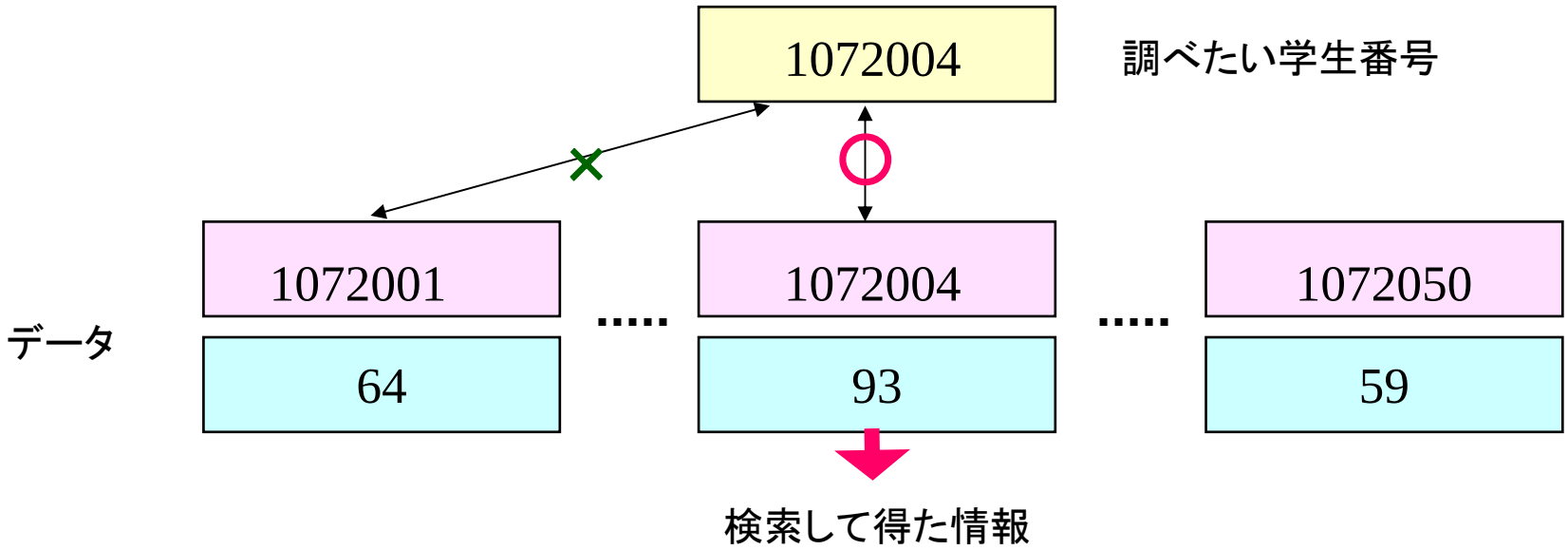
プログラミング1
第10回
構造体(3) -- 応用

- リスト操作

この資料にあるサンプルプログラムは
`/home/course/prog1/public_html/2007/HW/lec/sources/`
下に置いてありますから、各自自分のディレクトリに
コピーして、コンパイル・実行してみてください

データ構造：リスト

- データが「一連」のメモリに保管される
- 最も簡単なデータ構造は配列である
- 「リニアサーチ」は配列を検索するための方法だった



- もっと効率的なデータ構造として「連結リスト(linked list)」がある

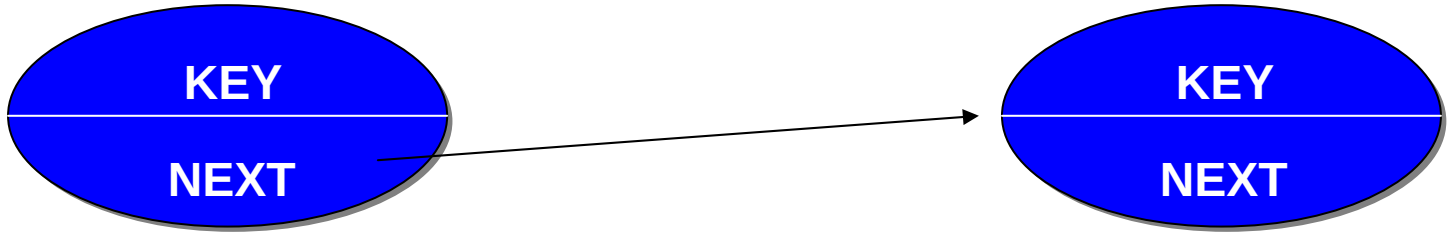
連結リスト (list)

