C(2)プログラム
線形リスト配列への登録と順位付け処理

変数 bucket の型

```
typedef struct Bucket { // Bucket は Bucket 構造体の別名
    struct Bucket *next;  // 次のBucketを指すポインタ
    int bangou;  // データの番号 (0始まり)
} Bucket;
Bucket *bucket[MAXG + 1];
```

bucket は、配列　← Bucket *　の

つまり

「Bucketを指すポインタ」の配列 (サイズ 301)

配列要素1個1個が、それぞれ線形リストであることに注意

初期設定

```
for ( int b = 0; b <= MAXG; ++b )
    bucket[b] = NULL;
```

配列の全要素を **NULL** にすることにより全リストを空に設定する

bucket

[0]	NULL
[1]	NULL
[2]	NULL
[3]	NULL
	⋮
[299]	NULL
[300]	NULL

始めての点数のデータ追加

i1 番の人の合計点が g で、初の点数の場合

bucket

[0]	NULL
[1]	a Bucketのアドレス
[2]	NULL
[3]	NULL
⋮	
[g]	NULL
⋮	pnに記憶
[299]	b Bucketのアドレス
[300]	NULL

```
for ( int i = 0; i < N; ++i ) {
    int g = getGokei( &data[i] );
    Bucket *pn = bucket[g];
    Bucket *p = bucket[g] =
malloc(sizeof(Bucket));
    if ( p == NULL ) {
        fputs("Out of memory¥n", stderr);
        exit(1);
    }
    p->bangou = i;
    p->next = pn;
}
```

