アルゴリズム論

2017年6月5日

樋口文人

目次

- 線形リスト
  - 片方向リスト
  - 双方向リスト
- 循環リスト

線形リスト

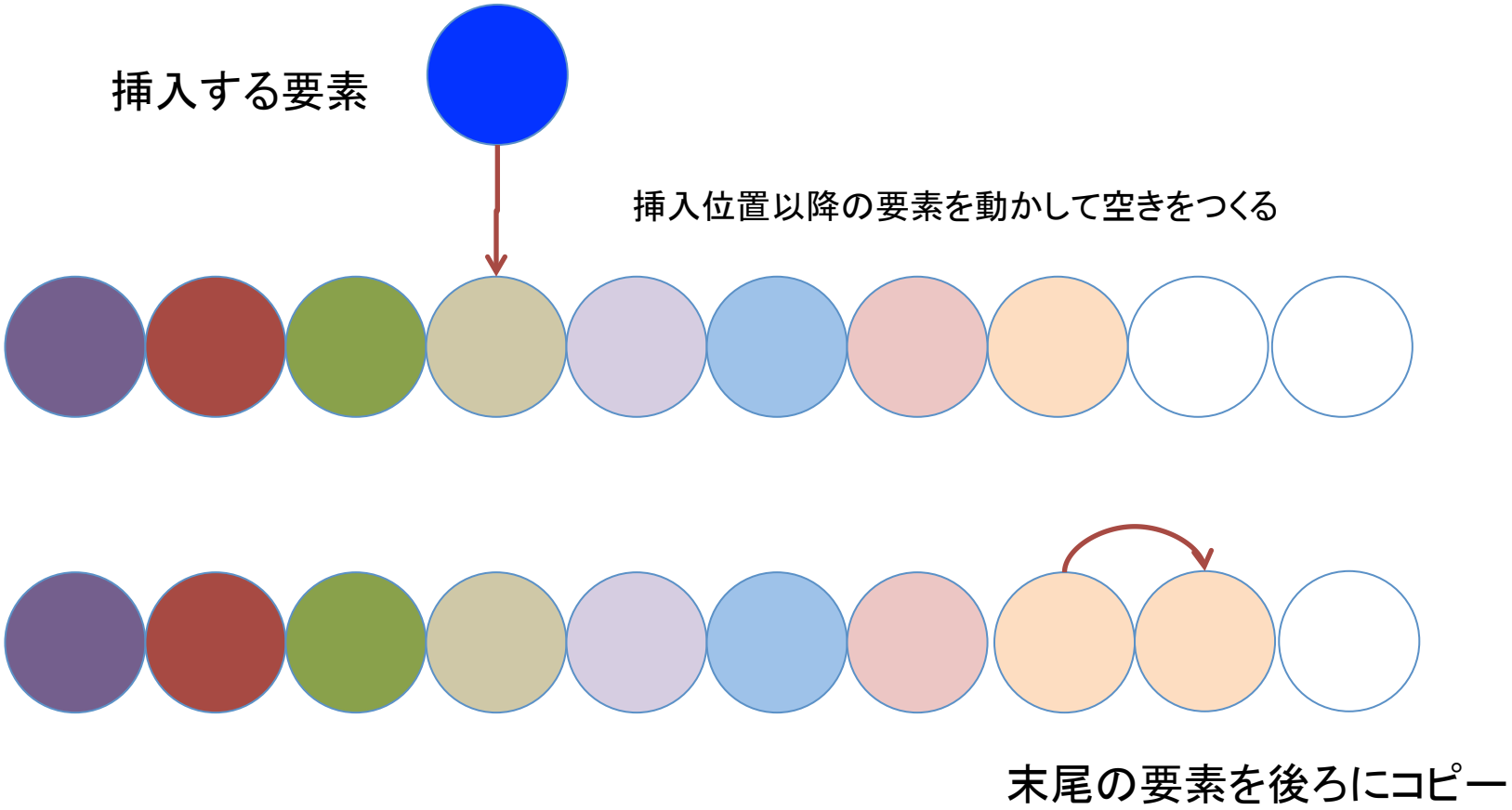
配列の長所と短所

- 長所
  - 添字を使うので，どの要素にも直接アクセス可能
  - メモリの使用効率が良い
- 短所
  - 挿入・削除に時間がかかる
  - サイズの変更ができない

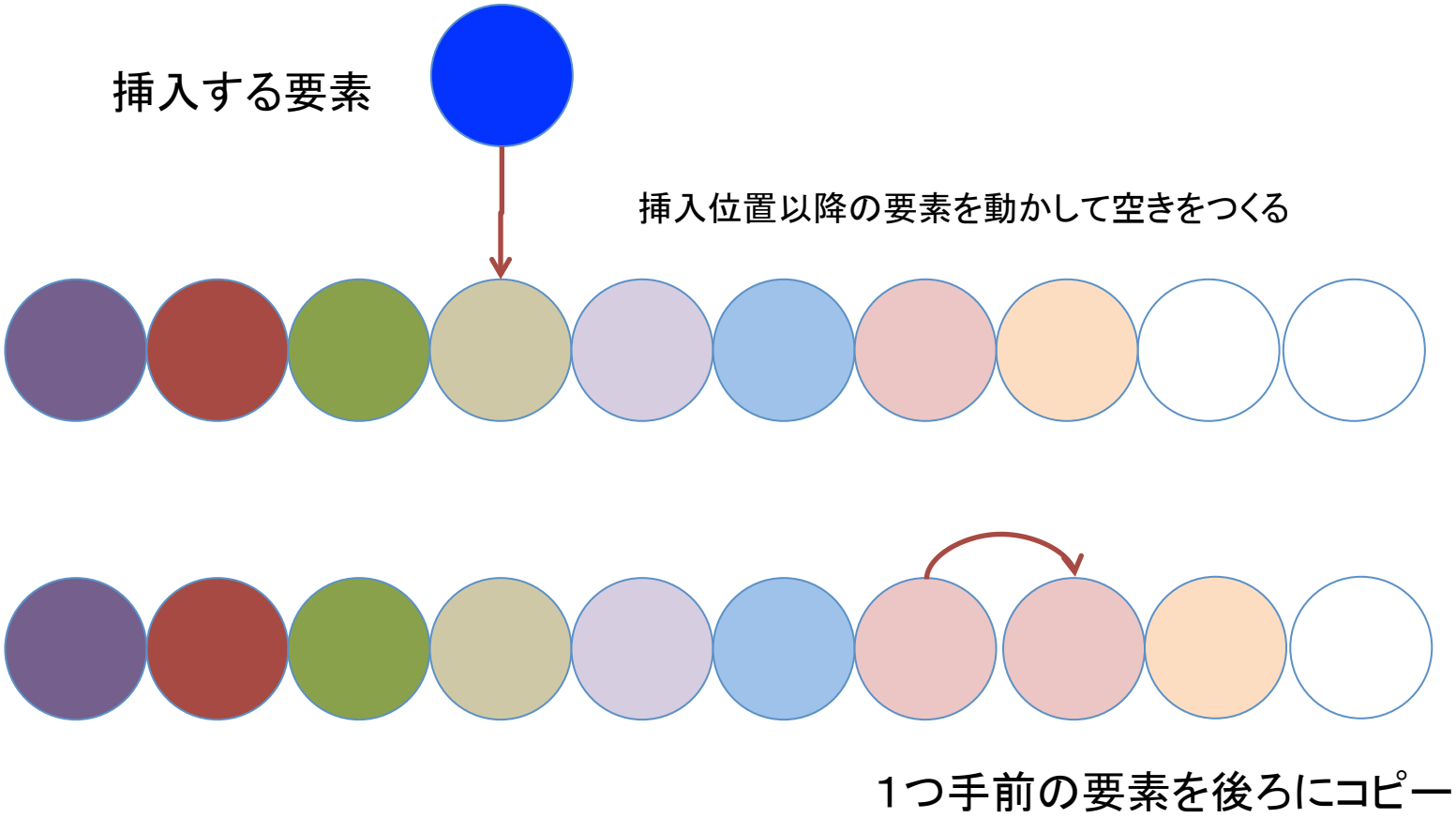
配列の問題

- データの挿入は代償が大きい
  - 挿入されたデータ以降の要素をずらす必要がある
  - データの削除についても同様
  - 手間はO(n)
- 挿入削除をしないためには
  - キーの空間と同じサイズの配列を作る必要
  - 実際にはデータ数は少なく，データの入っていない要素が大半を占める配列となる

