基本的なデータ構造 (2-2)

キュー

アルゴリズムとデータ構造B

第09回

本日の予定

基本的なデータ構造その２：

- **スタック**（後入れ先出し方式：**LIFO**, Last In First Out）
- **キュー**（先入れ先出し方式：**FIFO**, First In First Out）

スタック，キュー

アルゴリズムとデータ構造Bの講義では，まず1次元配列で実現

- 配列よりも機能が劣る
- スタックは関数呼出しの実行等に利用（教科書 p.146-147）

- **スタック**：スタックポインタ（**ptr**）の指定する要素のみ
  データの入出力が可能
- **キュー**：データの出力は先頭（**front**）からのみ，
  データの入力は末尾（**rear**）にのみ可能

スタックのプログラミング

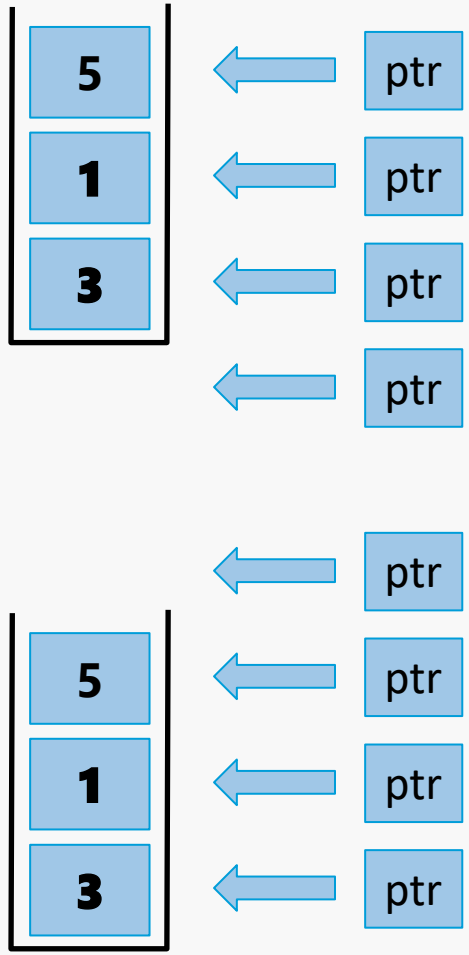
データ構造

- 配列　`int stk[MAX];`
- スタックポインタ　`int ptr =-1 ;`

} 簡単のため グローバル変数 (要改善)

アルゴリズム

- データの入力　`int Push(int x);`
  - ✓ スタックポインタの位置にデータを格納する
  - ✓ スタックポインタを1つずらす（進める）
- データの出力　`int Pop();`
  - ✓ スタックポインタの位置からデータを取り出す
  - ✓ スタックポインタを1つずらす（戻す）



ずらしてから入れる（出す）のか、入れ（出し）てからずらすのか？
⇒ ptr の初期値によってアルゴリズムが異なる

アルゴリズム（をもう少し詳しく）

`int Push(int x);`

- スタックが `full` でないとき，`++ptr` した後，`stk[ptr]` にデータ `x` を格納し，`0` を返す
- スタックが `full` のとき，エラー処理として `-1` を返す
  - ✓ これ以上データを追加できない（`-1` が返されたら `main` で “Stack full” と表示）

`int Pop();`

- スタックが `empty` でないとき，`stk[ptr]` の値を返し，`ptr--`
- スタックが `empty` のとき，エラー処理として `-1` を返す
  - ✓ 空の時はデータを取り出せない（`-1` が返されたら `main` で “Stack empty” と表示）

Push (ptr = -1 ver.)


```
int Push(int x) {
    if (ptr < MAX-1)
        stk[++ptr] = x;
    else
        return -1;

    return 0;
}
```

方針：**ずらしてから入れる**
スタックが full のときは -1 を返す

Pop (ptr = -1 ver.)


```
int Pop() {
    if (ptr >= 0)
        return stk[ptr--];
    else
        return -1;

}
```

