連結リスト

実装-連結リストへの操作

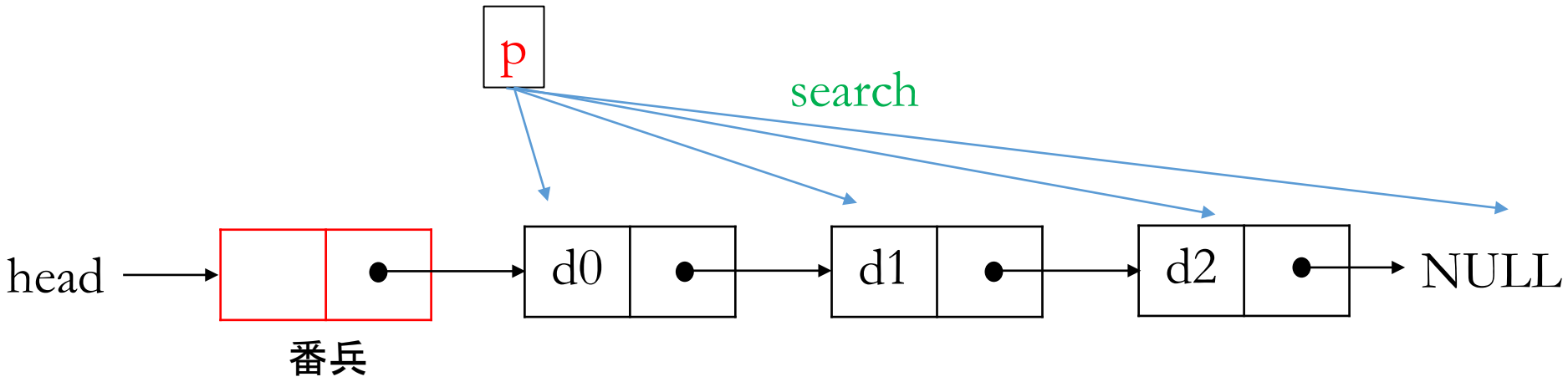
連結リストへの操作

- 操作：探索・更新・削除・挿入
- 本授業では以下の順で説明していく
  - 連結リスト（番兵あり）への操作
  - 連結リスト（番兵なし）への操作（発展）

探索

- 連結リストはノードに番号がないので、データ参照などは必ず探索しなければならない。探索にはデータを用いた場合と、何番目のノードという位置情報を用いた場合が考えられる
- ここではまずデータを用いた場合を説明する
- 挿入などに必要な位置情報を用いた探索はその時説明する

探索（イメージ）



pで各ノードを順に指し示して行き、探索データが見つかれば、探索終了。pがNULLまで到達したならばNULLを返して探索終了

探索の手順

1. 番兵のポインタ部に格納されているアドレスをポインタ変数pに代入する
2. pがNULLになるまで、以下を繰り返す
   2.1 探索データとpが指すノードのデータ部と照合し、一致するならばpを解とし終了
   2.2 一致しなければ、pの指すノードのポインタ部に格納されているアドレスをpに代入する
3. pを解とし終了

探索の関数

```
NODE *ListSearch(NODE *head, int x)
{
  NODE *p;

  p = ?1; // 注意：番兵あり！

  while(p != NULL){
    if(x == p->data) ?2;
    p = ?3;
  }

  return p;
}
```