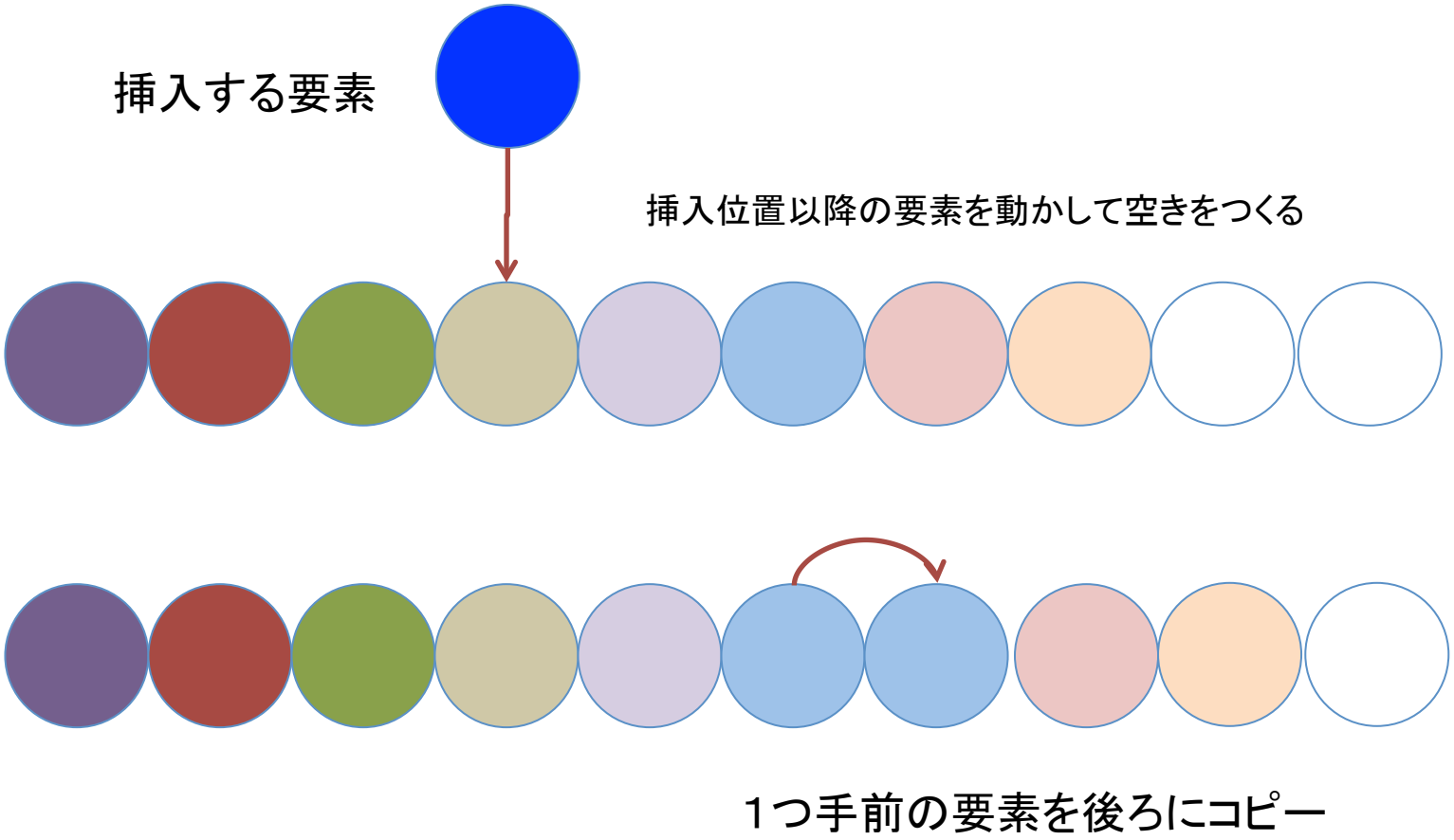
配列への挿入



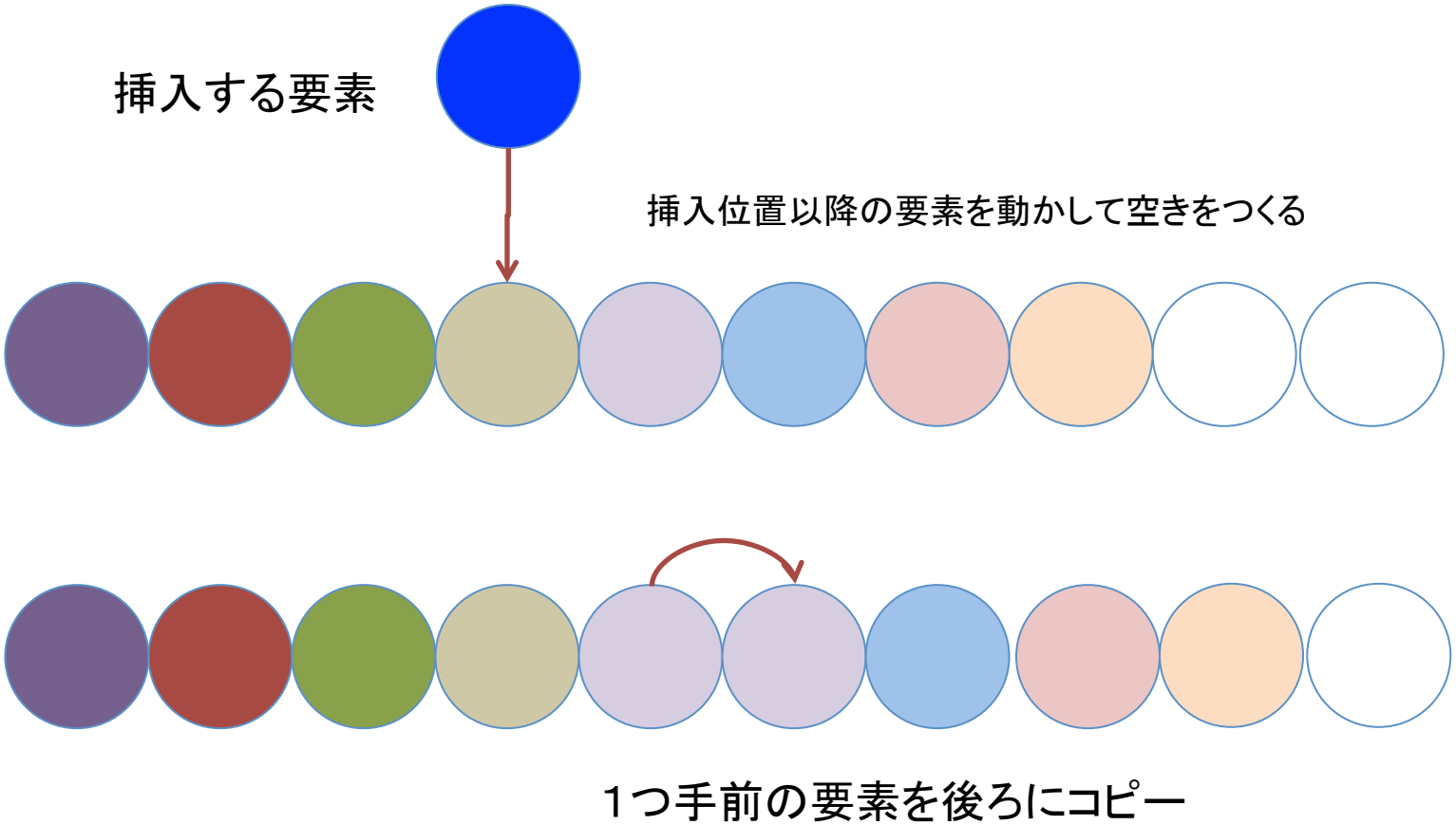
配列への挿入っづき



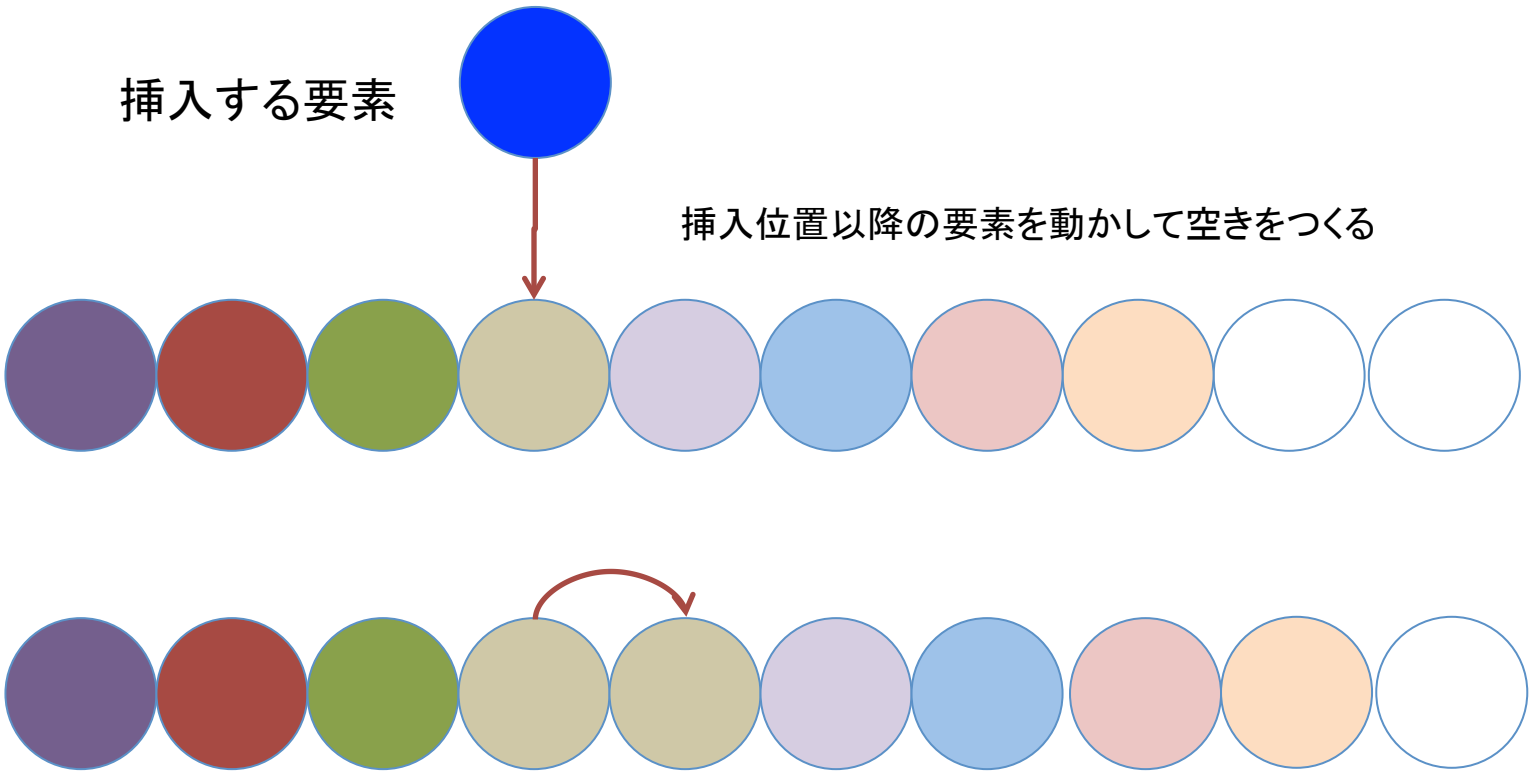
配列への挿入つづき



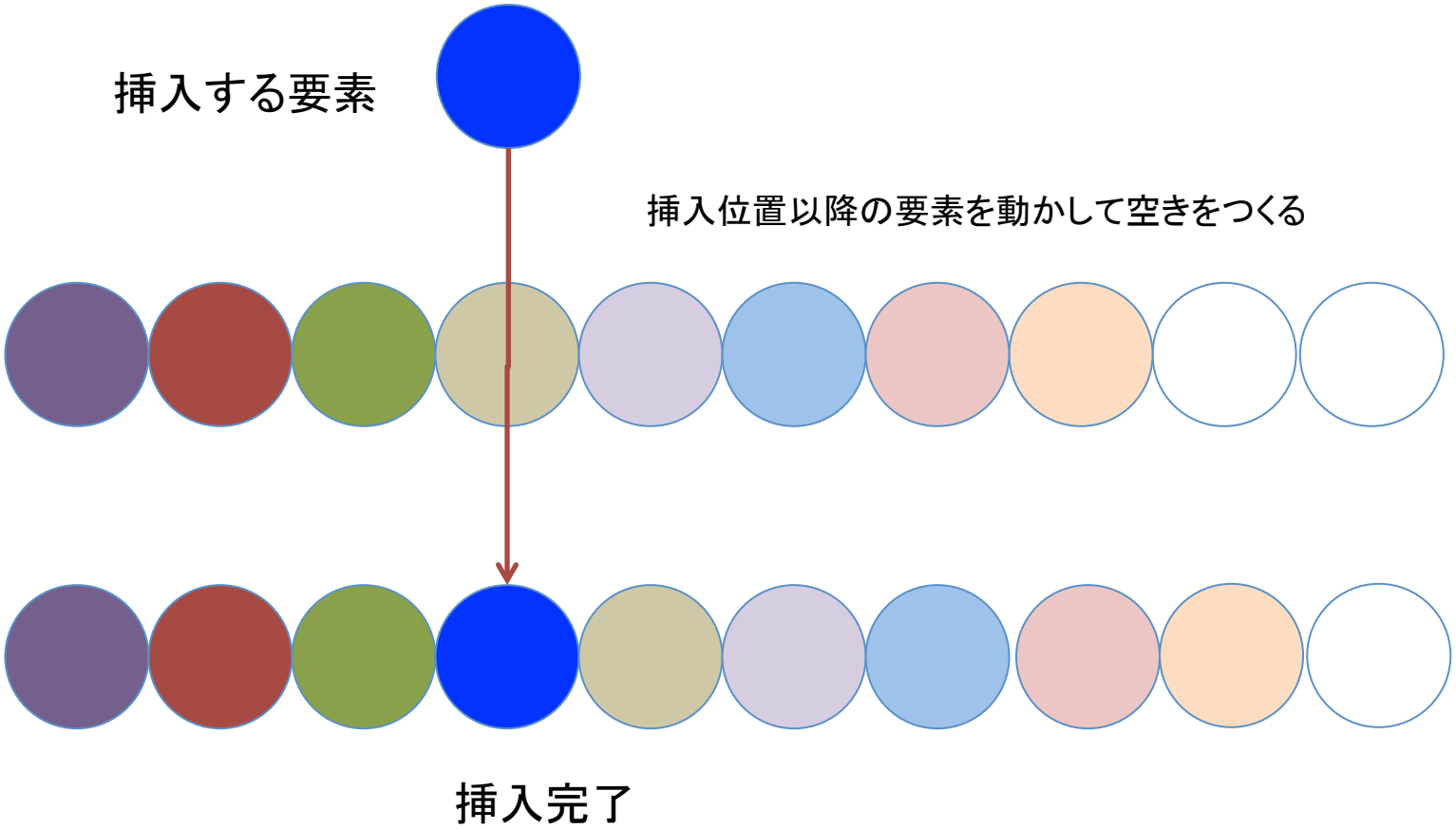
配列への挿入つづき



配列への挿入つづき



配列への挿入っづき



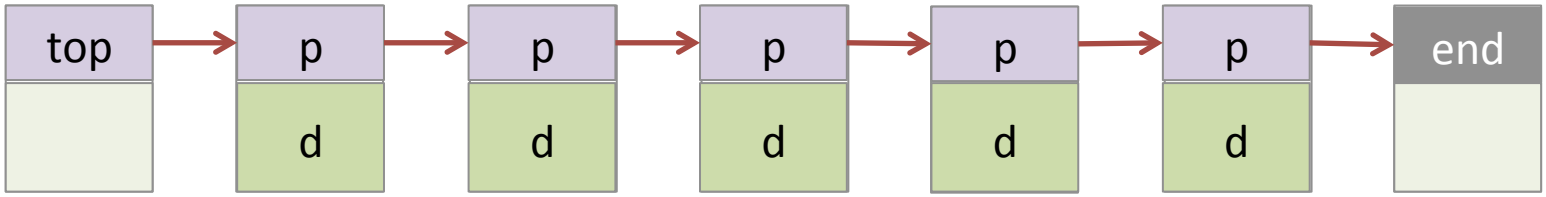
配列への要素挿入の手間

- 配列の大きさをnとする
  - 最良： 1 （末尾への追加）
