基本的なデータ構造 (2-1)

スタック

アルゴリズムとデータ構造B

第07回

本日の予定

課題２の確認

基本的なデータ構造（２）

1. **スタック**（後入れ先出し方式：**LIFO**, Last In First Out）
   - ✓ 教科書 p. 146～
2. **キュー**（先入れ先出し方式：**FIFO**, First In First Out）

課題２の確認

- m行n列の二次元配列の動的生成
  - ✓ n, m は scanf で取得

```
life = malloc(sizeof(int *) * m);
for (int i = 0; i < m; i++) {
    life[i] = malloc(sizeof(int) * n);
}
```

- life の各セルに対して，隣接する8つのセルの状態を確認，各セルの状態を更新

```
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        for (int k = i-1; k <= i+1; k++) {
            for (int l = j-1; l <= j+1; l++) {
                if (k < 0 || l < 0 || k >= m || l >= n || (k == i && l == j)) {
                    // 何もしない
                } else {
                    // 生きたセルの数を数える
```

基本的なデータ構造（2－1）：スタック

今日の内容はここから

データ構造概要

- 表（テーブル）
  - ✓ 2次元配列で実現，構造体配列でも実現可
- 棚（スタック）
  - ✓ 今日の内容：1次元配列で実現
- 待ち行列（キュー）
  - ✓ 次週の内容：1次元配列で実現
- 線形リスト（リンクリスト）
- 木構造（ツリー）
- グラフ

データ構造概要

- 表（テーブル）
  - ✓ 2次元配列で実現，構造体配列でも実現可
- **棚（スタック）**
  - ✓ **今日の内容：1次元配列で実現**
- 待ち行列（キュー）
  - ✓ 次週の内容：1次元配列で実現
- 線形リスト（リンクリスト）
- 木構造（ツリー）
- グラフ

