

アルゴリズム論 (第9回)



岩手県立大学
Iwate Prefectural University

佐々木研(情報システム構築学講座)

講師 山田敬三

k-yamada@iwate-pu.ac.jp

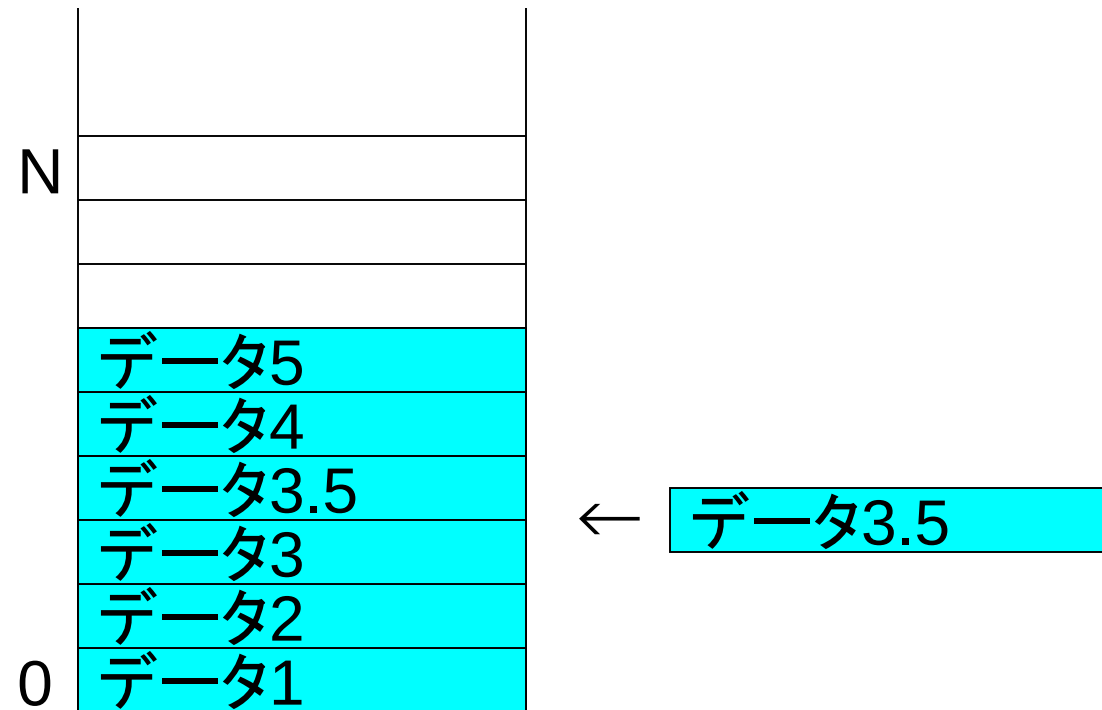
線形連結リスト



岩手県立大学
Iwate Prefectural University

配列

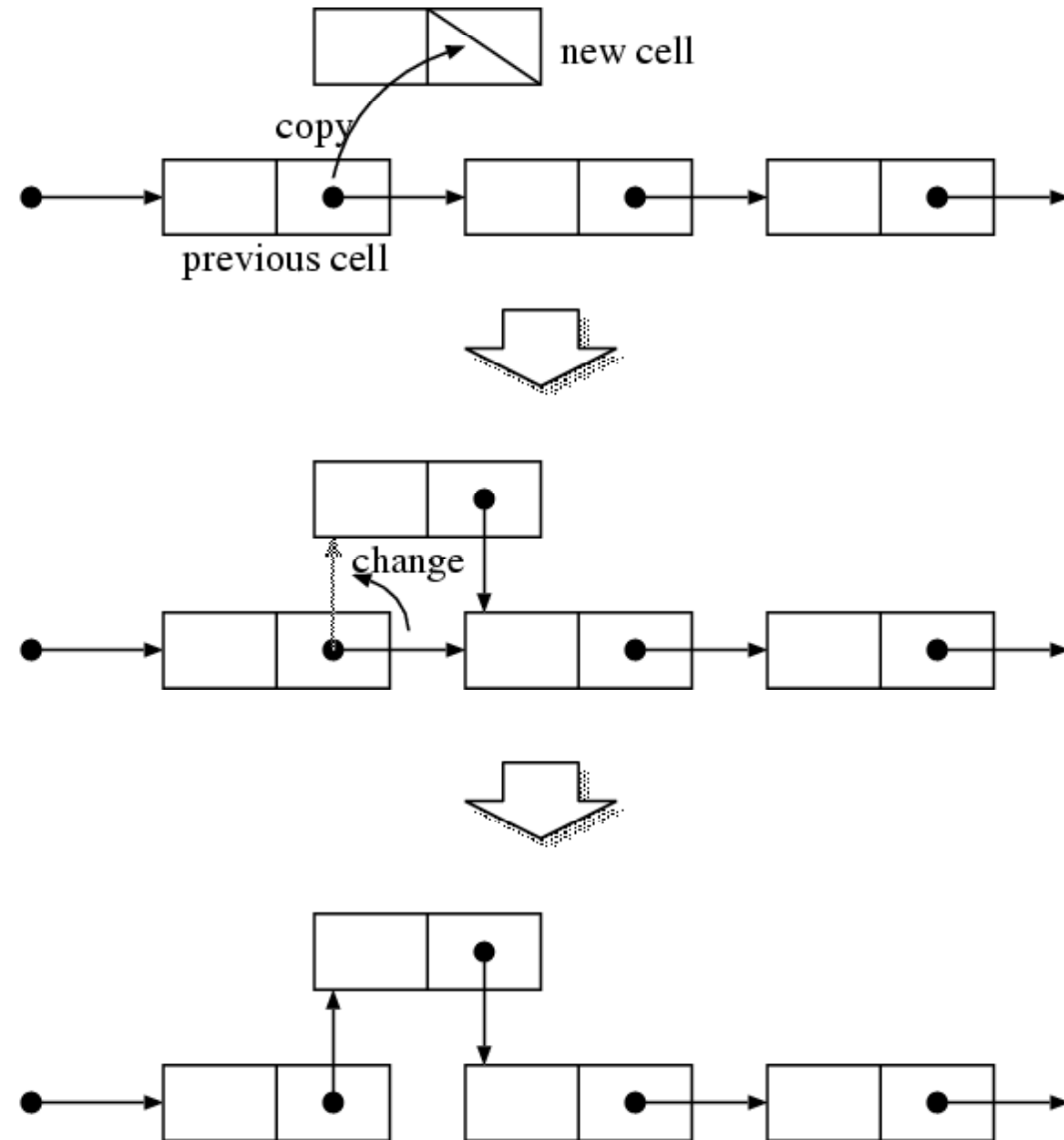