  - 最悪： n （先頭への挿入） → O(n)
  - 平均： n/2 → O(n)
- 配列の要素の削除についても同様
  - 挿入の手順が逆になるだけ
    - 配列の大きさが n-1 になるので末尾に使われないデータ（のコピー）が残るが無視する

配列における挿入問題

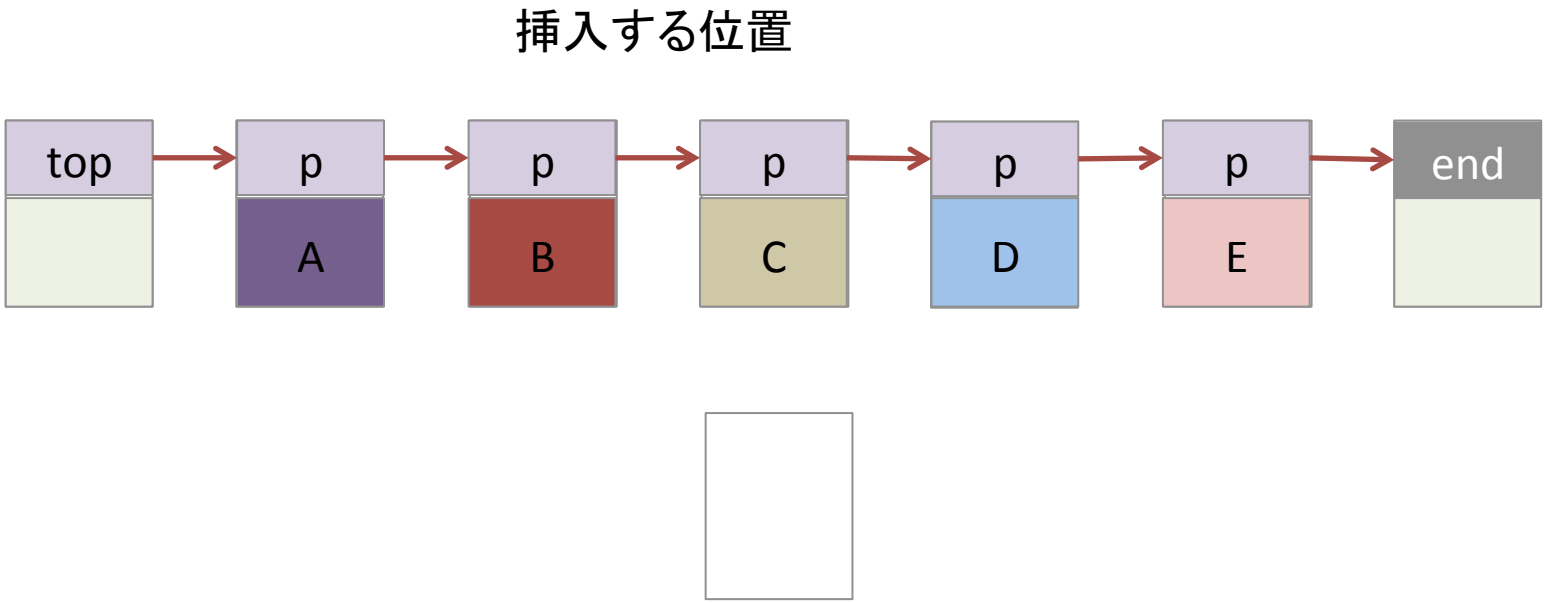
- メモリの使用効率が良くデータがコンパクトに格納されていることのトレードオフ
- メモリの使用効率を下げる
  - 配列で使用する要素の間隔を疎らにする
    - ハッシュ関数を使う
  - データだけでなく次のデータの位置も格納する
    - 線形リスト