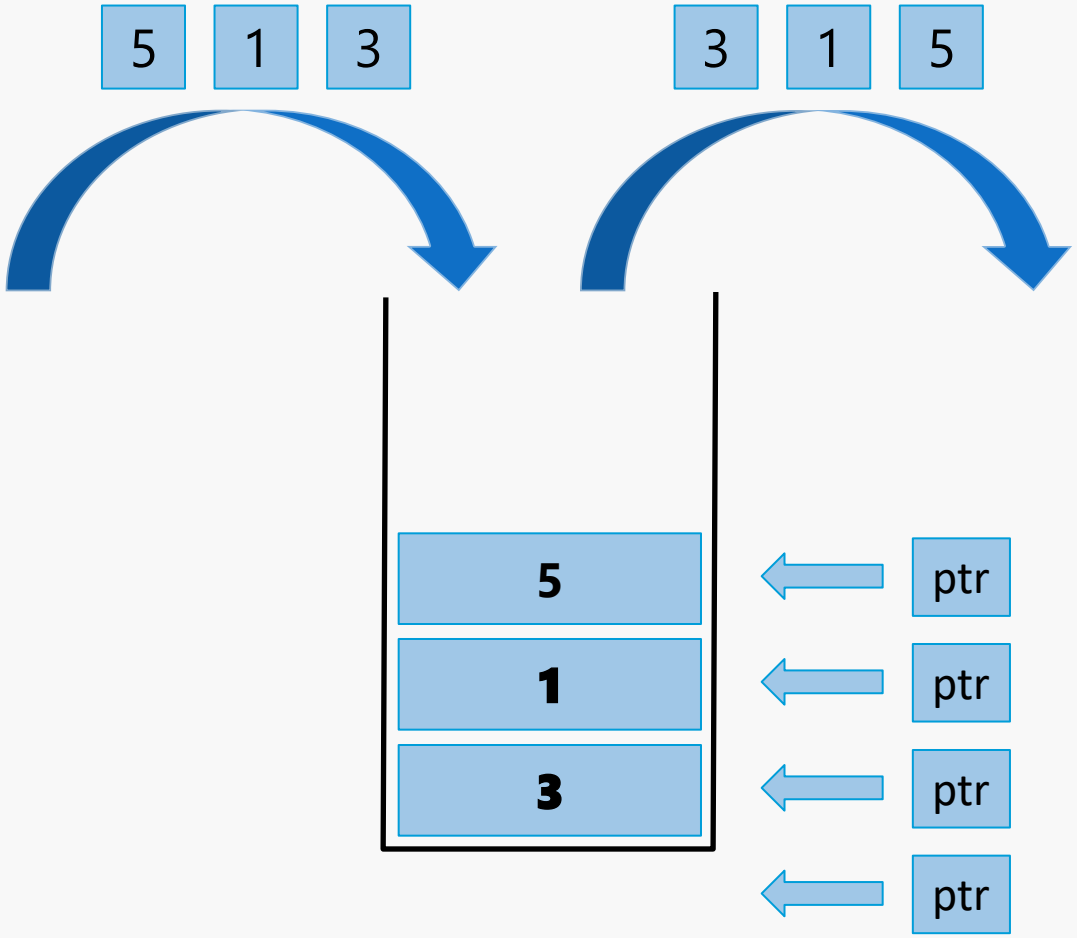
スタック，キュー

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

スタックの概念図



スタックのプログラミング

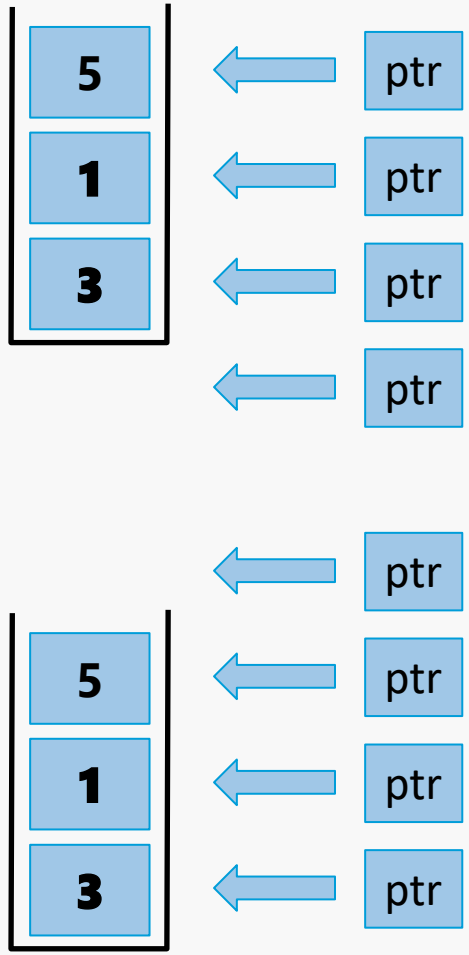
データ構造

- 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 のアルゴリズムの確認


```
int Push(int x) {



}
```

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

Push のアルゴリズムの確認


```
int Push(int x) {
    if (スタックがfullでない)
        ptr++;
        stk[ptr] = x;
    else
        // -1 を返す

    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

Push のアルゴリズムの確認


```
int Push(int x) {
    if (ptr < MAX-1)

        stk[++ptr] = x;
    else
        // -1 を返す

    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 のアルゴリズムの確認


```
int Pop() {



}
```

方針：出してからずらす
　　スタックがemptyのときは -1 を返す

Pop のアルゴリズムの確認


```
int Pop() {
    if (スタックがemptyでない)
        return stk[ptr--];
    else
        // -1 を返す

}
```

方針：出してからずらす
スタックがemptyのときは -1 を返す

Pop のアルゴリズムの確認


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

}
```

方針：出してからずらす
スタックがemptyのときは -1 を返す

本日の演習　ex07.c

スタックの実現

1. Push(), Pop() 関数を完成させる
2. スタックポインタ初期化用の Initialize() 関数,
   スタック表示用の Display() 関数を完成させる
   - ✓ Display() 関数の仕様

```
          4:     0
          3:     0
          2:     0
PTR->     1:   100
          0:    50
```

スタックの中身を上から順に出力
要素番号（6桁），コロン，値（6桁）
スタックポインタの位置に PTR->

- 以下の動作を確認し，出力結果を貼り付けて提出
  - ✓ データが正しく Pop，Push される
  - ✓ スタックが FULL / EMPTY の場合に、それぞれエラー表示がされる