方針：**出してからずらす**
スタックがemptyのときは -1 を返す

Push (ptr = 0 ver.)


```
int Push(int x) {
    if (ptr < MAX)
        stk[ptr++] = x;
    else
        return -1;

    return 0;
}
```

方針：**入れてからずらす**
スタックが full のときは -1 を返す

Pop (ptr = 0 ver.)


```
int Pop() {
    if (ptr > 0)
        return stk[--ptr];
    else
        return -1;

}
```

方針：**ずらしてからから出す**
スタックがemptyのときは -1 を返す

課題３　ファイルからの読み込み

- 前提：ファイルには空白区切りで，数値，演算子，式の終わりを示す記号 “end” が記入
  ⇨ **文字列として読み込めば良い（文字列は空白で自動的に区切られる）**
- fpでファイルオープン，char c[10];に読み込むとして，
  - ✓ fscanf() を用いて：fscanf(fp, “%s”, c);
- ファイルの終端は **EOF** で表される．よってファイル終端まで読み込むためには
  - ✓ while(fscanf(fp, “%s”, c) != **EOF**){ ...

とすれば良い．while の中 {...} で c に読み込んだ
文字列（数値 or 演算子 or 式の終わりの記号）を処理可能

ポイント以外の箇所は省略して書いています

課題３　数値と演算子の扱い

- c に読み込んだ文字列が数値であるか，演算子であるか，
  式の終わりの記号 “end” であるかを区別する必要
  - ✓ 演算子（+ or - or * or /）であるか，“end” であるか，を判定して，
    最後に残ったものは数値と判定する，とすると良いと思う
  - ✓ 文字列を比較する関数 strcmp で場合分けすると良い
    - #include <string.h> が必要
    - if (strcmp(c, “end”) == 0) {... // 読み込んだ文字列と “end” が等しい

ポイント以外の箇所は省略して書いています

課題３　数値と演算子の扱い

- 文字列が演算子である場合：計算して PUSH する

```
int a = Pop(); int b = Pop();
if (strcmp(c, “+”) == 0) { x = Push(a + b); }
```

- 文字列が数値である場合：PUSH する
  文字列 → 整数の変換　が必要
  - ✓ `// char c[10] に “24” がセットされているとして`

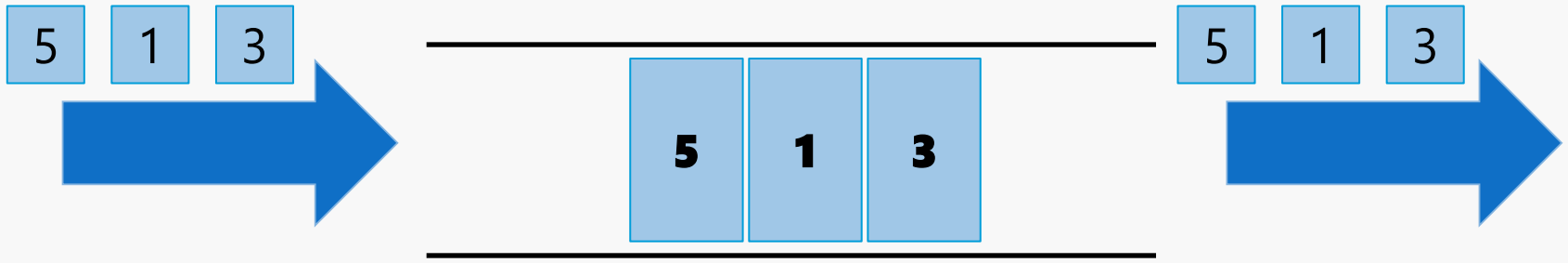
```
int num = atoi(c);  // num 24 がセットされる
x = Push(num);
```