manaba小テスト：04-1

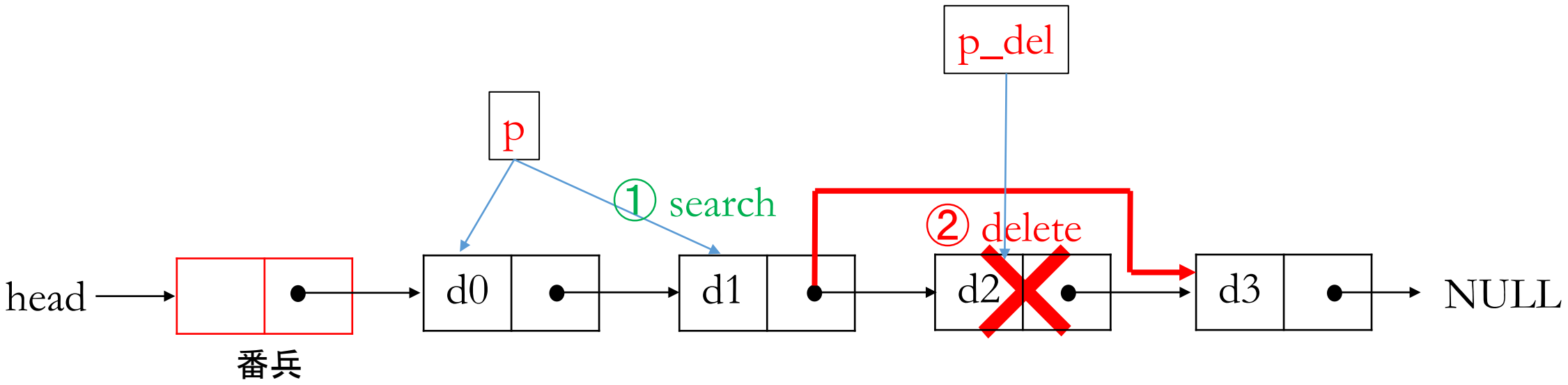
- 5分
- 6点

更新関数

```
void ListUpdate(NODE *head, int x, int y)
{
  NODE *p;

  p=ListSearch(head, x);
  p->data=y;
}
```

削除

d2ノードを削除の場合



①pでノードを順に指し示して行き、d1ノードで止める

②p->nextを2回辿って、d3ノードのアドレスを得、d1ノードのポインタ部に格納し、d2ノードを解放

削除の関数

```
//削除ノードより1つ前のノードを探索する
NODE *ListSearchPrev(NODE *head, int x)
{
  NODE *prev = head, *p;

  p=head->next;

  while(p != NULL){
    if(x == p->data) return prev;
    prev = p;
    p = p->next;}
  return p;
}
```

```
//削除
void ListDelete(NODE *head, int x)
{
  NODE *p, *p_del;

  p=ListSearchPrev(head, x);
  if(p == NULL) {
   printf("%d not found¥n", x); return;}
  p_del = p->next;
  p->next = p_del->next;
  free(p_del);
}
```

番兵の役割

- 先頭d0の削除を特別視しなくて済む。つまり、先頭か先頭でないかを区別せずに、（削除の）次のノードのアドレスを（削除の）前のノードのポインタ部に格納すればよい
- 番兵がなければ（「ポインタのポインタ」も用いなければ）、先頭か先頭でないかを判別し、先頭であれば次のノードのアドレスをheadに、先頭でなければ削除ノードより前のノードのポインタ部に、というように処理しなければならない