- リストの途中での追加, 削除は困難



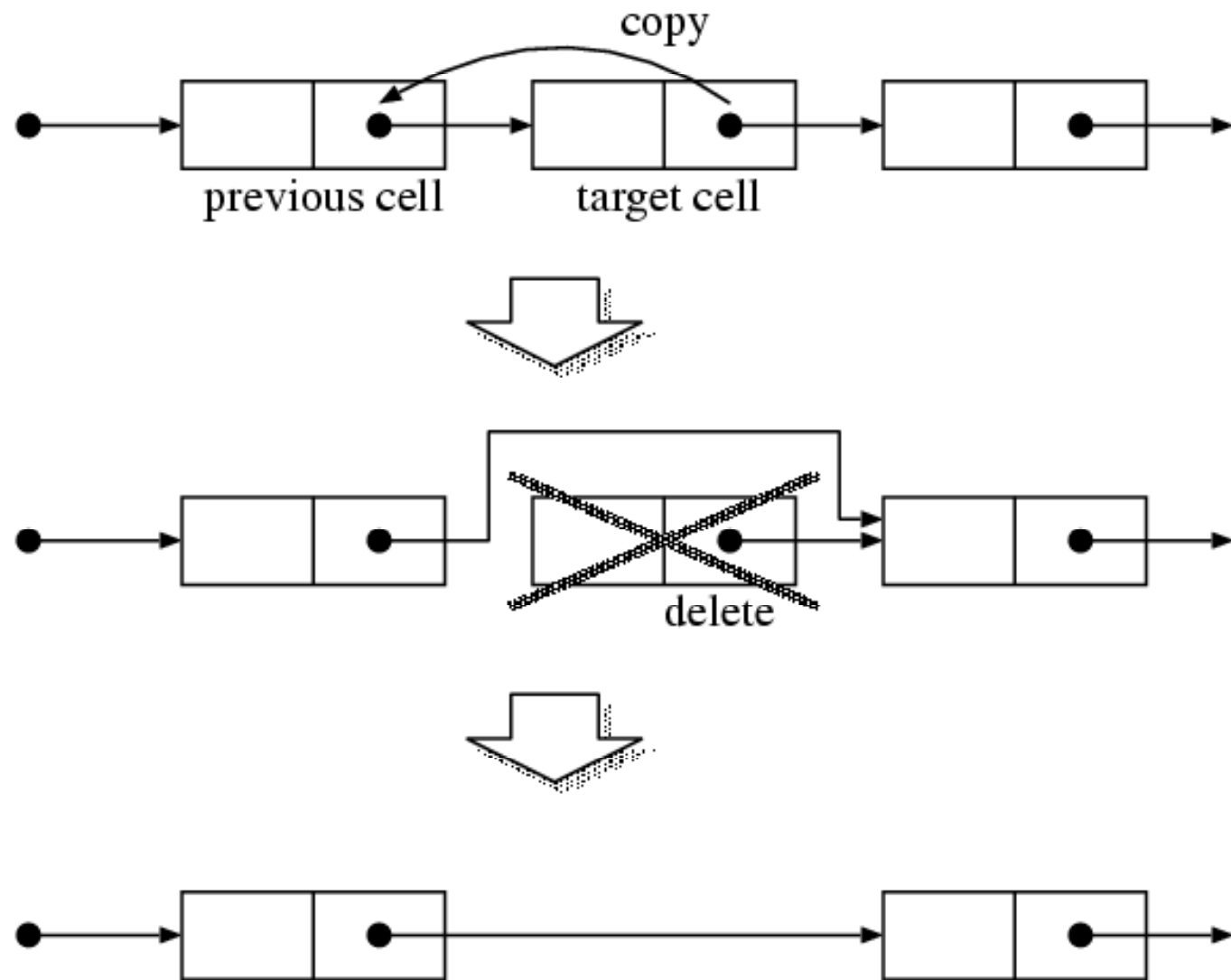
線形連結リスト (Linked List)

- 各要素をポインタで結合
 - 相対的に参照
- 配列の要素番号のようなインデックスはなし
 - 「何番目のデータ」という絶対的な参照は不可

セルの挿入

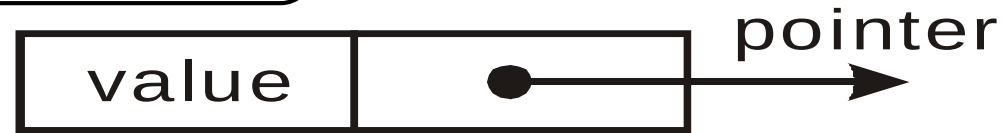


セルの削除



線形連結リスト (Linked List)