ポイント以外の箇所は省略して書いています

基本的なデータ構造（２－２）：キュー

今日の内容はここから

キューの概念図



先入れ先出し方式
FIFO（First In First Out）

キューの概念図

画像が更新されたみたいで，
今はもう見れない…

Amazon｜アーネスト 【日本製】 缶ビール ホルダー 冷えたビールが取り出しやすい/スッキリ収納/棚上も収納に (上にも置ける缶ストッカー) 大手飲食店愛用ブランド A-76572｜キッチンストッカー オンライン通販

配列を利用して効率よく実現するには？

良くない例

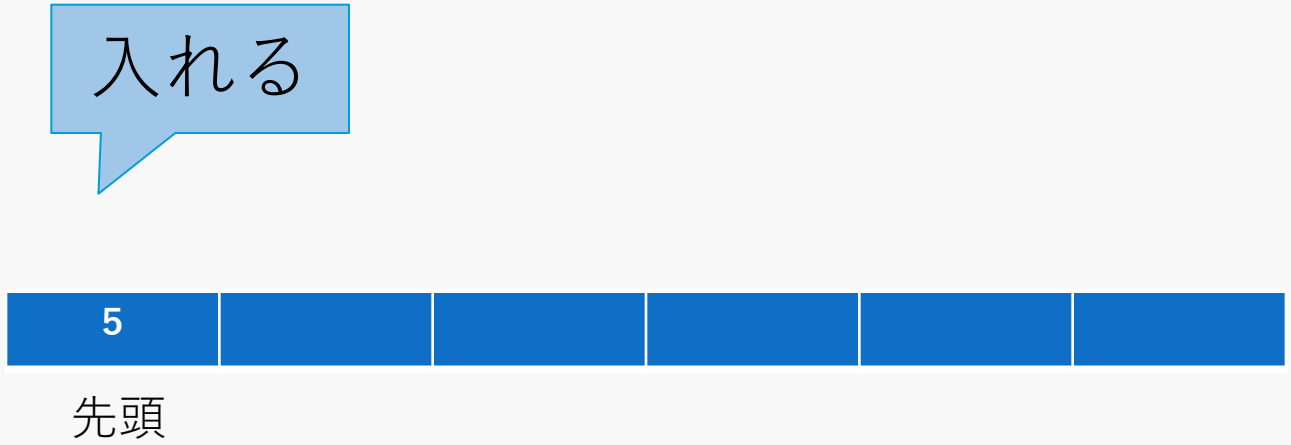
- 先頭を固定して，取り出すときデータを移動させる

先頭

配列を利用して効率よく実現するには？

良くない例