- 「リスト」はデータ構造の一種
- 何らかのしかけで一方向又は双方向に順に繋がる（連結）

- 矢印で繋がった各データを「**ノード（節） 又はセル**」と呼ぶ。
- リストを矢印の方向にたどることが出来る。
- 連結リストの実現（実装）方法は様々である。ここでは、教科書6.1とはちょっと違う実装を試みる。
- この授業では一方向のものだけを説明する。双方向につながる「双方向リスト」もあるので興味の有る人は調べてみると良い。

ノード

- 各ノードは構造体(タグnode)で実現されており、以下の要素を持つ
  - データを保管する領域:この例においてはint型変数 **key**
  - 次のノードを示す領域:node型構造体ポインタ **next**




```
struct node {
  int key;
  struct node *next;
};
typedef struct node * NodePointer;
```

簡単のためNode型ポインタを**NodePointer**と**typedef**する

•構造体の宣言の中に自分自身の構造体へのポインタを持つものを「自己参照的構造体」と言う

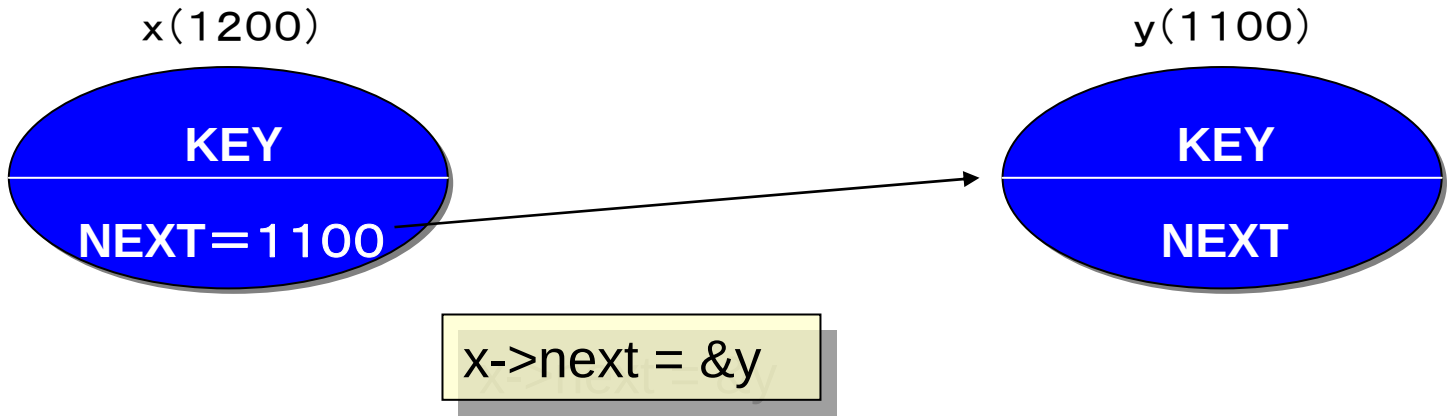
•データを保管する領域には、**key**以外に他のデータも入れることができる

リストの特徴

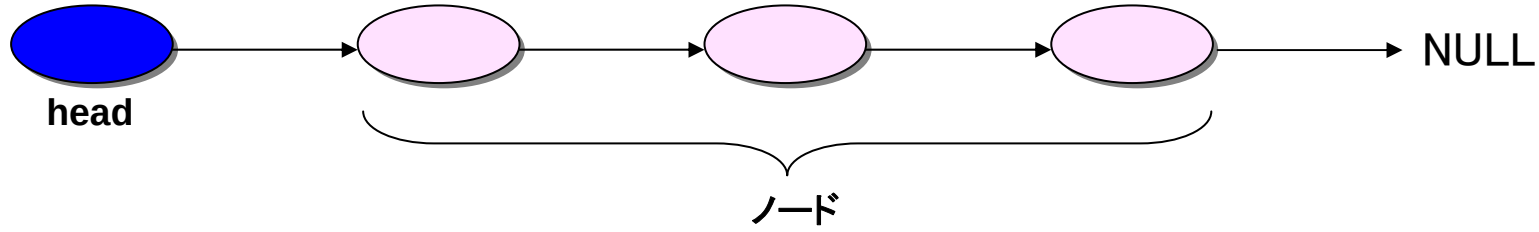
- リストは配列とは異なり、各ノードがメモリの中で順に並んでいる必要はないし、sizeof(struct node)のアドレスだけ離れている必要もない
- 大量のメモリ空間を一括に確保できなくても容易に利用できるし、必要に応じて長さを動的に変更することも簡単である
