プロ演IV 14回目

2024年７月11日

前橋工科大学　生命情報学科

中村 建介

アルゴリズム

数値計算

サーチ、ソート

ツリー構造・グラフ

再帰

City	Population	Area(km^2)
Maebashi	342678	311.64
Takasaki	371127	459.51
Saitama	1232577	217.49
Yokohama	3688624	437.38
Nara	364738	276.84
Ikoma	119258	53.18
.....	.....	.....

沢山の情報（レコード）を持つファイルのデータを処理する場合

1. ひとつひとつのデータを読んで処理（処理後は廃棄）
2. 大きめに配列を切り、読み込み　(MAX_HITS=1000)
3. **一旦総数をカウントしmallocしてrewindしてから格納**
4. **レコードを読み込むごとにmallocして格納**

→ リスト形式

リスト構造

4. レコードを読み込むごとにmallocして格納

```
struct city_record
 {
 char name[20];                    // 名前
 int  population;                  // 人口
 int  area;                        // 面積
 struct city_record *next;         // 次のレコードへの ポインタ部
 }
```

データ部

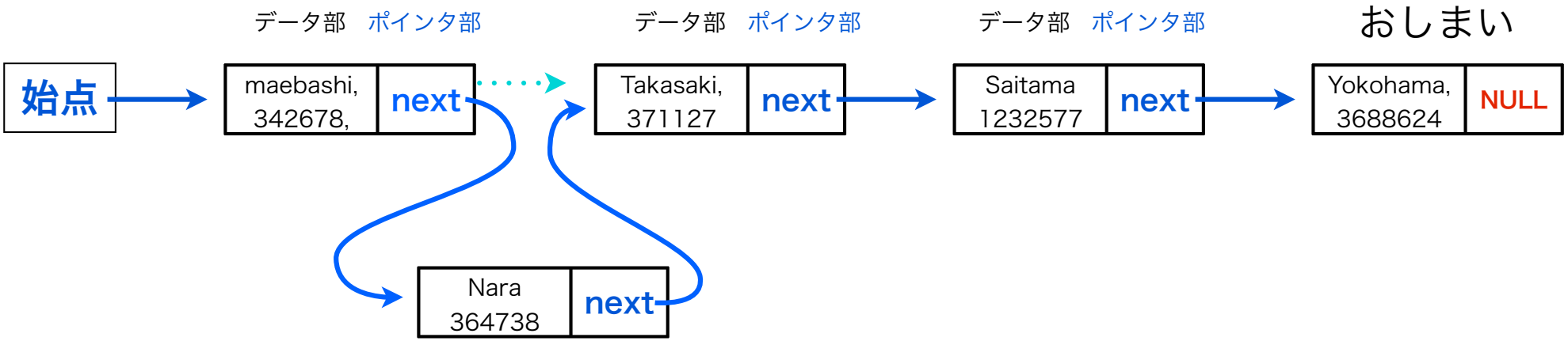

```
MAEBASHI      342678
TAKASAKI      371127
SAITAMA      1232577
YOKOHAMA     3688624
```

リスト構造

4. レコードを読み込むごとにmallocして格納

```
struct city_record
 {
 char name[20];                    // 名前
 int  population;                  // 人口
 int  area;                        // 面積
 struct city_record *next;         // 次のレコードへの ポインタ部
 }
```