片方向線形リスト



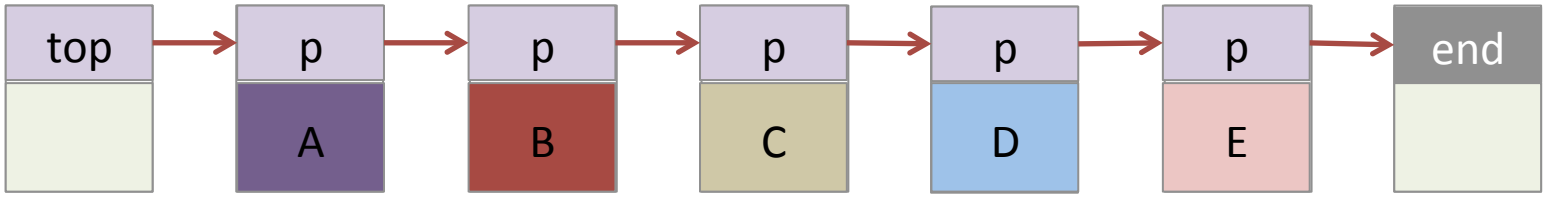
- 構造
  - ポインタ (p) : 次の要素の位置情報（具体的な値には興味はない）
  - データ (n) : その要素の保持する情報
- 長所
  - 挿入・削除の手間が位置によらない
  - 大きさを自由に変更することが可能
  - データ型を混在できる
- 短所
  - ランダムアクセスができない（常に先頭からアクセスする必要がある：シーケンシャルアクセス）
  - ポインタの分だけメモリが余計に必要（オーバーヘッド）
  - （逆向きのアクセスができない）

