  Q[rear] = x;
  rear = (rear + 1) % MAXQ;
}
```

エラー処理

変数rearを進める時の剰余演算

```
/* キューからの要素のデキュー（削除）*/
int dequeue() {
  int x;
  if (rear == front) { /* キューが空か */
    printf("アンダーフロー!¥n");
    exit(1);
  }
  x = Q[front];
  front = (front + 1) % MAXQ;
  return x;
}
```

エラー処理

変数rearを進める時の剰余演算

実装メモ： 一つのキューだけの実装です．演算引数のキューQは省略しています．
出力関数print()を書いてみましょう．

```
/* 主プログラム */
int main() {
  create(); enqueue(5); enqueue(3); enqueue(6);
  dequeue(); enqueue(3);  enqueue(7);
  print();
}
```

実装メモ： 上級編として，ヘッダーファイルを付けたり (C言語)，クラスのオブジェクトにして (C++)，ライブラリ化してみましょう．

修正2022.4.21: 関数enqueのエラー処理の条件部を修正しました．前の版は，front == 0のときに(front – 1)が負となり誤動作しました．s

今日はスタックまでとして

双方連結リストについては，
次の第4回に説明します

今日のあらすじ

授業のはじめに，抽象データ型としてのリストを紹介する

それから特殊なリストであるスタックとキューを紹介する

最後に，ふたたび，リストの実装に戻る

双方連結リストを用いたリストの実現

第3回基本データ構造

- 今日の内容:
  - 抽象データ型としてのリスト
    - 変更演算 insertとdelete
  - スタック:配列(と構造体)による実装
  - キュー :配列(と構造体)による実装
- 次回
  - リストの実装:双方向連結リストによる実装(ポインタ)
    - スタックとキューのリストによる別の実装
- ポイント
  - 抽象データ型とその実装方法(プログラム)

第5回「二分探索木」でポインタ構造を再び学ぶ予定