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



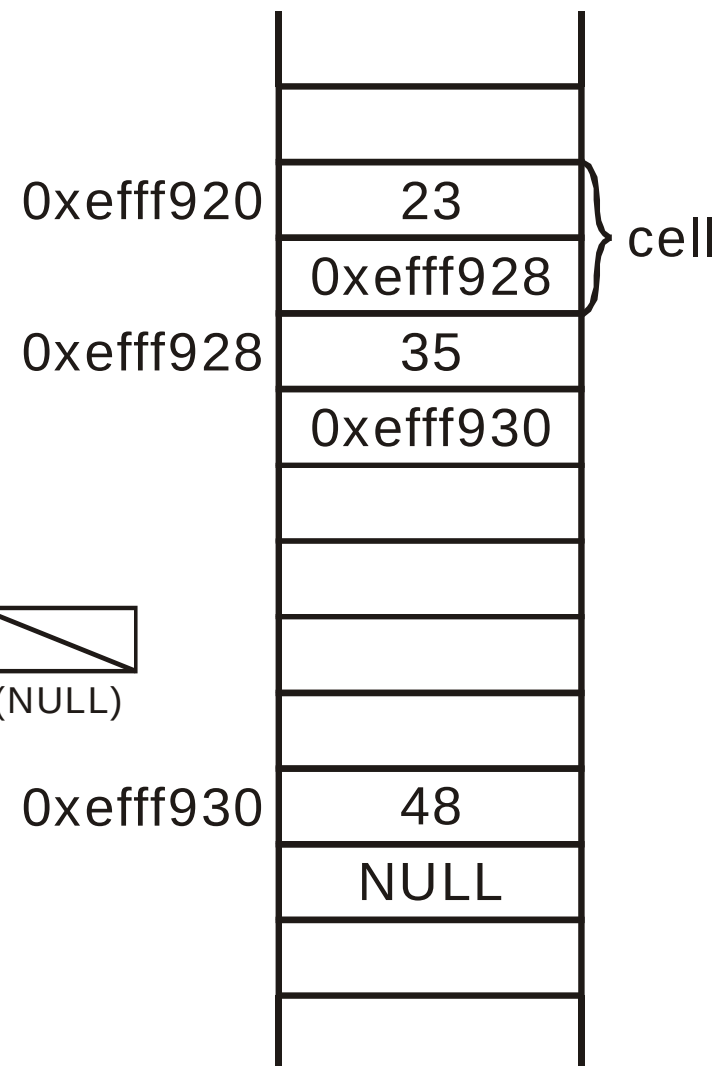
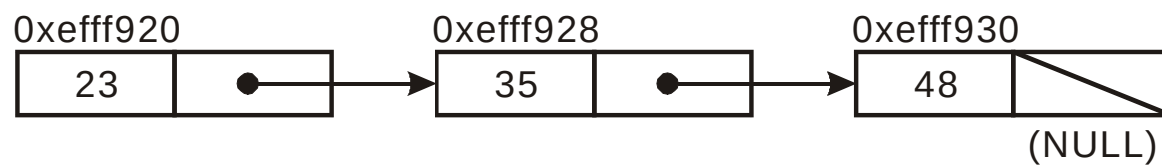
セル

- セル

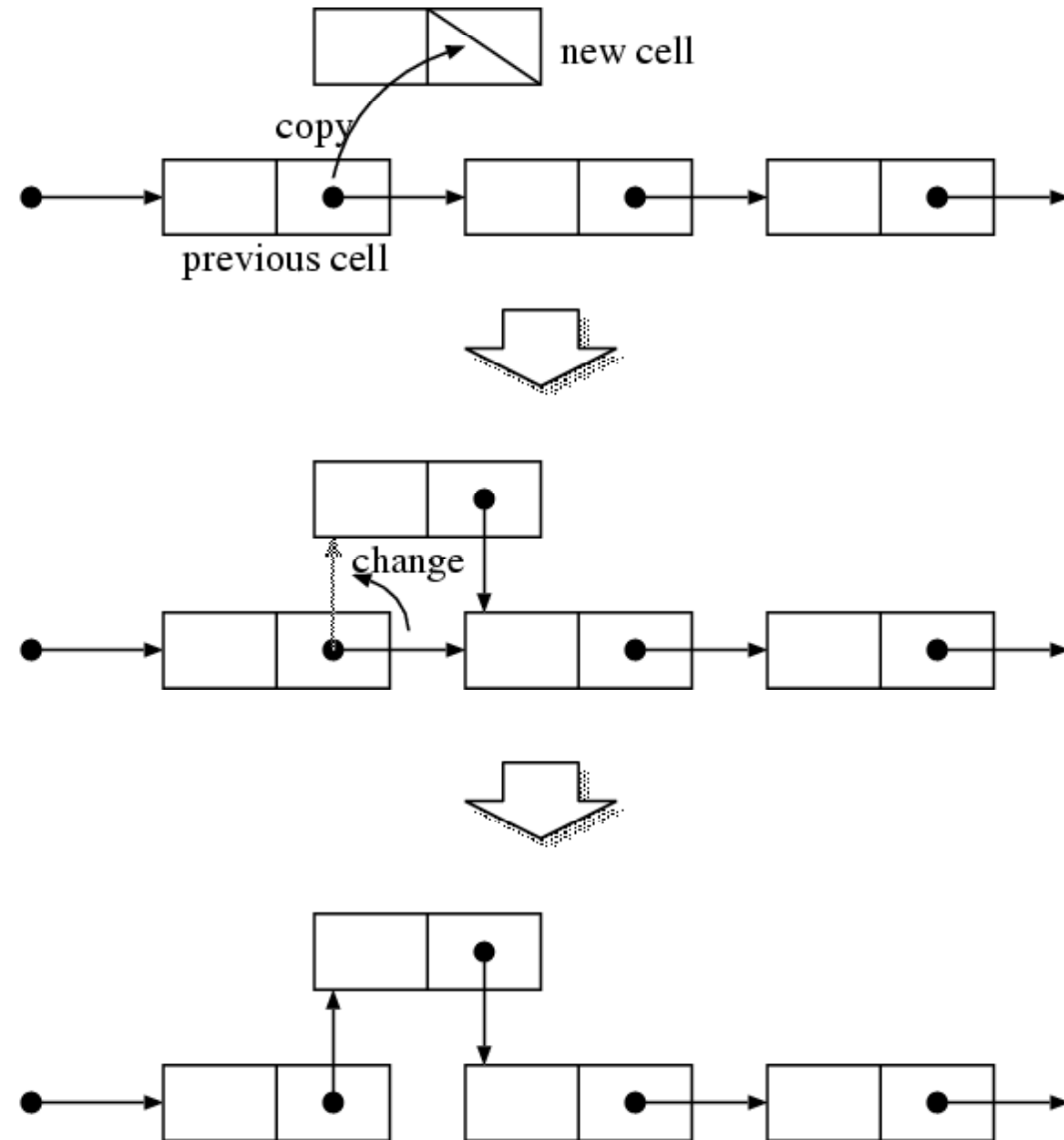
- データ, 及び, 次の要素の格納場所を示すポインタで構成
- 最後のデータの, 次を示すポインタは **NULL**