- 配列より少し複雑になるが、メモリをより効率的に利用できる特徴がある

ノードの連結

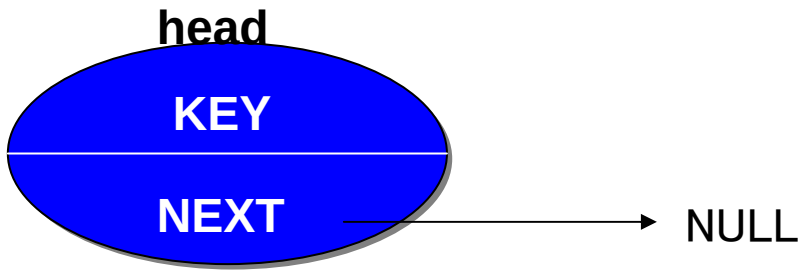
- 次のノードへのリンク（矢印）の実現
  （x → y の順で繋げるとする）
  - ノードx、yのアドレスがそれぞれ1200番地、1100番地だとする
  - ノードxのnext（つまり **x->next**）に次のノード（つまりy）のアドレス（つまり、1100）というアドレスを入れる
  - これでx → yと言う連結が出来た事になる。
  - リストをたどるには順にnextのアドレスをたどれば良い。



リストの構造



- 便宜的に先頭(head)ノードを連結リストに付加する。
  - 先頭（head）はリスト先頭にあり、前のノードがない
  - headではKeyは使わないので不定とする
  - headは外部変数として定義されてる。（複数のリストを扱うときは不便）
- リストの終端(最後のノード）はヌルポインタ(**NULL**)となっている
- 初期状態はhead のみが存在し、nextはヌルポインタとなっている（下図）
- リスト構造の実現方法は様々なバリエーションがある。



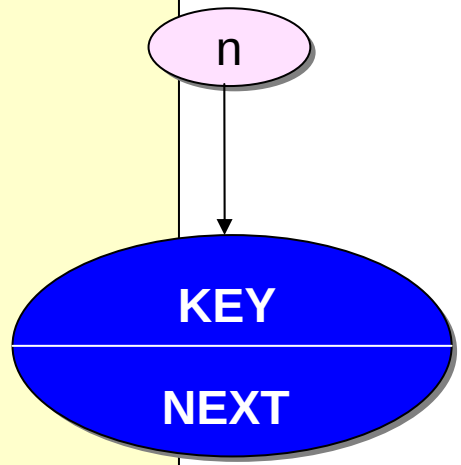
新しいノードを作る：ノードへのメモリの割り当て

- 連結リストでは通常必要な分だけノードを作成する。このような方法を、実行の途中にメモリを割り当てることから、動的リストと呼ぶ。
- 以下がノード作成の関数である。
- ノード1個分のメモリ割り当てでは、割り当てるバイト数は sizeof(struct node) である。