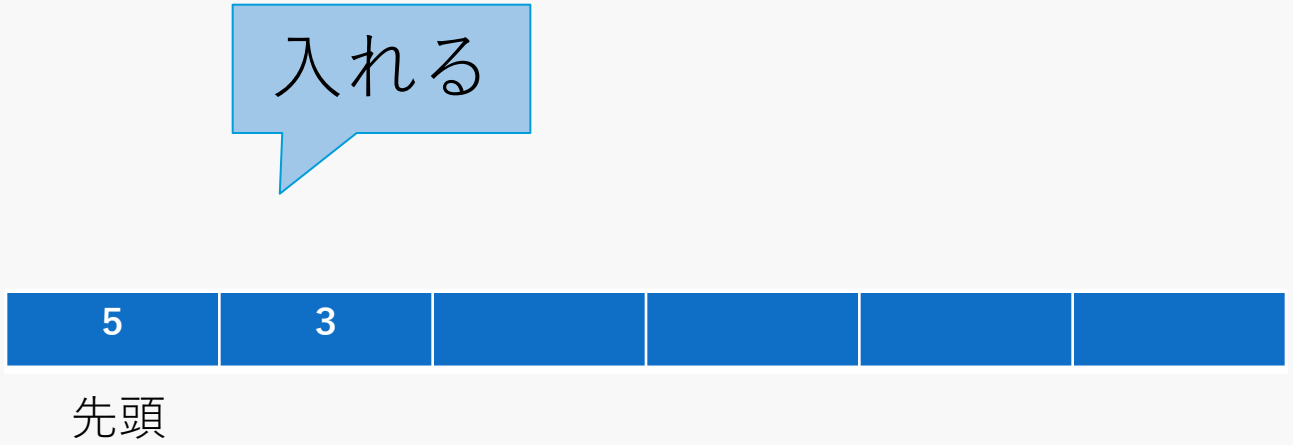
- 先頭を固定して，取り出すときデータを移動させる



配列を利用して効率よく実現するには？

良くない例

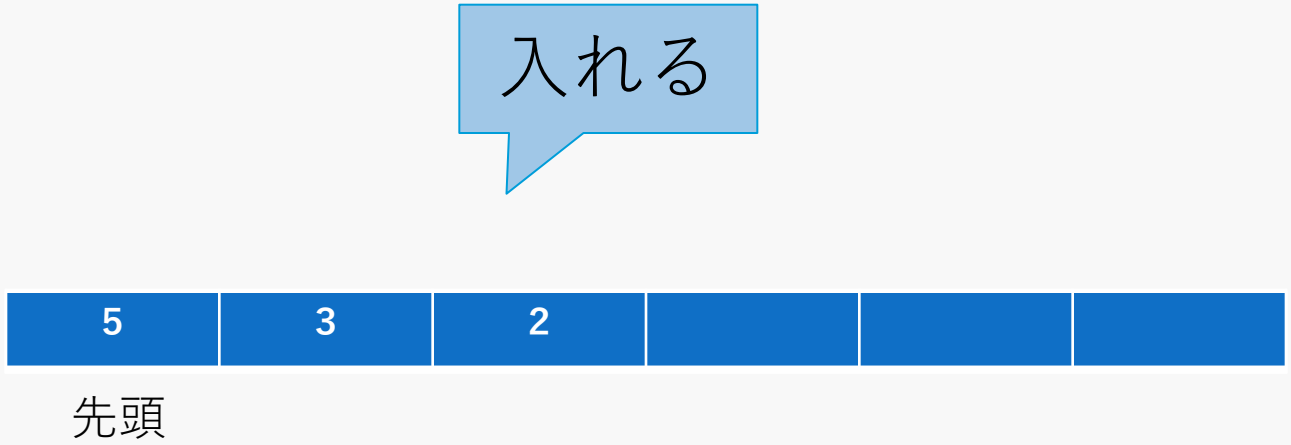
- 先頭を固定して，取り出すときデータを移動させる



配列を利用して効率よく実現するには？

良くない例

- 先頭を固定して，取り出すときデータを移動させる



配列を利用して効率よく実現するには？

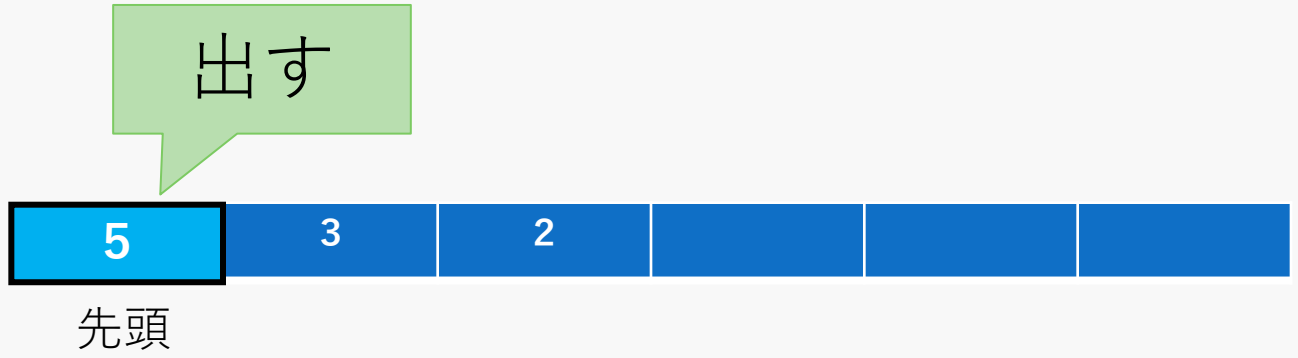
良くない例

- 先頭を固定して，取り出すときデータを移動させる
- データの移動に **$O(n)$ の計算量が必要**

配列を利用して効率よく実現するには？

良い例