線形連結リスト (Linked List)

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



セルの挿入



セルを挿入する関数

```
void insert_cell(struct cell  
    **pointer, int new_value)
```

- 挿入する位置を示すポインタへのポインタと、格納する値を引数とする.
 ※ポインタ自身を変更する必要があるため

セルを挿入する関数

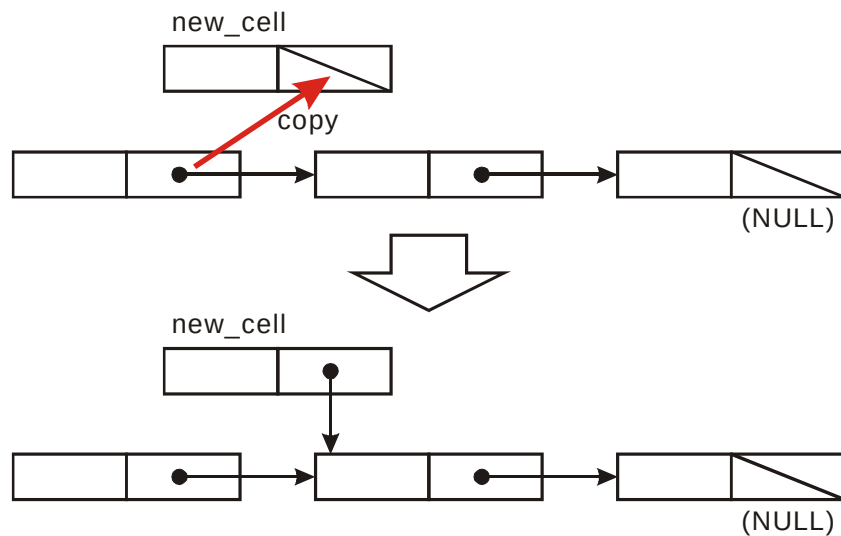
1. 新たなセルを用意し, 値を代入

```
new_cell=(struct cell *)malloc  
    (sizeof(struct cell));  
new_cell->value=new_value;
```