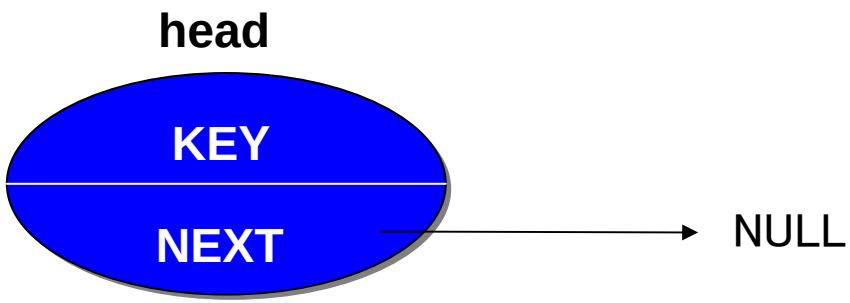
```
NodePointer make_1node(int keydata, NodePointer p){
  NodePointer n;

  if((n = malloc(sizeof(struct node))) == NULL) {
    printf("Error in memory allocation¥n");
    exit(8);
  }
  n->key = keydata;
  n->next = p;

  return n;
}
```

何らかの理由でmallocが失敗したら終了する

keyとnextに引数の値をセットする。



リストの初期化



リストの初期化は簡単である。
行うことは以下の2つで、関数
make_1node を呼ぶだけである
・headノードの作成
・nextをNULL(ヌルポインタ)にする