manaba小テスト：04-2

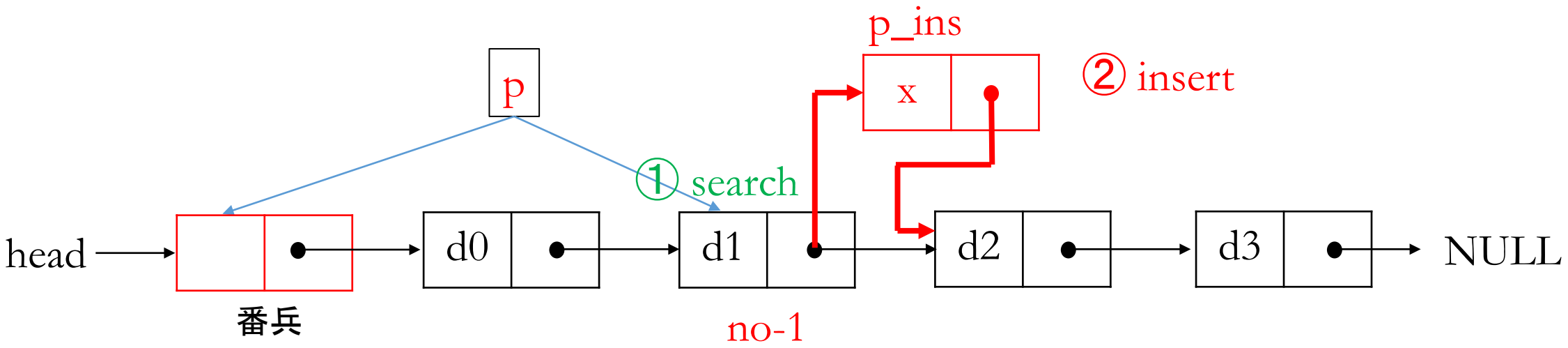
- 10分
- 10点

挿入について

- ノード指定の挿入は現実的ではない
  - 指定ノードの「後ろ」に挿入となれば、先頭としての挿入ができない
  - 指定ノードの「前」に挿入となれば、末尾としての挿入ができない
- 挿入は位置指定で行うのが普通。たとえば、「n番目として挿入」であれば挿入後その新しいノードはn番目に位置する（n=0, 1, 2, ...）

挿入

no番目のノードとして挿入（no=2の場合）



①pでノードを順に指し示して行き、no-1番目にあるd1ノードで止める

② 挿入ノードを作成し、そのアドレスをd1ノードのポインタ部に格納し、d2ノードのアドレスを挿入ノードのポインタ部に格納する

番兵の役割

- 先頭への挿入を特別視しなくて済む。つまり、先頭か先頭でないかを区別せずに、挿入ノードのアドレスを前のノードのポインタ部に格納すればよい
- 番兵がなければ（「ポインタのポインタ」も用いなければ）、先頭か先頭でないかを区別し、先頭であれば挿入ノードのアドレスをheadに、先頭でなければ前のノードのポインタ部に、というように処理しなければならない

挿入の関数

```
// no-1番目ノードを探索
NODE *ListSearchNo(NODE *head, int no)
{
  NODE *p = head;

  for(int i=0; i<no && p->next != NULL; i++)
    p = ?1;

  return p;
}
```

```
//挿入
void ListInsert(NODE *head, int no, int x)
{
  NODE *p, *p_ins;
  if(no<0) {
    printf("%d not found¥n", no); return;}

  p = ListSearchNo(head, no);

  p_ins = malloc(sizeof(NODE));
  p_ins->next = ?2;
  p_ins->data = ?3;
  p->next = ?4;
}
```

manaba小テスト：04-3

- 8分
- 8点

プログラムを実行してみる

```
#include <stdio.h>
#include “list.h” // 番兵あり

void ArrayPrint(int n, int a[])
{
  for(int i=0; i<n; i++) printf("%d ", a[i]);
  printf("¥n");
}

int main(void)
{
  int i, na=6, a[]={10, 20, 30, 40, 50, 60};
  int nb=4, b[]={10, 100, 30, 60}, pos[]={0, 2, -10, 1000};
  NODE *head, *p;

  head=ListMake(na, a);
  ListPrint(head); // 講義資料より詳細情報出力
```

```
  printf("¥nSearch: ");
  ArrayPrint(nb, b);
  for(i=0; i<nb; i++){
    p=ListSearch(head, b[i]);
    if(p==NULL) {printf("%d not found!¥n", b[i]);
      continue;}
    printf("addr. %p, val. %d¥n", p, p->data);
  }

  printf("¥nDelete: ");
  ArrayPrint(nb, b);
  for(i=0; i<nb; i++)
    ListDelete(head, b[i]);
  ListPrint(head);
```