セルを挿入する関数

2. 新たなセルの次へのポインタに, 元の場所へのポインタをコピー

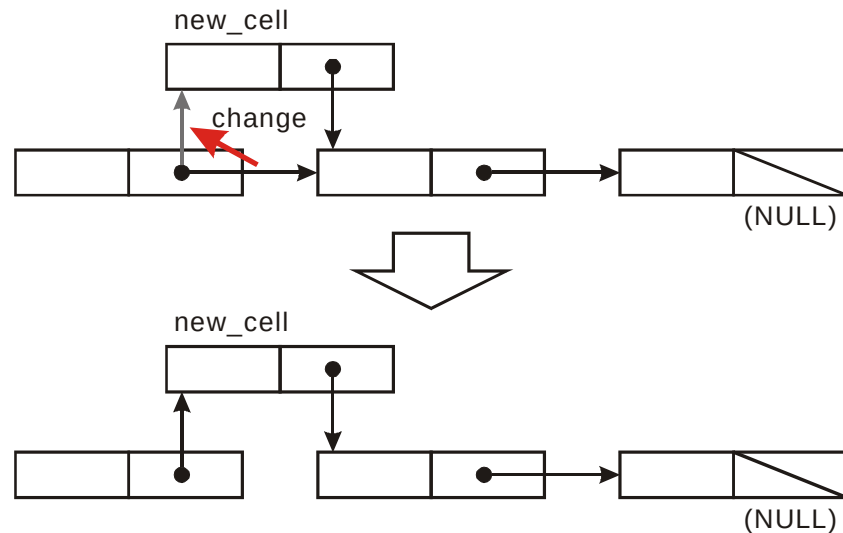
```
new_cell->next=*pointer;
```



セルを挿入する関数

3. 元のポインタの値を新たなセルに変更

```
*pointer=new_cell;
```



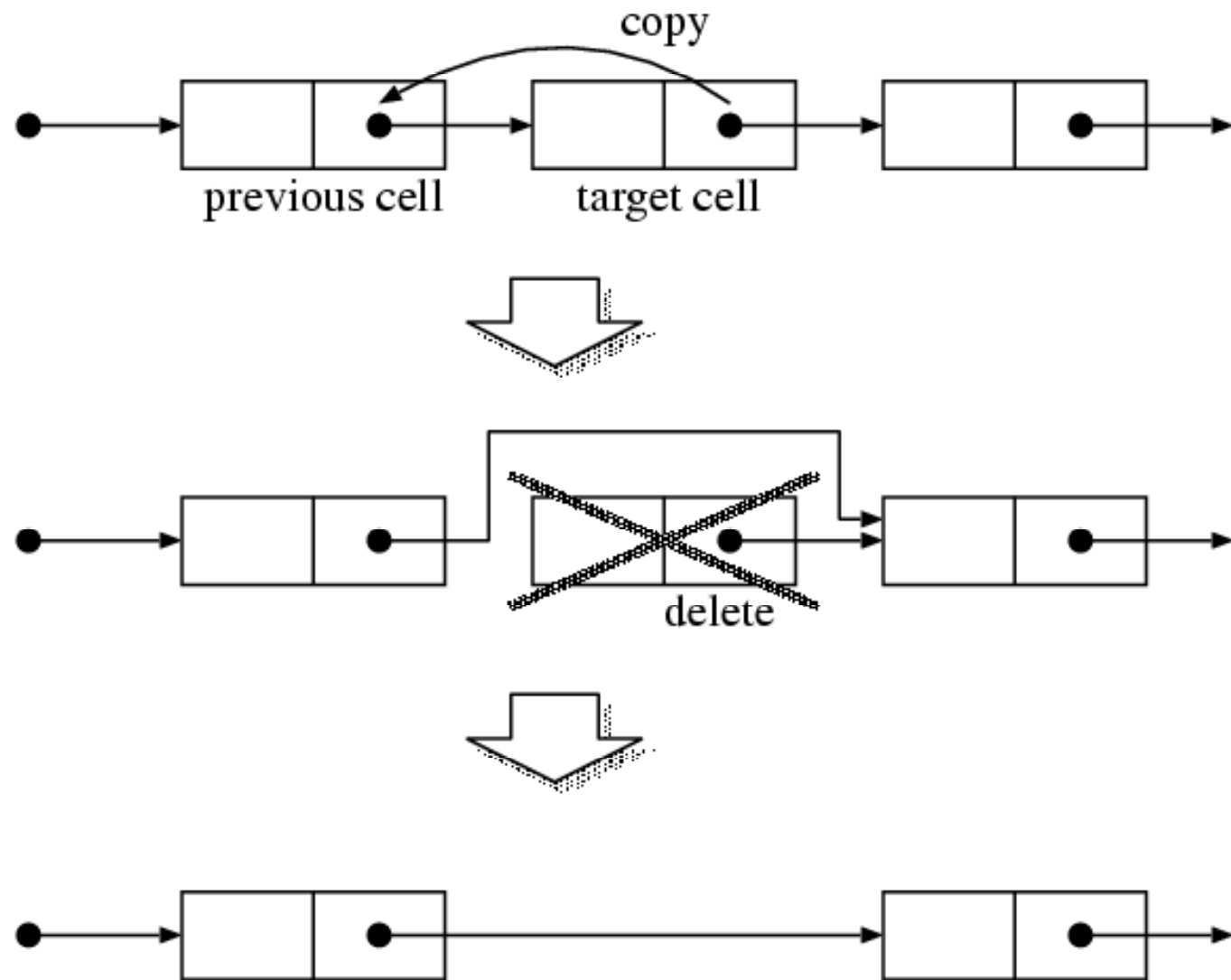
セルの挿入

```
struct cell *head=NULL, **p;  
p=&head;  
while(...) {  
    insert_cell(p, data);  
    p=&(( *p )->next);  
}
```

- 引数

- 挿入場所のアドレスを格納している変数のアドレス
- つまり, 前のセルの次へのポインタが格納されている変数(next)のアドレス

セルの削除



セルを削除する関数

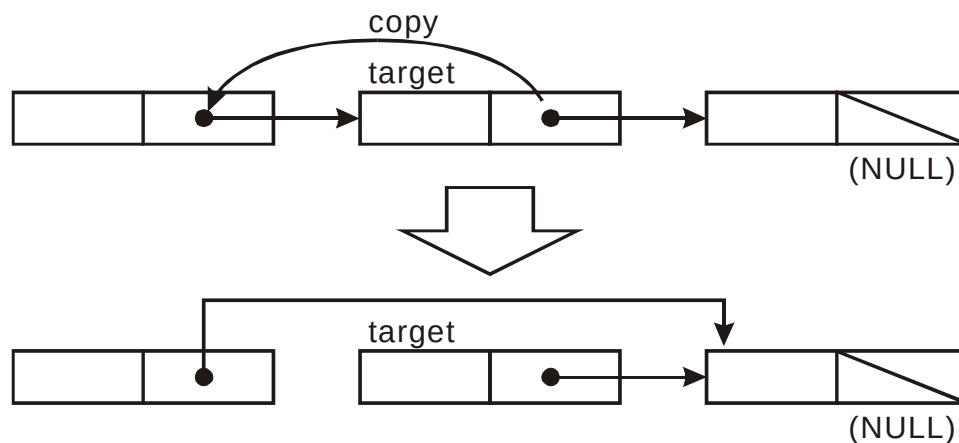
```
void delete_cell(struct cell **pointer)
```