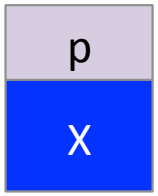
片方向線形リストへの要素の挿入



線形リストの要素のことをセル (cell) と呼ぶ

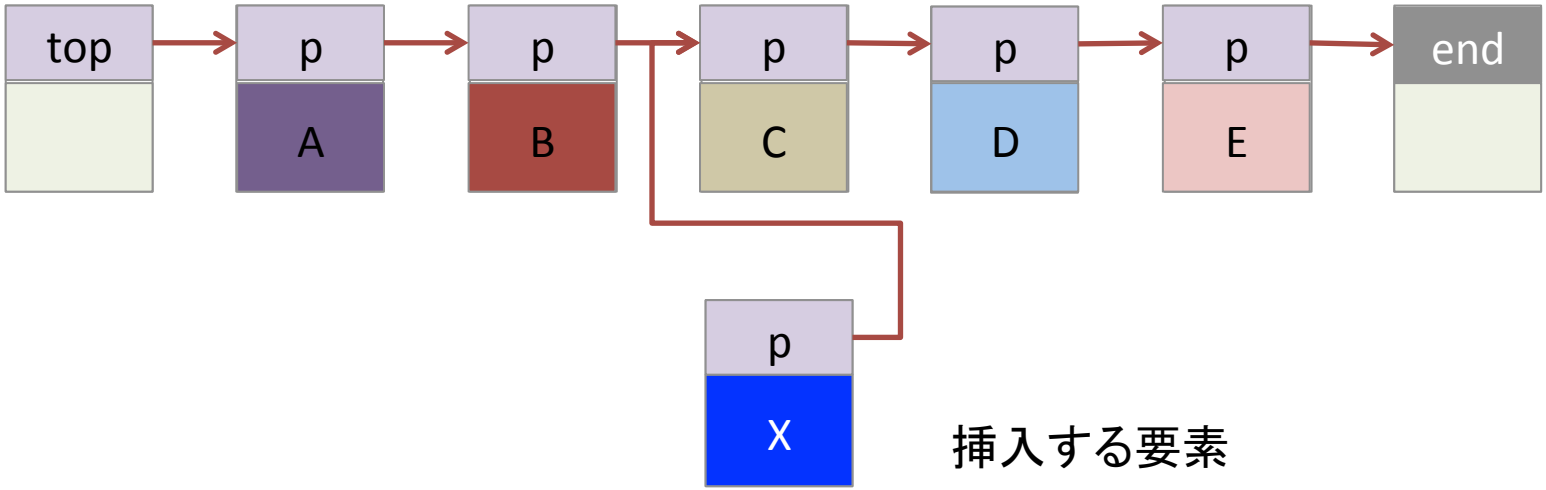
挿入する要素の生成



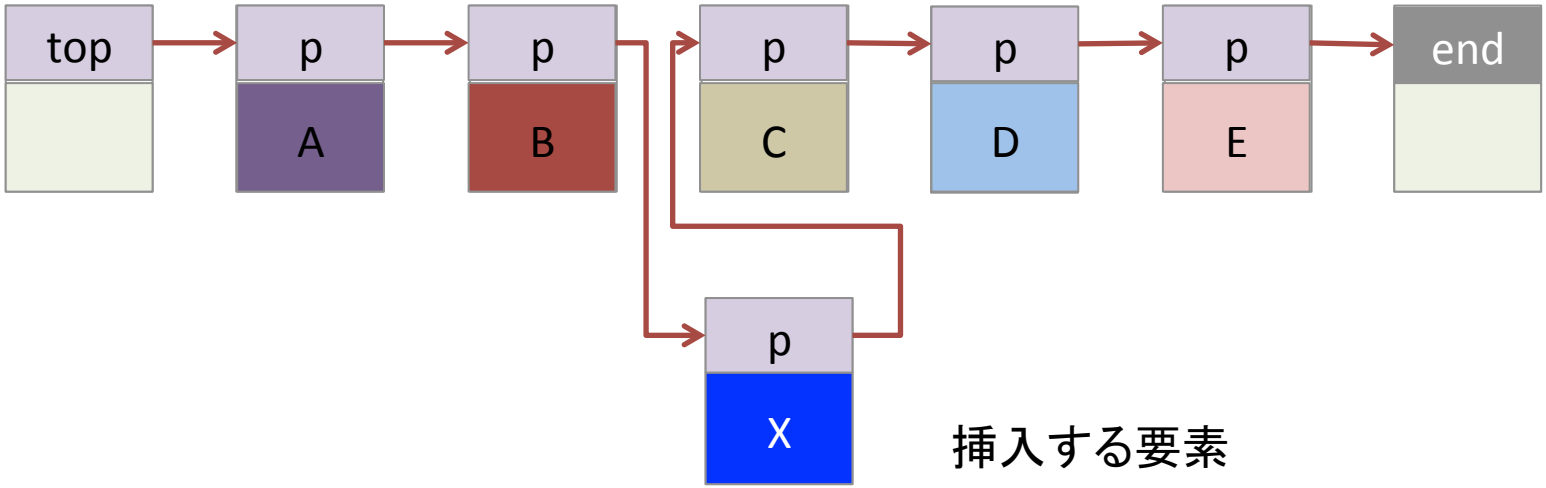


挿入する要素

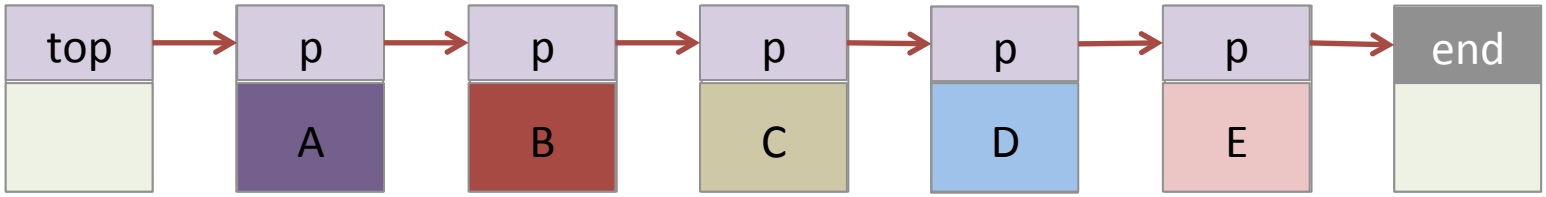
挿入要素の「次」を指定



挿入要素を「次」に指定し直す

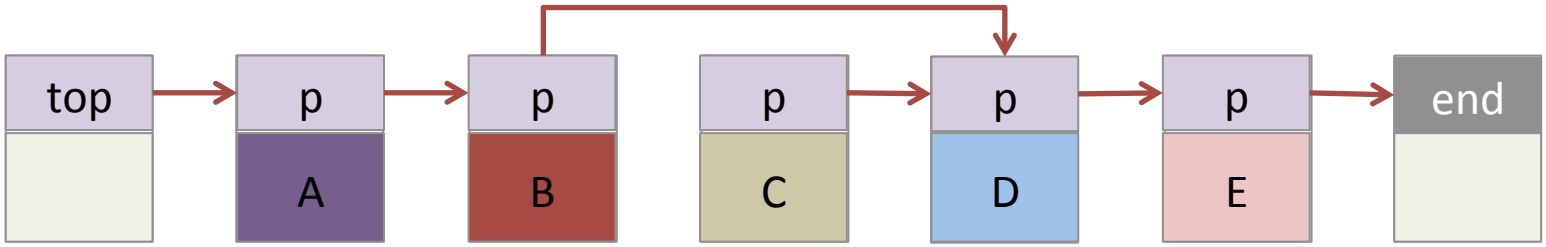


片方向線形リストからの要素の削除



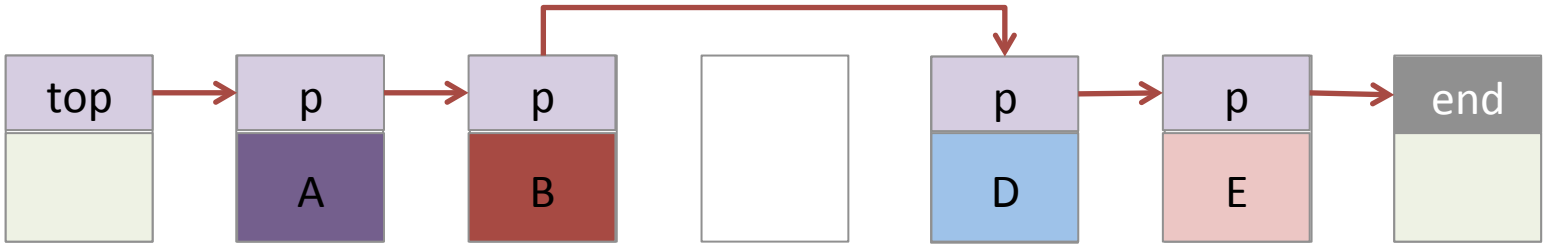
削除する要素

「前」の要素の「次」を指定し直す



削除する要素

削除する要素を取り除く



要素の消去

片方向線形リストの挿入・削除

- リストの大きさや対象要素の位置に係らず手間は一定
  - 最悪: O(1)
  - 平均: O(1)

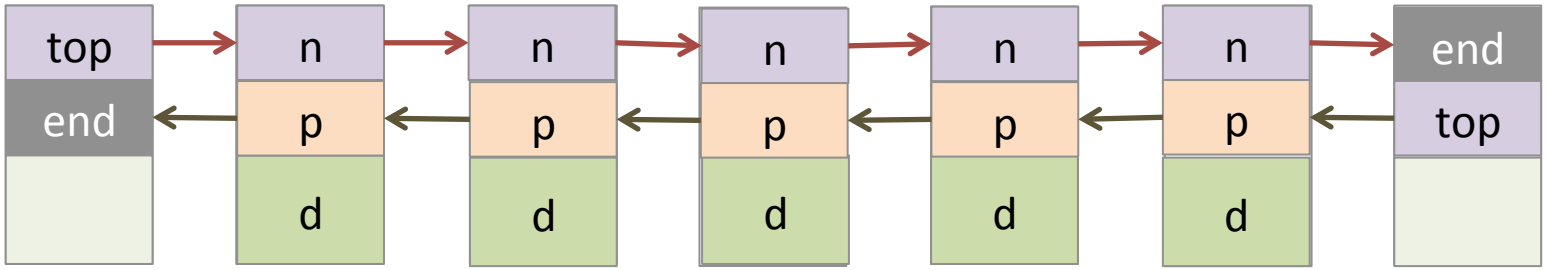
実装例:C

```
struct head {
  struct cell *first;
};

struct cell {
  struct cell *next;
  int         index;
  double      value;
}
```

実装例：python

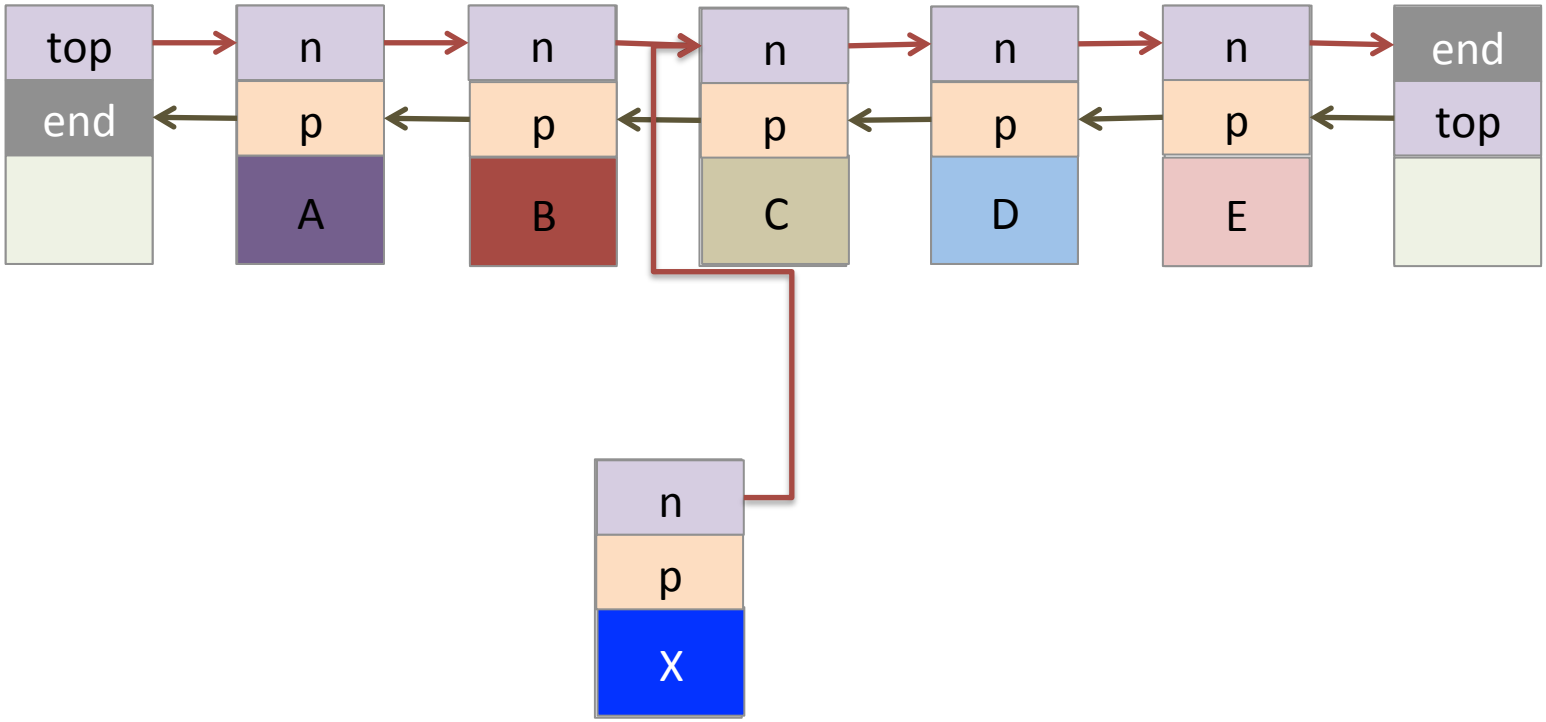
```
Class LinkedList:
  Class Cell:
      def __init__(self, data, link=None):
       self.data = data
       self.link = link

  def __init__(self, *args):
    self.top = LinkedList.Cell(None)
    for x in reversed(args):
      self.insert(0, x)

  def insert(self, n, x):
    …
```