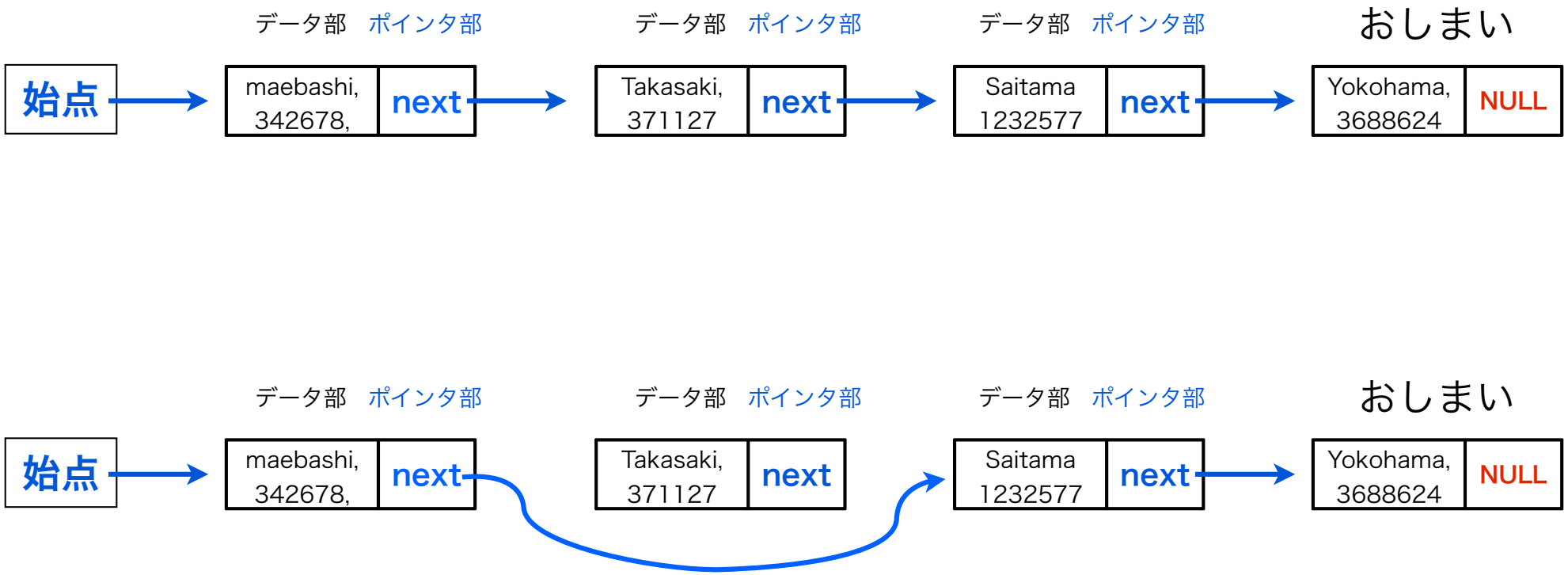
データ部



ポインタをつなぎ変えるだけでレコードを簡単に挿入できる

リスト構造

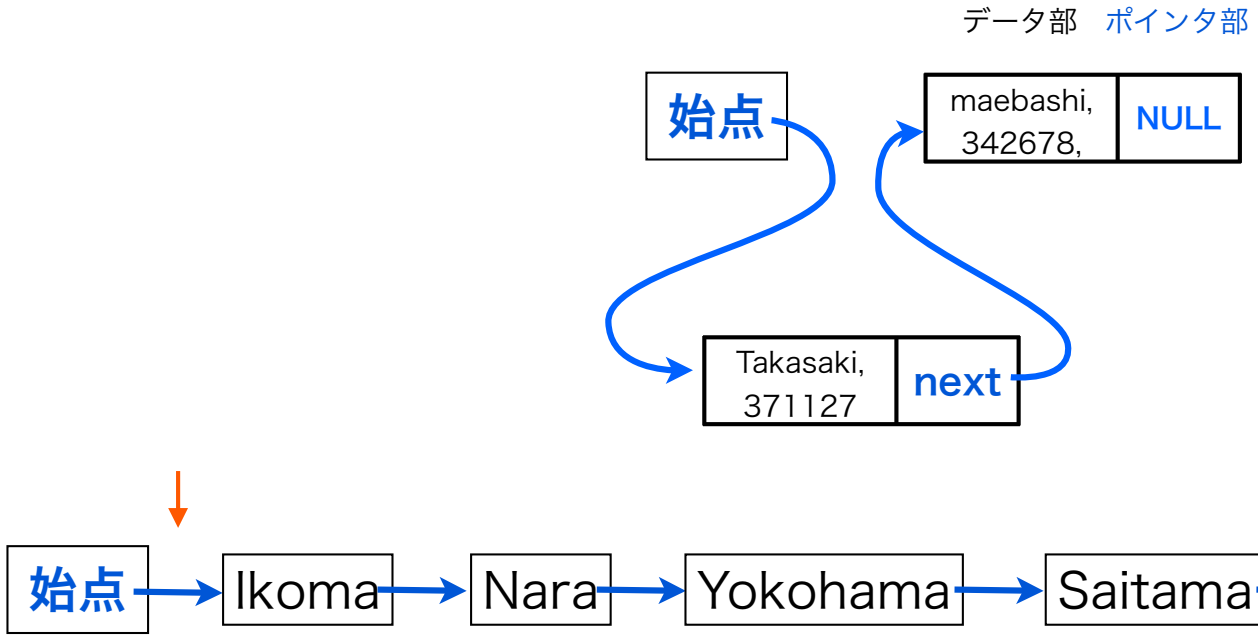
4. レコードを読み込むごとにmallocして格納



ポインタをつなぎ変えるだけでレコードを簡単に削除できる

リスト構造

4. レコードを読み込むごとにmallocして格納



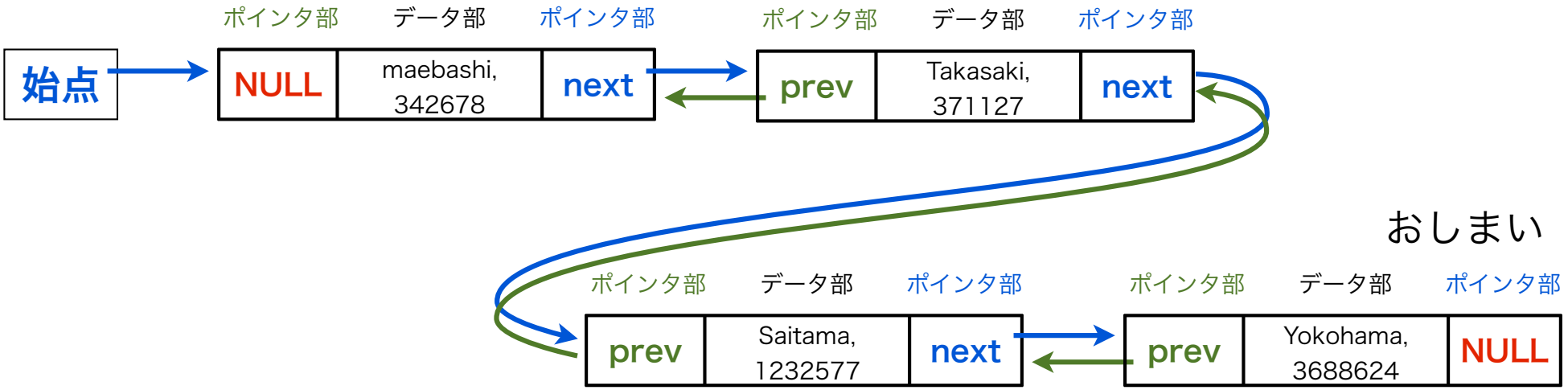
city_pop.dat

```
MAEBASHI 342678
TAKASAKI 371127
SAITAMA 1232577
YOKOHAMA 3688624
NARA 364738
IKOMA 119258
```

演習１では新しい(後から来た）レコードを先頭に挿入し、
最初に入れたレコードがリストの最後に来る繋ぎ方をする

双方向リスト

```
struct city_record
 {
 char name[20];                  // 名前
 int  population;                // 人口   } データ部
 int  area;                      // 面積
 struct city_record *next;       // 次のレコードへの ポインタ部
 struct city_record *prev;       // 前のレコードへの ポインタ部
 }
```