- 削除するセルへのポインタへのポインタを引数とする.
 - つまり, 前のセルの, 次のセルへのポインタのアドレス
- ※ポインタ自身を変更する必要があるため

セルを削除する関数

1. 削除したいセルへのポインタを求め、元のポインタの値を、削除するセルの後のセルに変更

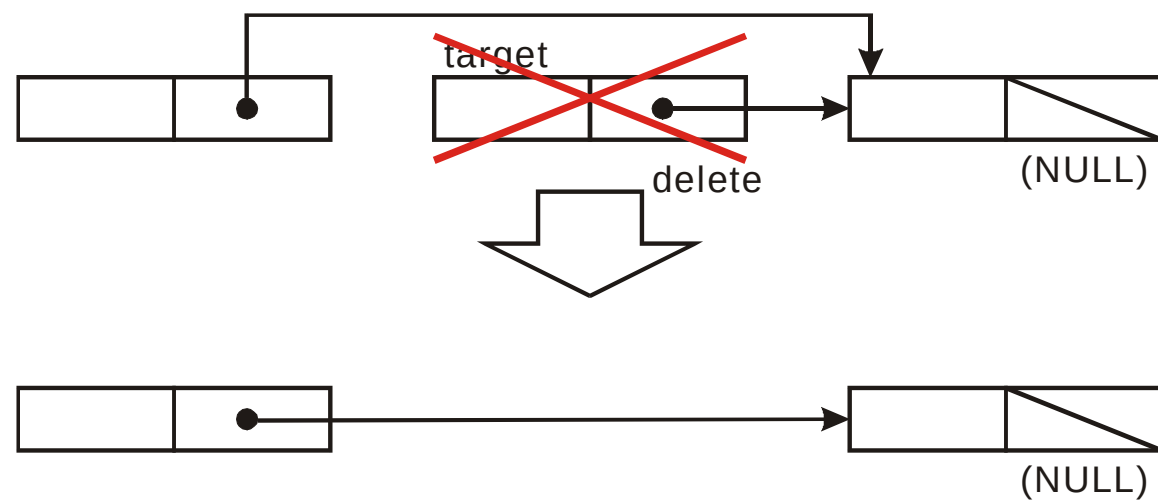
```
target=*pointer;  
*pointer=target->next;
```



セルを削除する関数

2. セルを削除

```
free(target);
```



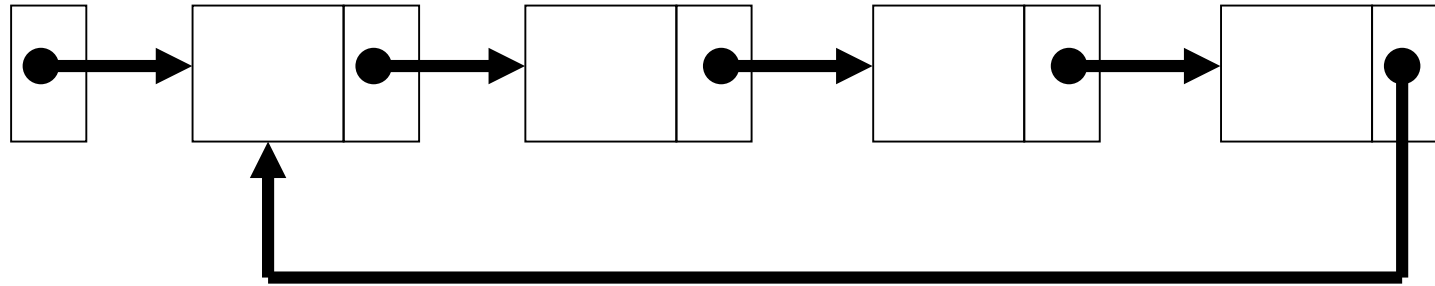
循環・重連結リスト



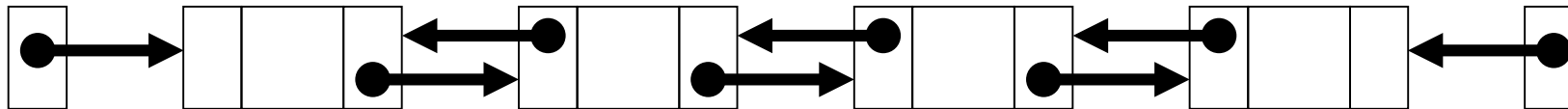
岩手県立大学
Iwate Prefectural University

循環リストと重連結リスト

- 循環リスト (circular list)

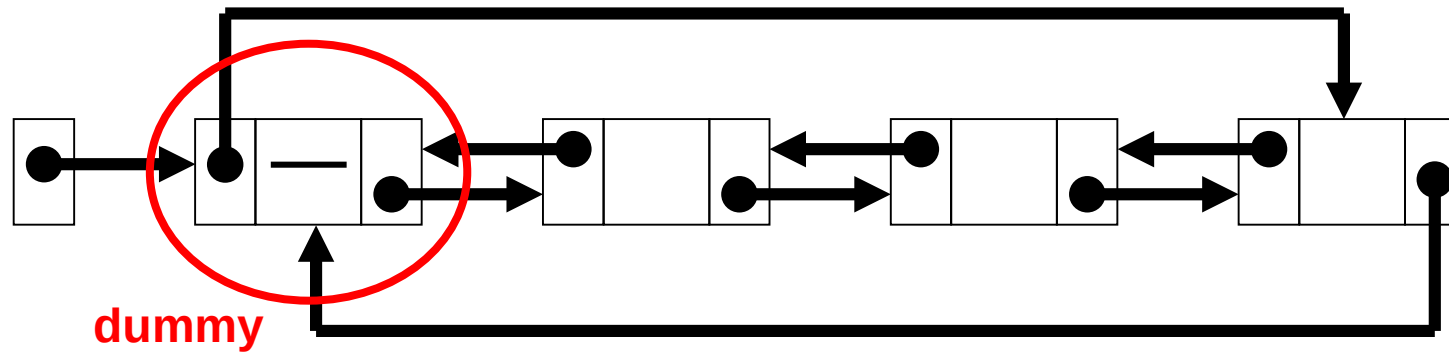


- 重連結リスト (doubly linked list)