双方向線形リスト

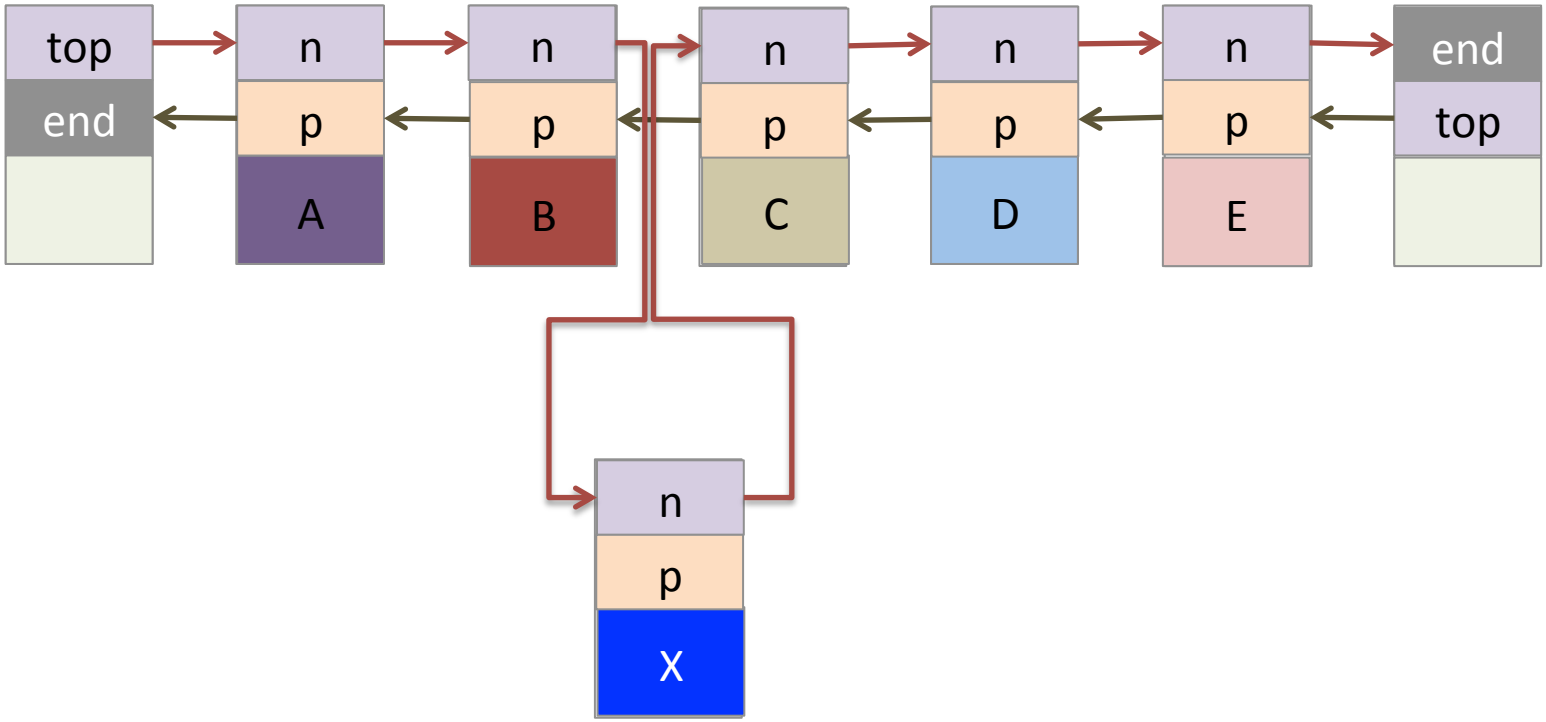


- 構造
  - ポインタ (n) : 次の要素の位置情報
  - ポインタ (p): 前の要素の位置情報
  - データ (n) : その要素の保持する情報
- 特徴
  - 逆向きのアクセスを改善