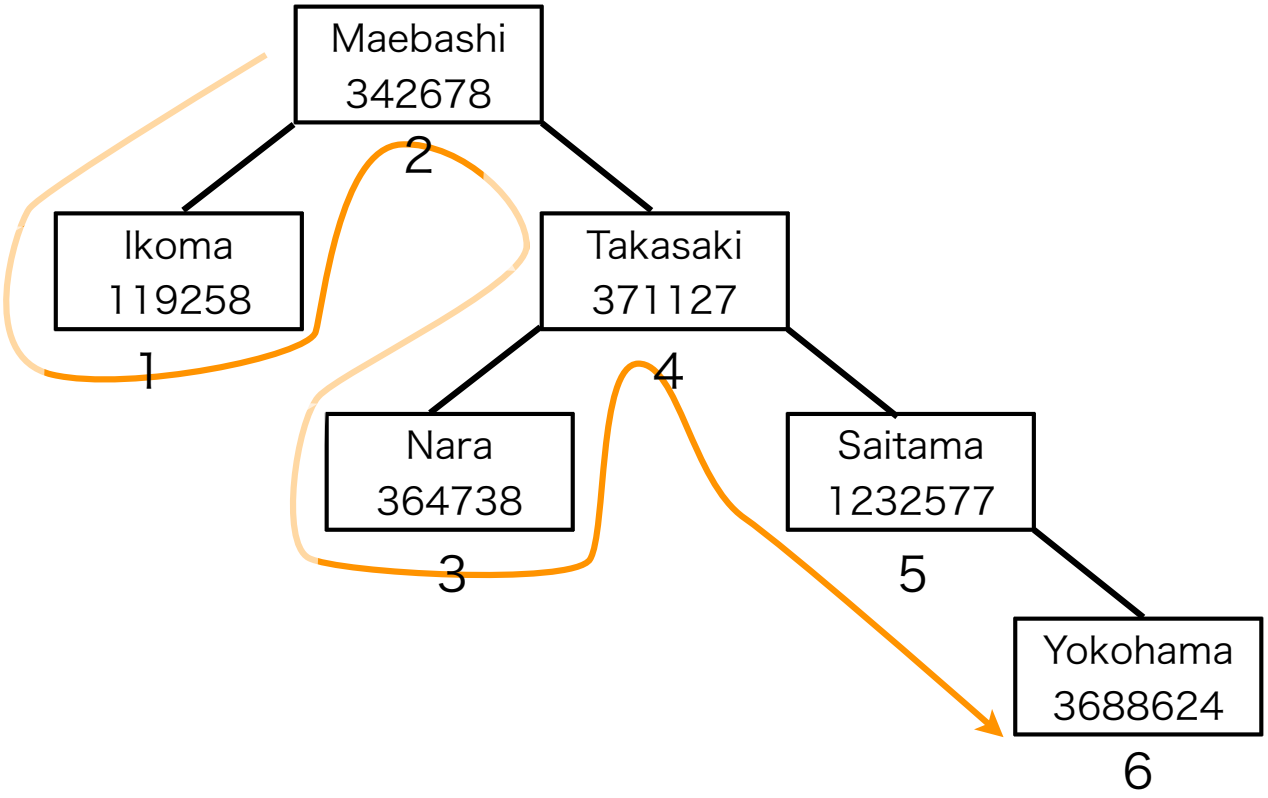


始点から終点まで辿るだけでなく、戻ることも出来る

二分木

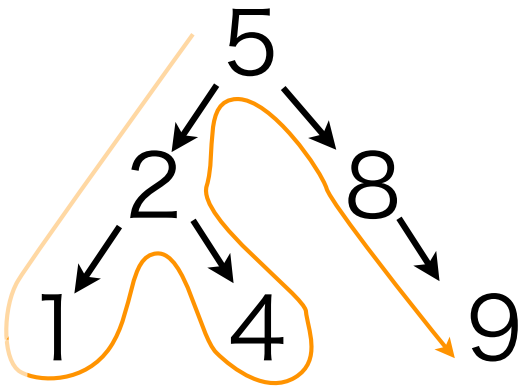
City	Population
Maebashi	342678
Takasaki	371127
Saitama	1232577
Yokohama	3688624
Nara	364738
Ikoma	119258
.....	.....

```
struct city_record
 {
 char name[20];                  // 名前
 int population;
 struct city_record *right;      // 右のレコードへの ポインタ部
 struct city_record *left;       // 左のレコードへの ポインタ部
 }
```