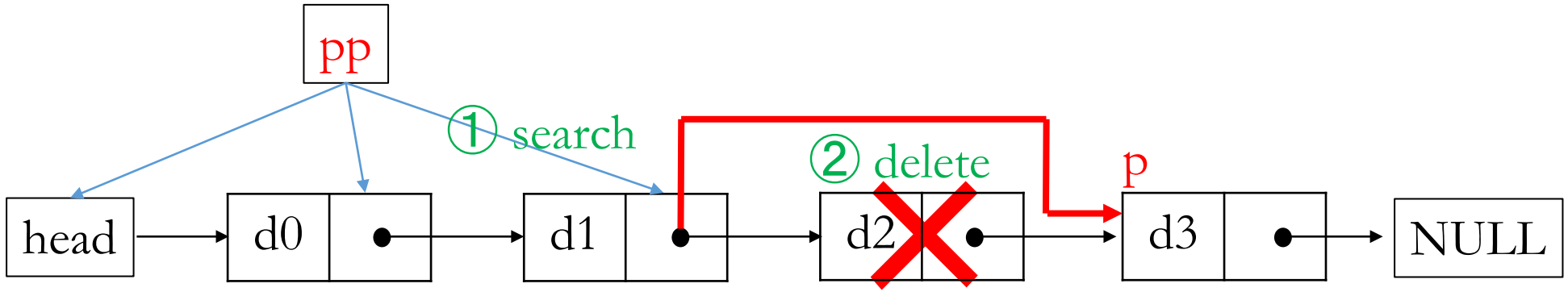
プログラムを実行してみる

```
  printf("¥nInsert: ");
  for(i=0; i<nb; i++)
    printf("(%d, %d) ", pos[i], b[i]);
  printf("¥n");

  for(i=0; i<nb; i++)
    ListInsert(head, pos[i], b[i]);
  ListPrint(head);

  for(i=0; i<2; i++){
    printf("¥nupdate: 100->30¥n");
    ListUpdate(head, 100, 30);
    ListPrint(head);
  }
  return 0;
}
```

```
./a.out
list
no. 0, addr. 0x55891bbe42c0, val. 10
no. 1, addr. 0x55891bbe42e0, val. 20
no. 2, addr. 0x55891bbe4300, val. 30
no. 3, addr. 0x55891bbe4320, val. 40
no. 4, addr. 0x55891bbe4340, val. 50
no. 5, addr. 0x55891bbe4360, val. 60

Search: 10 100 30 60
addr. 0x55891bbe42c0, val. 10
100 not found!
addr. 0x55891bbe4300, val. 30
addr. 0x55891bbe4360, val. 60

Delete: 10 100 30 60
100 not found
list
no. 0, addr. 0x55891bbe42e0, val. 20
no. 1, addr. 0x55891bbe4320, val. 40
no. 2, addr. 0x55891bbe4340, val. 50
```

```
Insert: (0, 10) (2, 100) (-10, 30) (1000, 60)
-10 not found
list
no. 0, addr. 0x55891bbe4360, val. 10
no. 1, addr. 0x55891bbe42e0, val. 20
no. 2, addr. 0x55891bbe4300, val. 100
no. 3, addr. 0x55891bbe4320, val. 40
no. 4, addr. 0x55891bbe4340, val. 50
no. 5, addr. 0x55891bbe42c0, val. 60

update: 100->30
list
no. 0, addr. 0x55891bbe4360, val. 10
no. 1, addr. 0x55891bbe42e0, val. 20
no. 2, addr. 0x55891bbe4300, val. 30
no. 3, addr. 0x55891bbe4320, val. 40
no. 4, addr. 0x55891bbe4340, val. 50
no. 5, addr. 0x55891bbe42c0, val. 60
```

番兵なしの削除（ポインタのポインタ利用）



①ppでポインタ部を順に指し示して行き、d1のポインタ部で止める。その時点*ppがd2のアドレス

```
x=d2;
while(*pp != NULL){
    if(x==(*pp)->data) break;
    pp = &((*pp)->next);}
```

②d3ノードのアドレスpを得、それをppで指し示しているポインタ部に入れる

```
p=(*pp)->next;
free(*pp);   // d2ノードを解放
*pp=p;
```

番兵なしの挿入（ポインタのポインタ利用）