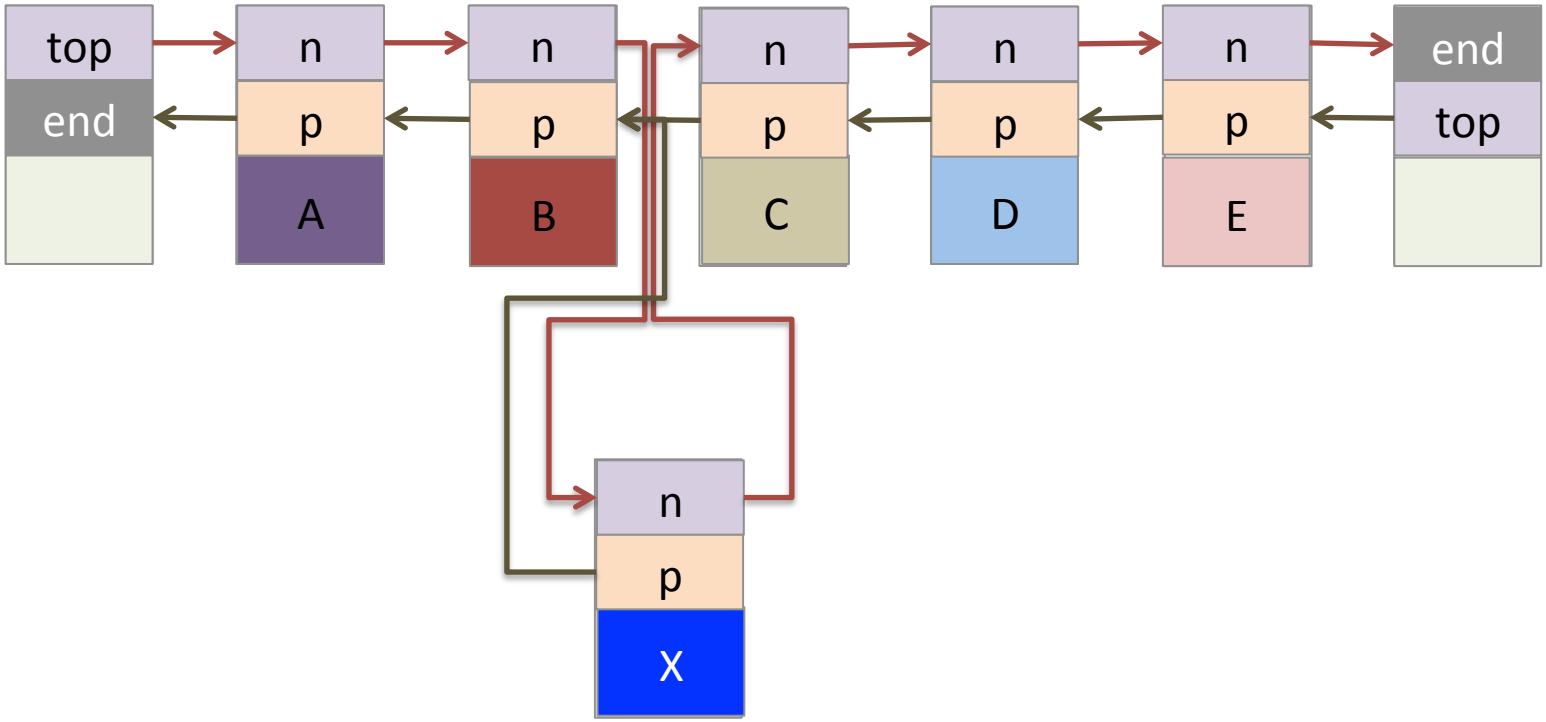
双方向線形リストの挿入



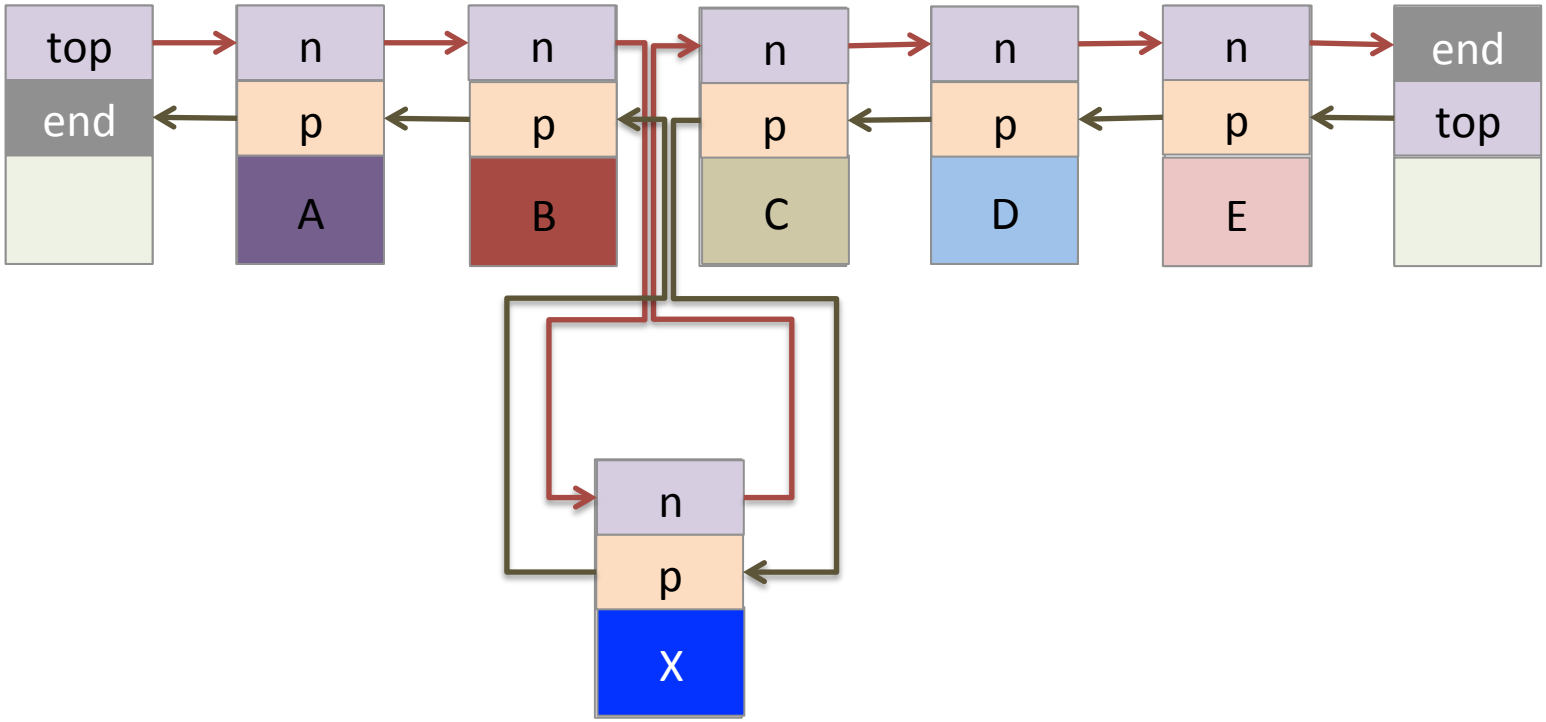
双方向線形リストの挿入



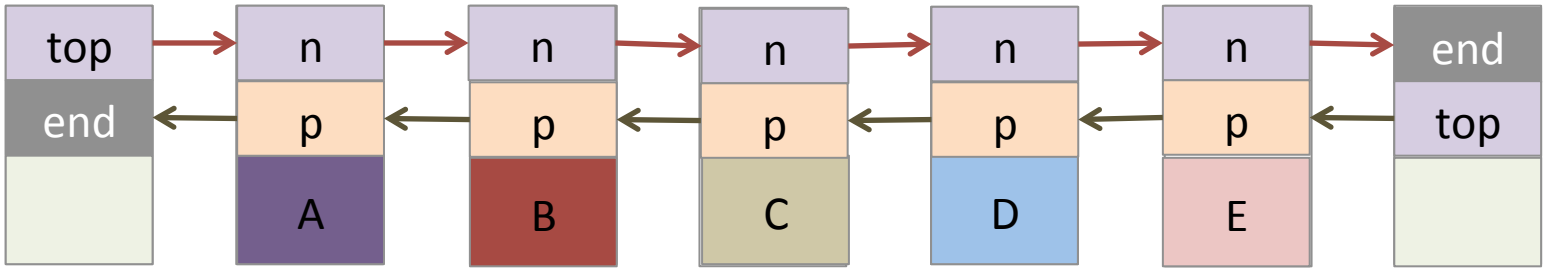
双方向線形リストの挿入



双方向線形リストの挿入

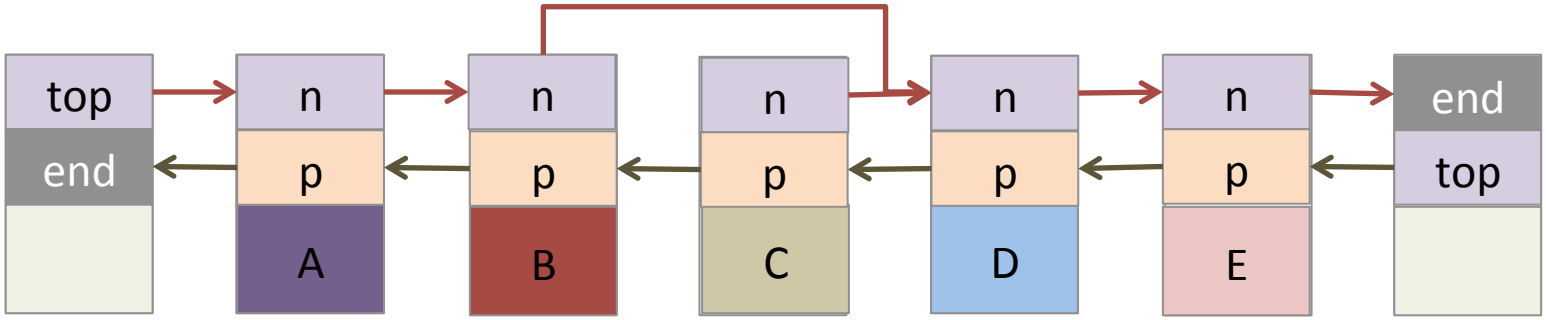


双方向線形リストの削除



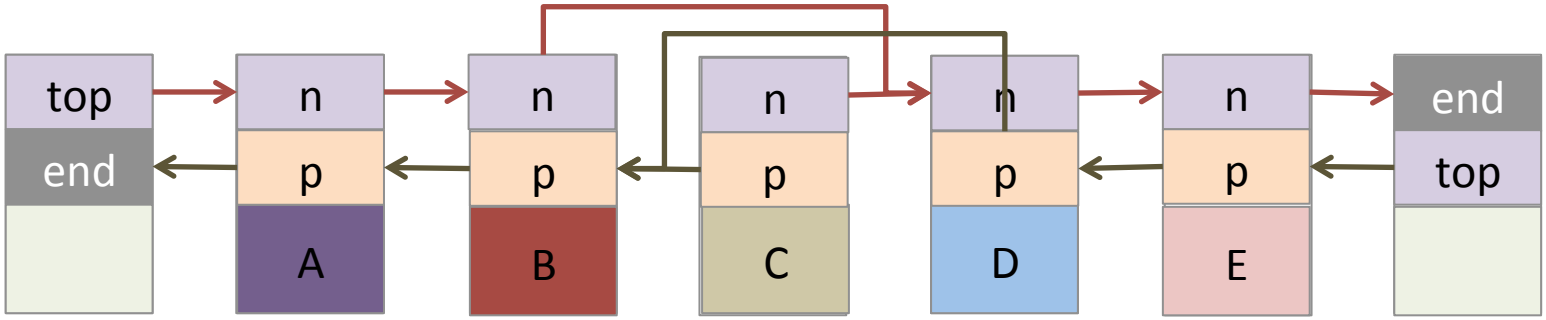
削除する要素

双方向線形リストの削除



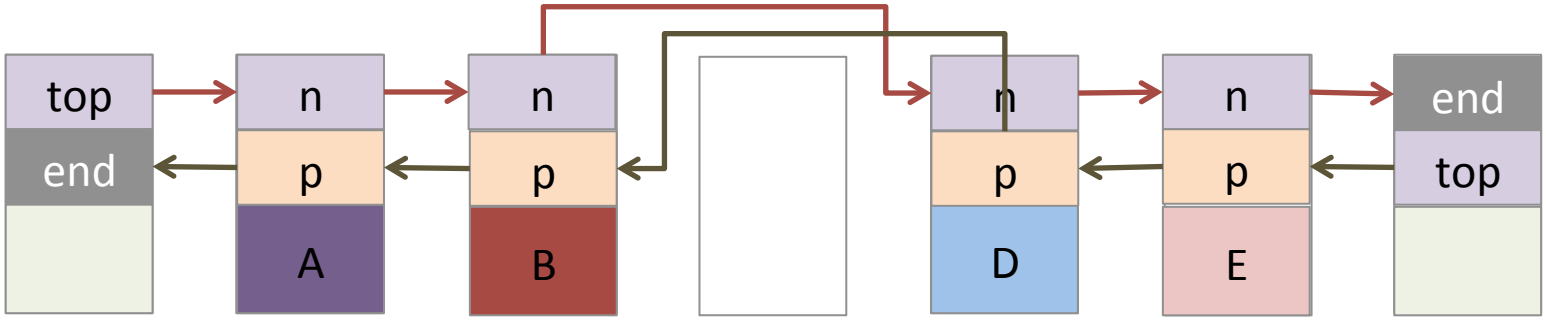
削除する要素

双方向線形リストの削除



削除する要素

双方向線形リストの削除



要素の消去

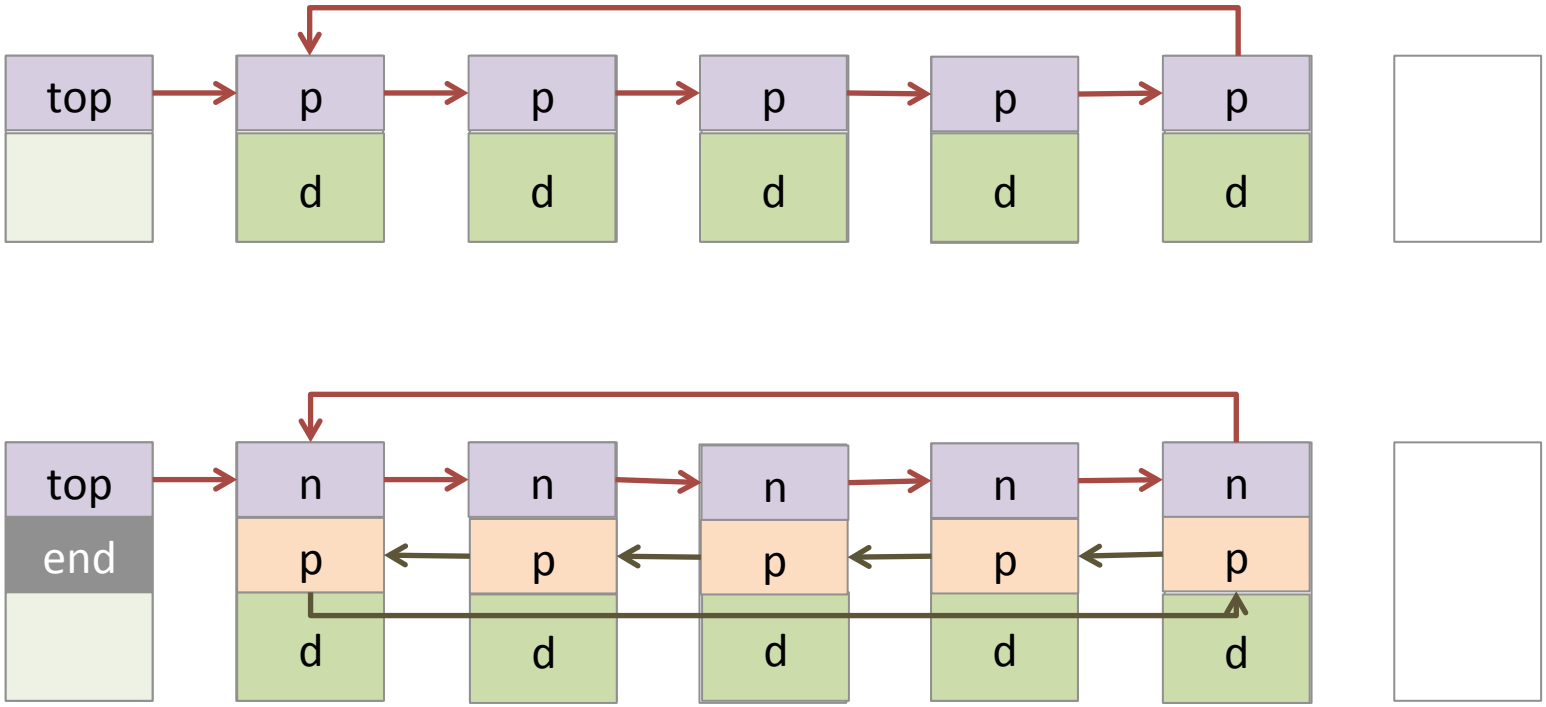
実装例:C

```
struct head {
   struct cell *first;
};

struct cell {
   struct cell *next;
   struct cell *prev;
   int         index;
   double      value;
}
```

実装例：python

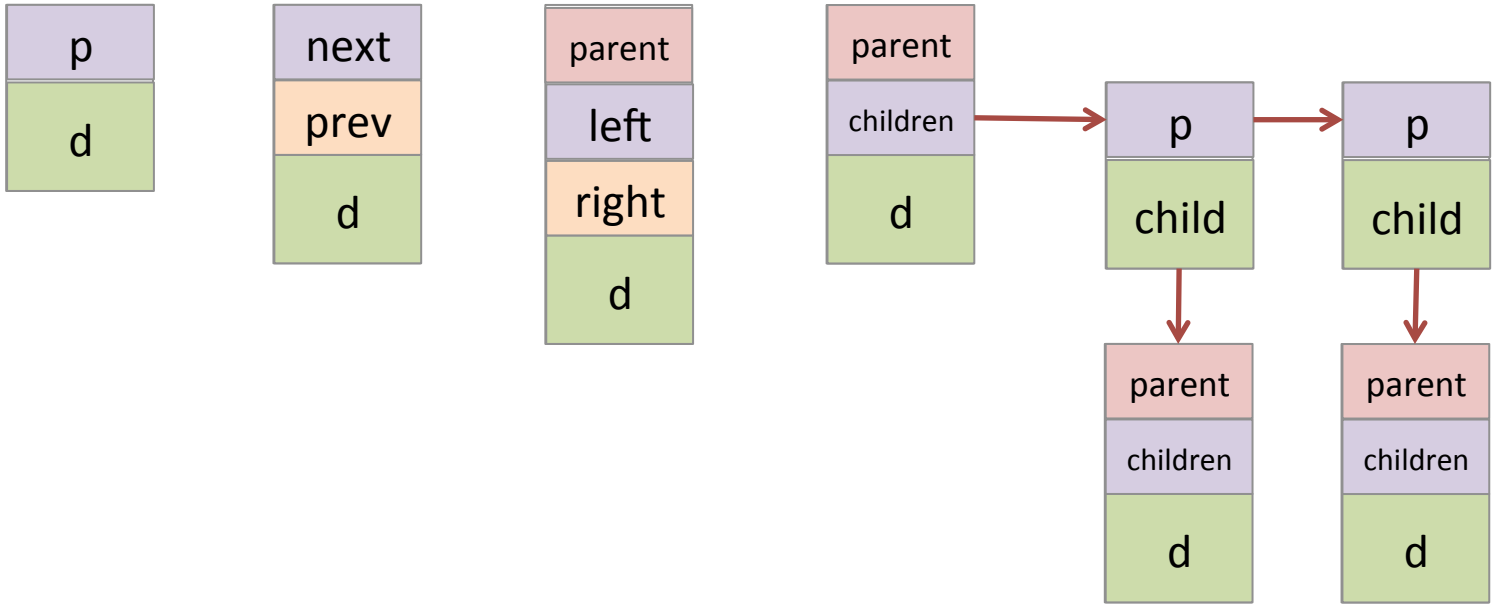
```
DoubleLList:
  Class Cell:
    def __init__(self, data, next=None, prev=None):
    self.data = data
    self.next = next
    self.prev = prev

  def __init__(self, *args):
    self.top = DoubleLList.Cell(None)
    self.size = 0
```

循環リスト



リストの応用と発展

- スタックとキュー
- セルの拡張



ポインタ

- 変数のアドレスを表す変数
  - 通常の変数

    int x = 0; // x という名前の変数に値 0 を設定
    int *a; // a は整数型の変数へのポインタ
    a = &x; // &x は変数 x のアドレスを表す
  - 例

```
int x, y, z = 1;
int *a, *b, *c;
a = &x; b = &y; c = &z;