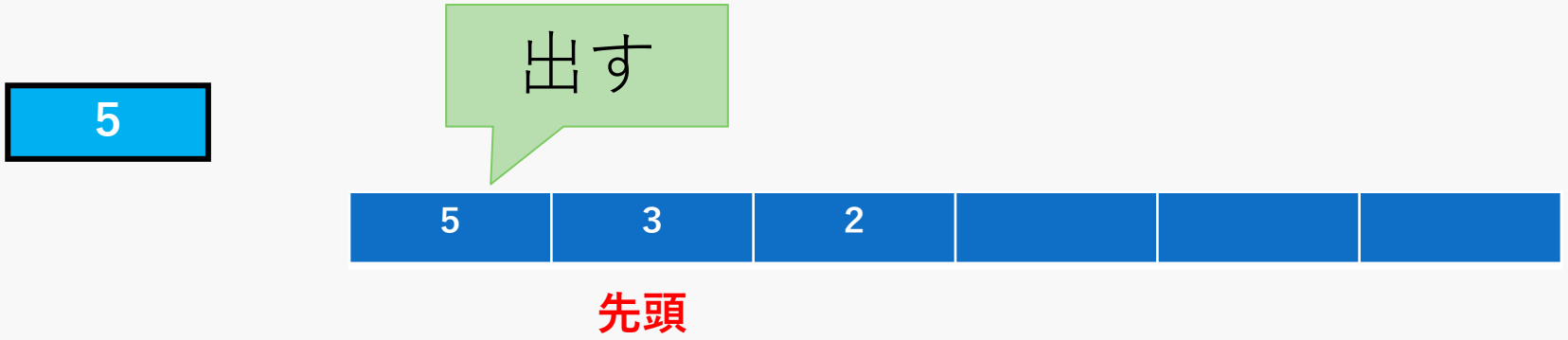
- データを取り出すとき，残りのデータを移動させるのでなく取り出す場所（先頭）を移動する



配列を利用して効率よく実現するには？

良い例

- データを取り出すとき，残りのデータを移動させるのでなく取り出す場所（先頭）を移動する
- **$O(1)$ で取り出し可能**



キューのプログラミング（基本）

データ構造

- 配列 **`int que[MAX];`**
- 先頭，末尾 **`int front, rear;`**
- 要素数 **`int num;`**

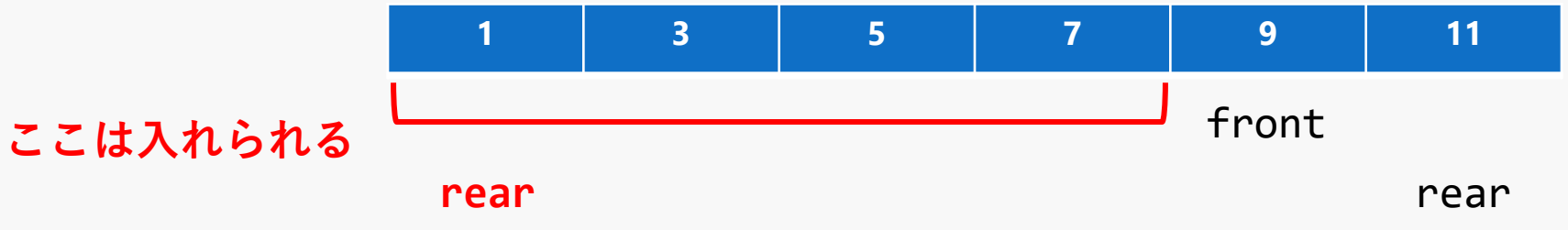
} グローバル変数
（**`front, rear, num`** は **`0`** で初期化）