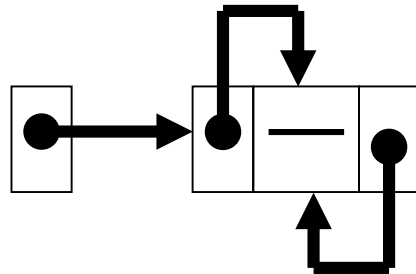


循環リストと重連結リスト

- 循環・重連結リスト

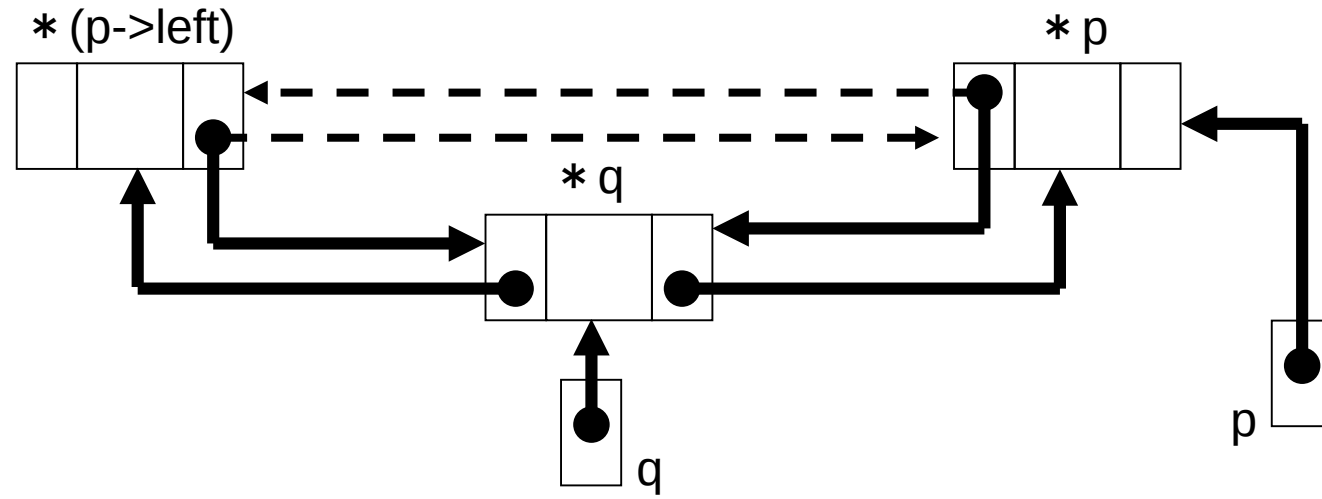


- 空の循環・重連結リスト

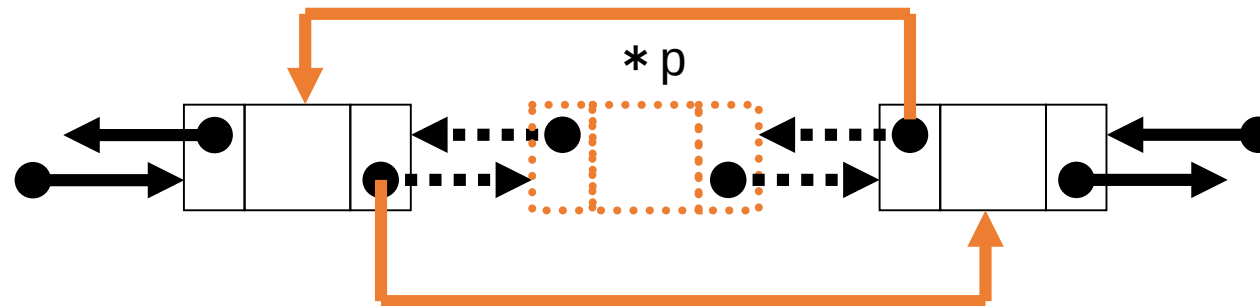


循環リストと重連結リスト

- ノードの挿入



- ノードの削除



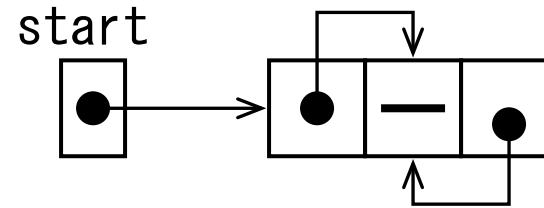
循環・重連結リスト

```
typedef struct Node {
    int num;
    struct Node *left, *right;
} node;

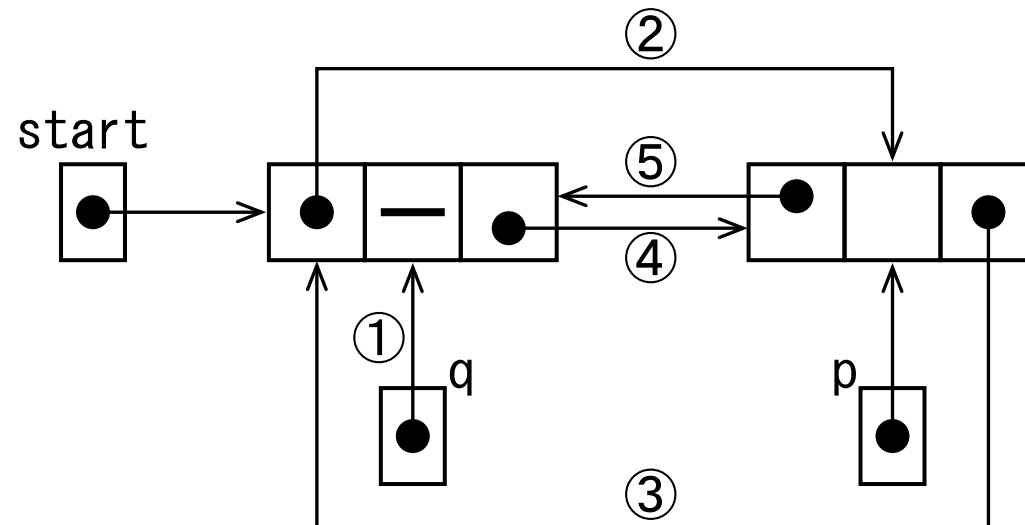
int insert_cdl_list(int x)
{
    node *q, *p = (node*)malloc(sizeof(node));
    if (p == NULL)
        return 0;
    p->num = x;
    ① q = start->left;
    ② start->left = p;
    ③ p->right = start;
    ④ q->right = p;
    ⑤ p->left = q;
    return 1;
}

int insert_left(node *p, int x)
{
    node *q = (node*)malloc(sizeof(node));
    if (q == NULL)
        return 0;
    ⑥ q->left = p->left;
    q->num = x;
    ⑦ q->right = p;
    ⑧ p->left->right = q;
    ⑨ p->left = q;
    return 1;
}
```