*a = *c + *c;
*b = *a + *c;
printf(“%i, %i, %i”, x, y, z);
```

malloc()

- プログラムからメモリ上の領域を確保してその先頭アドレスを返す関数
- 使い方（int型の領域を確保する場合）
  - #include <stdlib.h> が必要
  - (int *)malloc(sizeof(int))

```
int *new;
new = (int *)malloc(sizeof(int));
```

- 不要になったメモリ領域はfree()関数で解放する
  - free(解放するメモリ領域へのポインタ)

```
free(new);
```

やってみよう： 作業

- 片方向線形リストで cell の挿入や削除を行う関数またはメソッドはどのようにプログラムできるでしょうか？
- また、そのプログラムのコードでは，最初のデータの前や最後のデータの後にcellを挿入したり，これらのデータを削除する場合，特別な配慮が必要か？

やってみよう：作業２

- 双方向線形リストで cell の挿入や削除を行う関数またはメソッドはどのようにプログラムできるでしょうか？
- 何番目のデータかを調べるには？
- 前後のデータを交換するには？
  - それを使ってバブルソートを実装できるか？

課題: ex07

- C/C++のポインタ変数を利用して片方向線形リストを作成してください。
  - セル数10個
  - データはランダムな整数値
  - 片方向線形リストの内容を順に表示して終了する
- ねらい
  - ポインタ変数の利用
    - ＊や＆の使用法を理解する
  - malloc()関数の利用

やる気のある人へ

- セル操作に使う関数を作成する
  - 先頭に追加
  - 末尾に追加
  - 任意の場所に挿入
  - リスト内のセルの個数を数える
- さらに:
  - セルの削除
  - 双方向線形リストで同様にプログラムしてみる

提出についての注意

- プログラム名
  - デフォルトでは sketch_yymmdda などだが…
  - higuchi_fumito_ex07 のように氏名と宿題番号に変えること
  - higuchi_fumito_c5_ex07 (同姓同名はクラスを付加）
- プログラムの冒頭に氏名、学年、クラス、番号等をコメントとして記入
  - 参考にした資料の他、簡単な感想も付け加えてください
- Oh-o!Meijiから提出（次回の授業開始までに）

連絡先

樋口文人

wenren@meiji.ac.jp