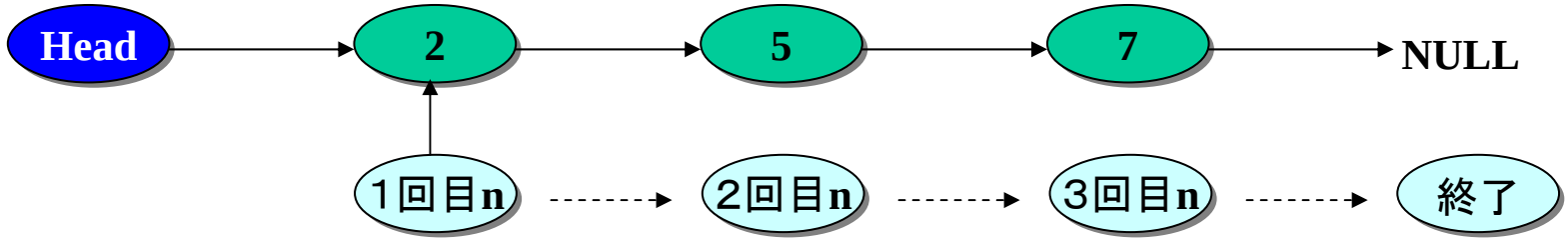
```
NodePointer head;
```

headはmainの外で定義
つまり外部(グローバル)
変数

```
head = make_1node(0,NULL);
```

headノードを作成する
値は仮に0とする(何でも良い)
nextにNULL(ヌルポインタ)を設定する

リストの表示



- head から NULLの前まで key の値を出力

例の場合の出力:
Head - 2 - 5 - 7

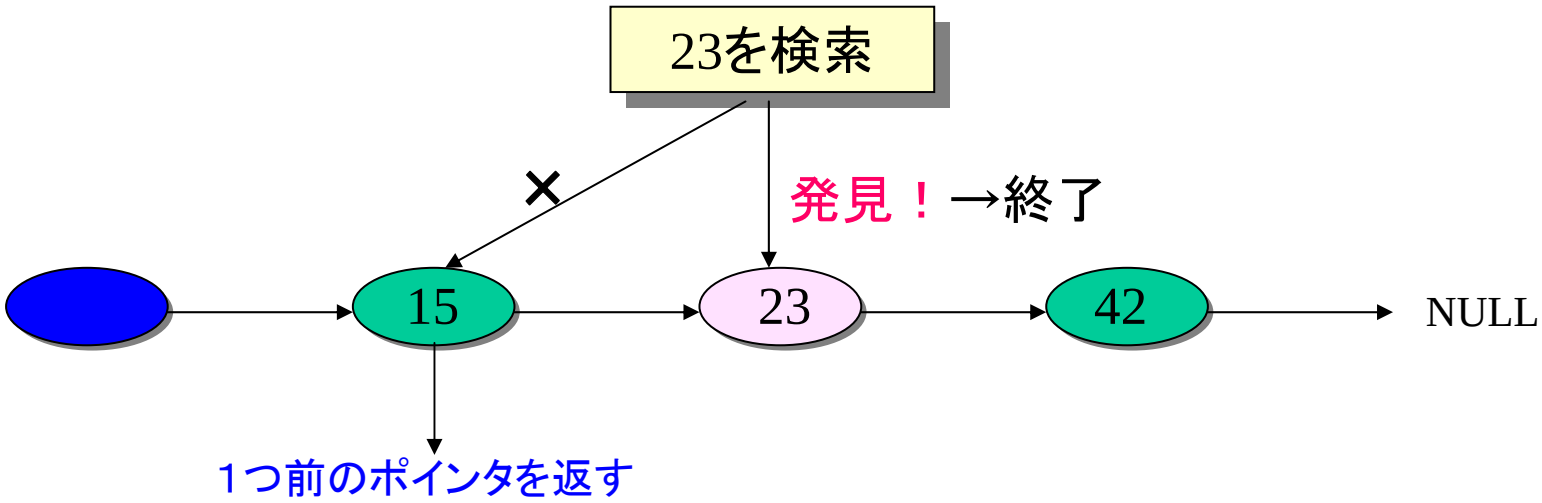
```
void listprint(void){
  NodePointer n;

  printf("Head");
  for(n = head->next; n != NULL; n = n->next){
    printf(" - %d", n->key);
  }
  printf("\n");
}
```

headの次からスタート

NULLになるまでノードを一つずつ表示する

リストを順にたどる

検索方法



- keyがkeydata（例では23）のノードn を検索する手順:
  ① 値keydataをheadの次からNULLの前まで順にkeyと比較する
  ② 発見したら一つ前のノードへのポインタを返す。
  ③ 発見出来なかったらNULLを返す。
- 一つ前のポインタを返す理由:削除の時に便利（後述）

検索ソース

```
NodePointer finditem(int keydata)
{
  NodePointer n;

  for(n = head; n->next != NULL; n = n->next){
    if(n->next->key == keydata) return n;
  }
  return NULL;
}
```