深さ有先探索

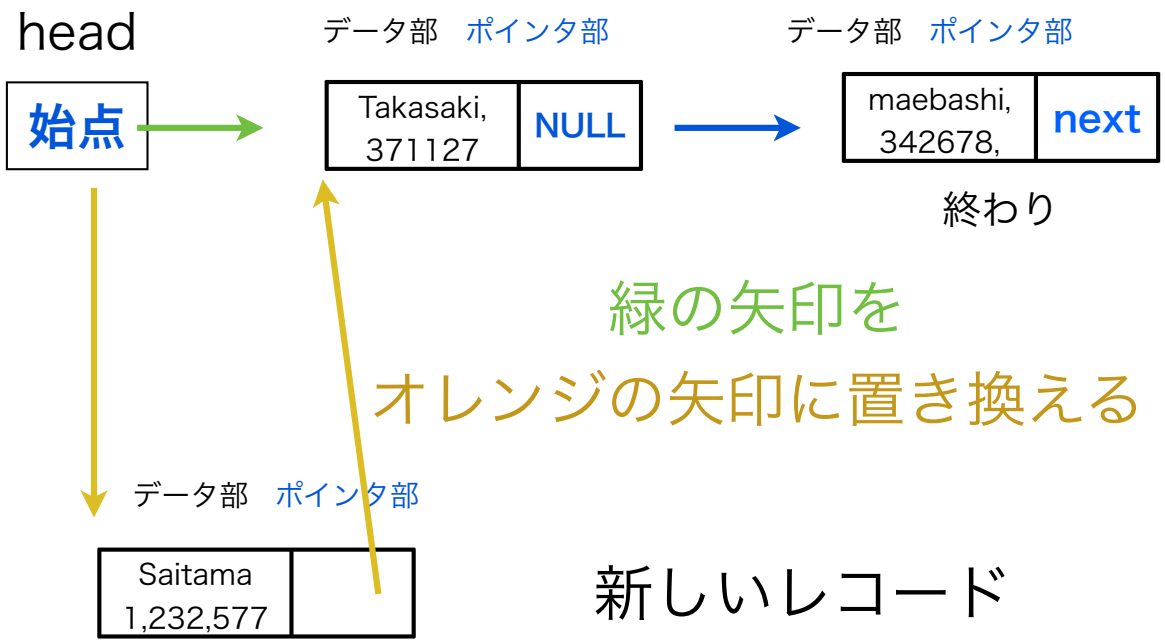
5
2
1
4
8
9



左の底に付いたら拾い始める

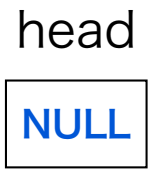
順番にレコードを加えて深さ優先探索するだけでソートされる

逆順のつなぎ方 課題13-1



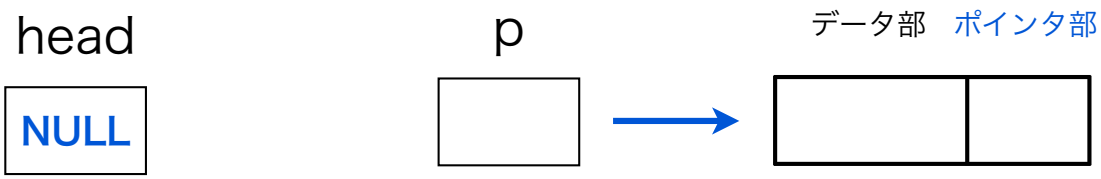
最初に挿入

逆順のつなぎ方



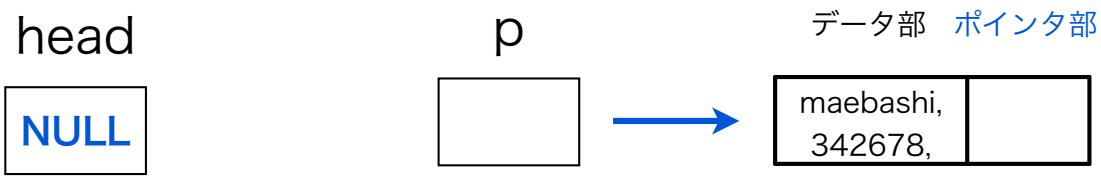

```
city_record *head = NULL;
```

逆順のつなぎ方



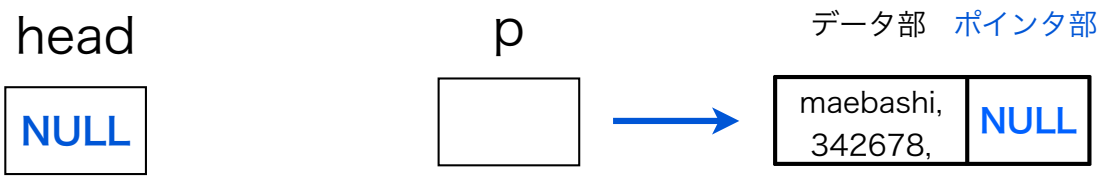

```
city_record *head = NULL;
p = new_record();
```