・初期状態



・関数 insert_cdl_list を実行



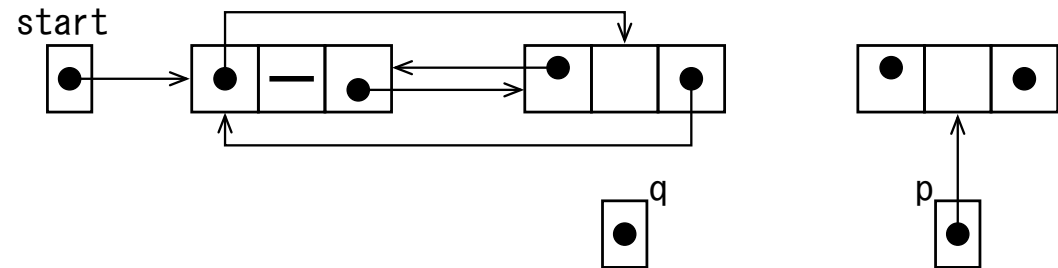
問題1

```
typedef struct Node {
    int num;
    struct Node *left, *right;
} node;

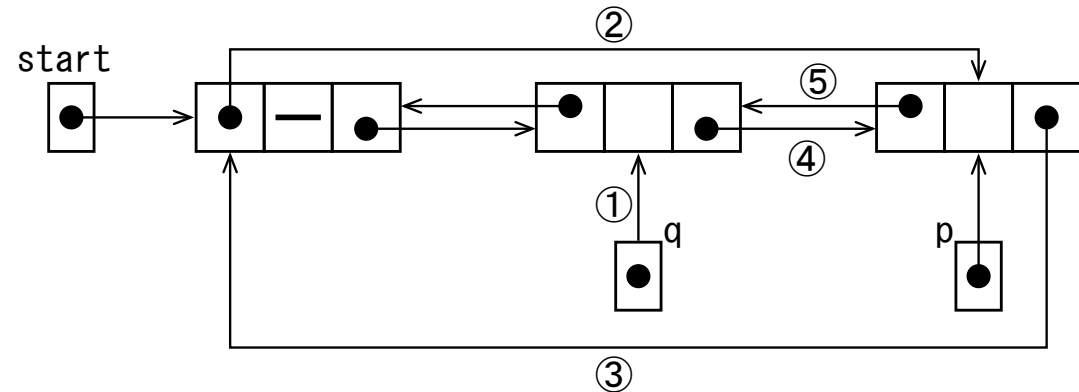
int insert_cdl_list(int x)
{
    node *q, *p = (node*)malloc(sizeof(node));
    if (p == NULL)
        return 0;
    p->num = x;
    ① q = start->left;
    ② start->left = p;
    ③ p->right = start;
    ④ q->right = p;
    ⑤ p->left = q;
    return 1;
}

int insert_left(node *p, int x)
{
    node *q = (node*)malloc(sizeof(node));
    if (q == NULL)
        return 0;
    ⑥ q->left = p->left;
    q->num = x;
    ⑦ q->right = p;
    ⑧ p->left->right = q;
    ⑨ p->left = q;
    return 1;
}
```

・再度，関数 insert_cdl_list を
実行した結果を答えなさい。



・解答



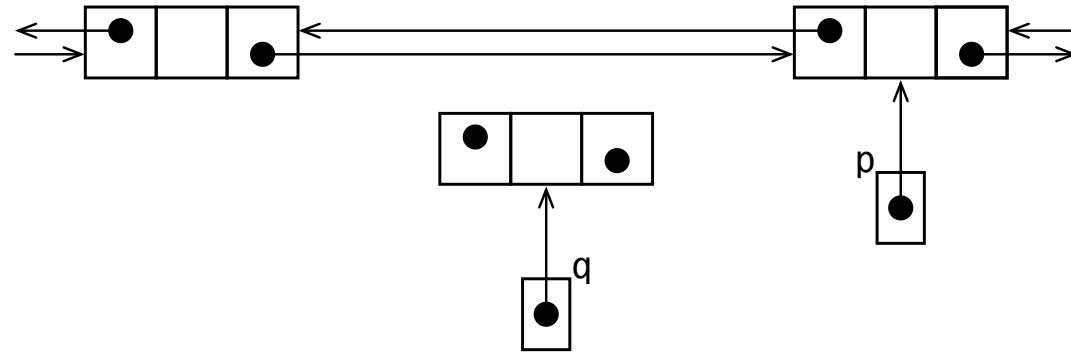
問題2

```
typedef struct Node {
    int num;
    struct Node *left, *right;
} node;

int insert_cdl_list(int x)
{
    node *q, *p = (node*)malloc(sizeof(node));
    if (p == NULL)
        return 0;
    p->num = x;
    ① q = start->left;
    ② start->left = p;
    ③ p->right = start;
    ④ q->right = p;
    ⑤ p->left = q;
    return 1;
}

int insert_left(node *p, int x)
{
    node *q = (node*)malloc(sizeof(node));
    if (q == NULL)
        return 0;
    ⑥ q->left = p->left;
    q->num = x;
    ⑦ q->right = p;
    ⑧ p->left->right = q;
    ⑨ p->left = q;
    return 1;
}
```

・関数 insert_left を実行した
結果を答えなさい。



・解答