アルゴリズム

- データの入力 **`int Enque(int x);`**
  - ✓ 末尾の位置にデータを格納する
  - ✓ 末尾を１つずらす（進める），
    要素数を１つ**増やす**
  - ✓ **`rear`** が末尾まで行ったら先頭に戻す
- データの出力 **`int Deque();`**
  - ✓ 先頭の位置からデータを取り出す
  - ✓ 先頭を１つずらす（進める），
    要素数を１つ**減らす**
  - ✓ **`front`** が末尾まで行ったら先頭に戻す

リンクバッファによるキューの実現（先頭に戻す）

キューの先頭（front），もしくは末尾（rear）が
配列の範囲を超えてしまったとき：



- front もしくは rear を 0 に戻す

```
rear++;
if (rear == MAX)
  rear = 0;
```

Enque のアルゴリズムの確認


```
int Enque(int x) {



}
```

方針：入れてからずらす
キューが full のときは -1 を返す
rear が配列の範囲を超えたとき 0 に戻す

Enque のアルゴリズムの確認


```
int Enque(int x) {
    if (キューがfullでない) {
        xを入力してからずらす;
        if (rearが超えた)
            rearを0に戻す;
        データ数を一つ増やす;
    } else
        return -1;

    return 0;
}
```

方針：入れてからずらす
キューが full のときは -1 を返す
rear が配列の範囲を超えたとき 0 に戻す

Enque のアルゴリズムの確認


```
int Enque(int x) {
    if (num < MAX) {
        que[rear++] = x;
        if (rear == MAX)
            rear = 0;
        num++;
    } else
        return -1;

    return 0;
}
```

方針：入れてからずらす
キューが full のときは -1 を返す
rear が配列の範囲を超えたとき 0 に戻す

Deque のアルゴリズムの確認

方針：出してからずらす
キューがemptyのときは -1 を返す
front が配列の範囲を超えたとき先頭に戻す

```
int Deque() {



}
```

Deque のアルゴリズムの確認

方針：出してからずらす
キューがemptyのときは -1 を返す
front が配列の範囲を超えたとき先頭に戻す

```
int Deque() {
    if (キューがemptyでない) {
        tmpにデータを取り出してからずらす;
        if (frontが超えた)
            frontを先頭に戻す;
        データ数を一つ減らす;
        return tmp;
    } else
        return -1;
}
```

Deque のアルゴリズムの確認

方針：出してからずらす
キューがemptyのときは -1 を返す
front が配列の範囲を超えたとき先頭に戻す

```
int Deque() {
    if (num > 0) {
        int tmp = que[front++];
        if (front == MAX)
            front = 0;
        num--;
        return tmp;
    } else
        return -1;
}
```

本日の演習　ex9.c

キューの実現

- Enque(), Deque() 関数を完成させる
- front, rear, num 初期化用の Initialize() 関数,
  キュー表示用の Display() 関数を完成させる
  - ✓ Display() 関数の仕様

```
front->     0:    50
            1:   100
            2:     0  <-rear
            3:     0
            4:     0
```

キューの中身を順に出力
要素番号（6桁），コロン，値（6桁）
frontの位置に front->（左側）
rearの位置に <-rear（右側）

- 以下のことが分かる出力結果をソースコードに張り付けて提出
  - ✓ データを正しく追加・取り出すことができる
  - ✓ キューが FULL / EMPTY の場合に，それぞれエラー表示がされる
  - ✓ front / rear が配列の範囲を超えたとき，0 に戻る