逆順のつなぎ方




```
city_record *head = NULL;
p = new_record();
fscanf(“〜〜”,p->name,&p->pop);
```

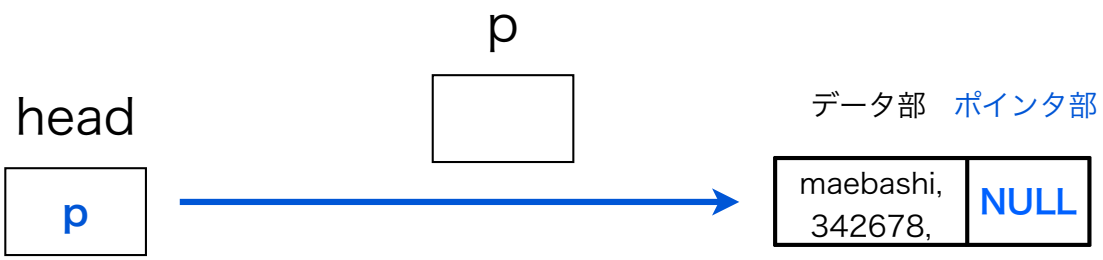
逆順のつなぎ方




```
city_record *head = NULL;
p = new_record();
fscanf(“〜〜”,p->name,&p_population);

p->next = head;
```

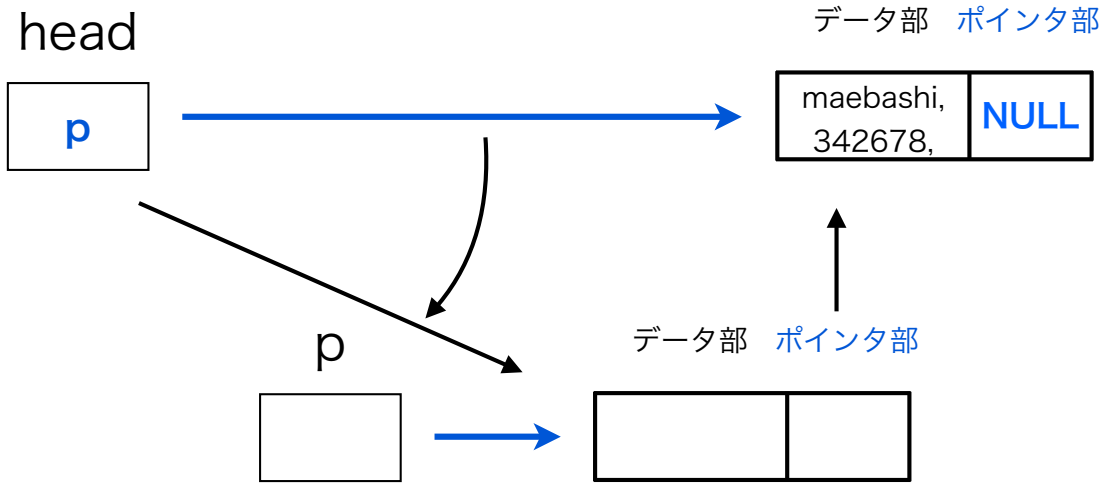
逆順のつなぎ方




```
city_record *head = NULL;
p = new_record();
fscanf(“〜〜”,p->name,&p_population);

p->next = head;
head = p;
```

逆順のつなぎ方




```
city_record *head = NULL;
p = new_record();
fscanf(“”,p->name,&p_population);
p->next = head;
head = p;
p = new_record () ;
```

ループ

入力順リスト（課題13-2)

正順のつなぎ方