nextがNULLになるまで探す①

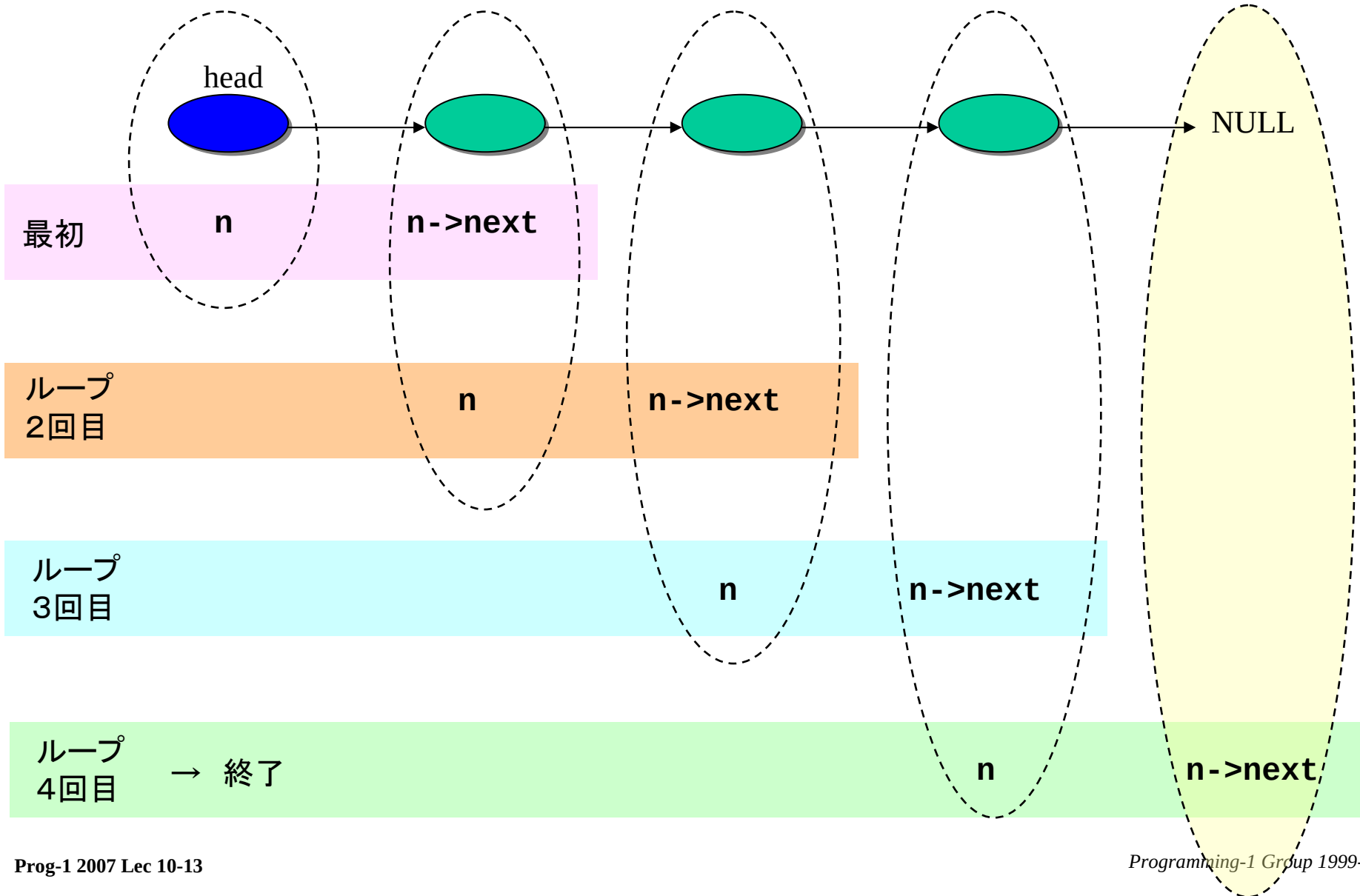
見つかったら一つ前の
ノード(n)をリターン②

次のノードのkeyと
keydataを比較する①

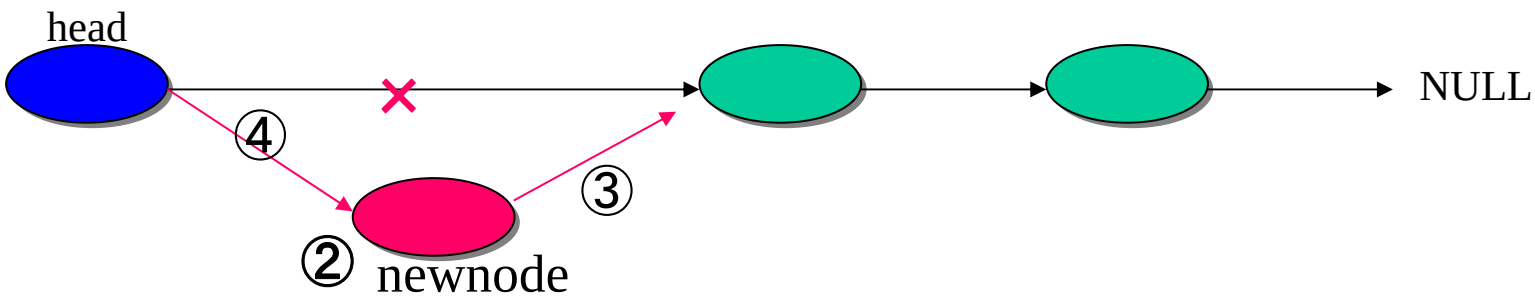
発見出来なかったら
NULLをリターン③

- **keydataをheadの次からNULLの前まで順にkeyと比較し、発見したら一つ前のノードへのポインタを返す；発見出来なかったらNULLを返す。**

検索イメージ



リストの構築(挿入)方法



- headの直後にkeyがkeydataの値のノードnewnode を挿入する手順:

① **keyがkeydataのノードを検索する。**
  - **もしあればNULLをリターンし、終了。**
  - **もしなければ(検索結果がNULLなら)以下の処理を行う。**

② **keyがkeydataのノード newnode を作成**

③ **newnode->next を head->next と同じ(代入)にする**

④ **head->next を newnode にする**

挿入ソース

- headの直後にkeyがkeydataのノードnewnode を挿入する

```
NodePointer insert(int keydata)
{
  NodePointer newnode;

  if(finditem(keydata) == NULL){
    newnode = make_1node(keydata,head->next);
    head->next = newnode;

    return newnode;
  }
  else return NULL;
}
```

keyがkeydataのノードを検索し、なければ新しいノードを作り、リストに挿入する;あればNULLを返す①

keyがkeydataのノードnewnodeを作成②

newnode->nextを**head->next**と同じ(代入)にする③

head->nextを**newnode**にする④

プログラム(1)

ヘッダファイル　list.h

```
/* struct declaration */
struct node {
  int key;
  struct node *next;
};
typedef struct node * NodePointer;

/* prototype declaration */
NodePointer  insert(int);
NodePointer  finditem(int);
void         listprint(void);
NodePointer  make_1node(int,NodePointer);


/* Global Variable head */
NodePointer head;
```

プログラム全体は
/home/course/prog1/public_html/2007/lec/source/{lec10-1.c,list.h}にある

プログラム(2)

```
#include <stdio.h>
#include <stdlib.h>
#include "list.h"
main(){
  int i,num;

  printf("[Initial]¥n");
  head = make_1node(0,NULL);
  for (i = 1; i <= 9 ; ++i) insert(i);
  listprint();

  printf("[Insert](enter number)¥n");
  while(scanf("%d",&num) == 1){
    if (insert(num) == NULL) printf("Data %d is already on the list¥n",num);
    listprint();
  }
}
```

プログラム(3)

```
NodePointer insert(int keydata){
  NodePointer newnode;

  if(finditem(keydata) == NULL){
    newnode = make_1node(keydata,head->next);
    head->next = newnode;

    return newnode;
  }
  else return NULL; /* in case of data found */
}

void listprint(void){
  NodePointer n;

  printf("Head");
  for(n = head->next; n != NULL; n = n->next) {
    printf(" - %d", n->key);
  }
  printf("\n");
}
```

プログラム(4)

```
NodePointer finditem(int keydata){
  NodePointer n;

  for(n = head; n->next != NULL; n = n->next){
    if(n->next->key == keydata) return n;
  }

  return NULL; /* in case of not found */
}

NodePointer make_1node(int keydata, NodePointer p){
  NodePointer n;

  if((n = malloc(sizeof(struct node))) == NULL) {
    printf("Error in memory allocation\n");
    exit(8);
  }

  n->key = keydata;
  n->next = p;

  return n;
}
```

実行結果

```
s1000000{std0ss0}:1 ./a.out
[Initial]
Head - 9 - 8 - 7 - 6 - 5 - 4 - 3 - 2 - 1
[Insert](enter number)
5
Data 5 is already on the list
Head - 9 - 8 - 7 - 6 - 5 - 4 - 3 - 2 - 1
10
Head - 10 - 9 - 8 - 7 - 6 - 5 - 4 - 3 - 2 - 1
25
Head - 25 - 10 - 9 - 8 - 7 - 6 - 5 - 4 - 3 - 2 - 1
Control+D
s1000000{std0ss0}:2
```