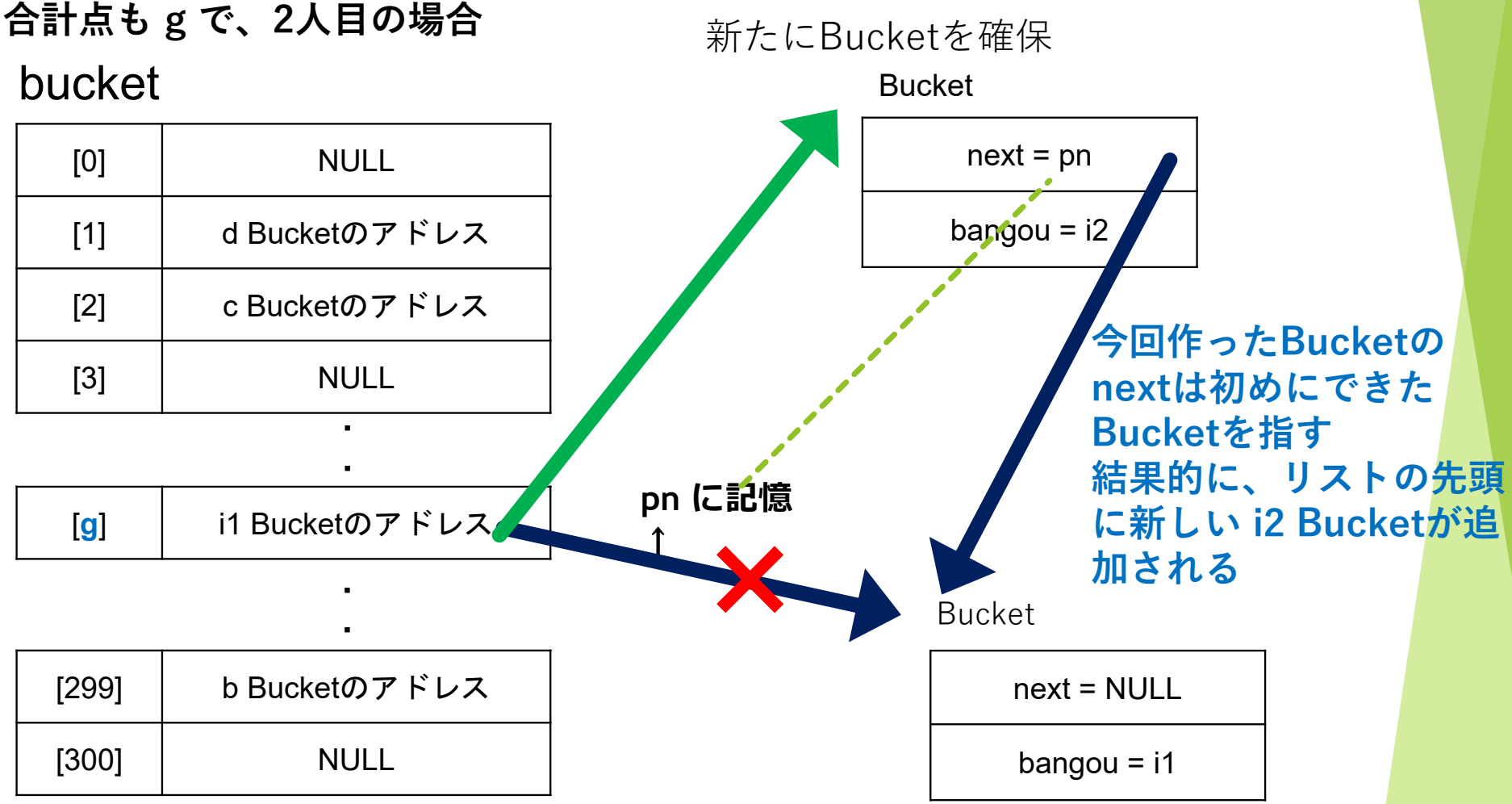
bucket[g] は新しく確保したBucketを指すようになる

Bucket

next = NULL (pn)
bangou = i1

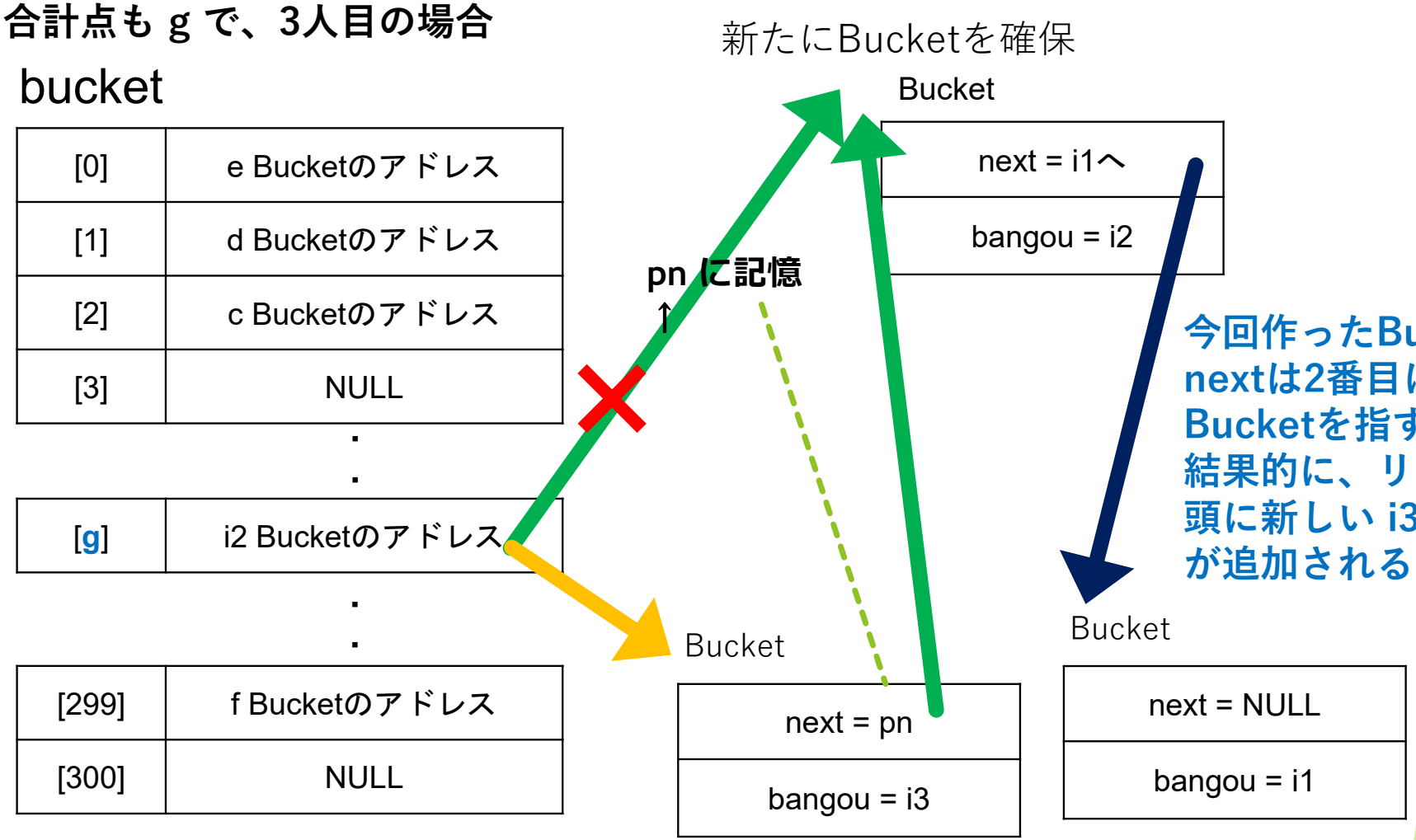
同じ点数の2人目のデータ追加

i2 番の人の合計点も g で、2人目の場合



同じ点数の3人目のデータ追加

i3 番の人の合計点も g で、3人目の場合



片方向リスト (単方向リスト)

- 今回リストで使ったデータ構造は**線形リスト**。中でも最も単純な**片方向リスト (単方向リスト)**
- リスト末尾を記憶しない場合、末尾への追加はリストを先頭から順に辿り、末尾に達し、そこで追加を行なう必要。コストが大きい O(n)
- リスト先頭への追加はポインタを切り替えれば即座に実行でき、コストが小さい O(1)
- **C++の std::forward_list は片方向リスト**なので、**先頭への追加 push_front メソッド**はあるが、末尾への追加 push_back メソッドはない
  (参考) std::list は双方向リストで、push_front も push_back もある

bucketを使った順位付け

```
int j = 1;
for ( int b = MAXG; b >= 0; --b ) {
    Bucket *p = bucket[b];
    if ( p != NULL ) {
        int ii = 0;
        for ( ; p != NULL; ++ii, p = p->next ) {
            jun[p->bangou] = j;
        }
        j += ii;
    }
}
```

変数 j は現在調べている順位を表す
上位から調べるので初期値は 1

最高点(300)から最低点(0)の方向で、順に配列 bucket の要素の値を見ていく。ここで、変数 b は合計点を表している

配列要素がNULLでない、つまり、リストが空でないなら、ii を初期値 0 のカウンタとして、順位配列の該当番号 p->bangouの要素に現在の順位を設定し、ii に1加える。
p にリストの次のポインタを代入し、一つ先を見る。
p が NULL でない間繰り返す

内側のfor文を抜けたときには、ii は合計点 b 点の同点の人数を持つ。
次の順位を調べるため、ii を j に加える

jun は全員の順位を持つ配列