プログラミング発展

2023年度2Q　火曜日5~7時限(13:45~16:30)
金曜日5~7時限(13:45~16:30)

工学院　情報通信系

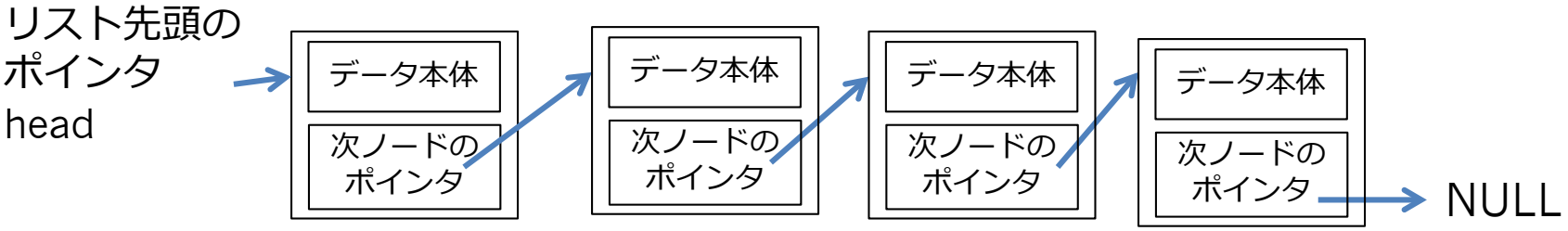
尾形わかは，松本隆太郎，
Chu Van Thiem, Saetia Supat
TA:東海林郷志，千脇彰悟

第8回「データ構造」の続き

1. 前回の課題の解説
2. 連想配列
    1. 二分探索木
    2. ハッシュテーブル

データ構造２

線形リスト (おさらい)



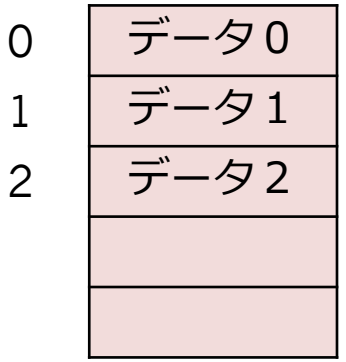

```
typedef struct zipnode_st {
        int zip;                          } データ本体
        char fulladdr[170];
        struct zipnode_st *next;   ← 次ノードへのポインタ
    } zipnode_t;
```

- データの数が決まっていないとき、データサイズが大きいときに、メモリを効率よく利用できる　☺
- 探索：線形探索をするしかない（O(n)）☹

連想配列

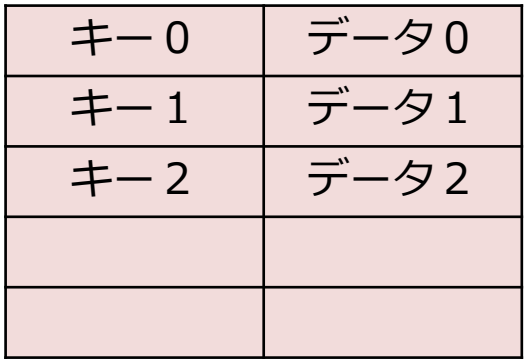
通常の配列：

- インデクス i を指定して，データにアクセスする．
  インデクスは通し番号．
  （data[i] へのアクセスは O(1) ）



連想配列：索引・辞書

- 任意のデータ（文字列など） を「キー」として指定して，キーに対応したデータにアクセスする．
  - 全てのキーは異なると仮定

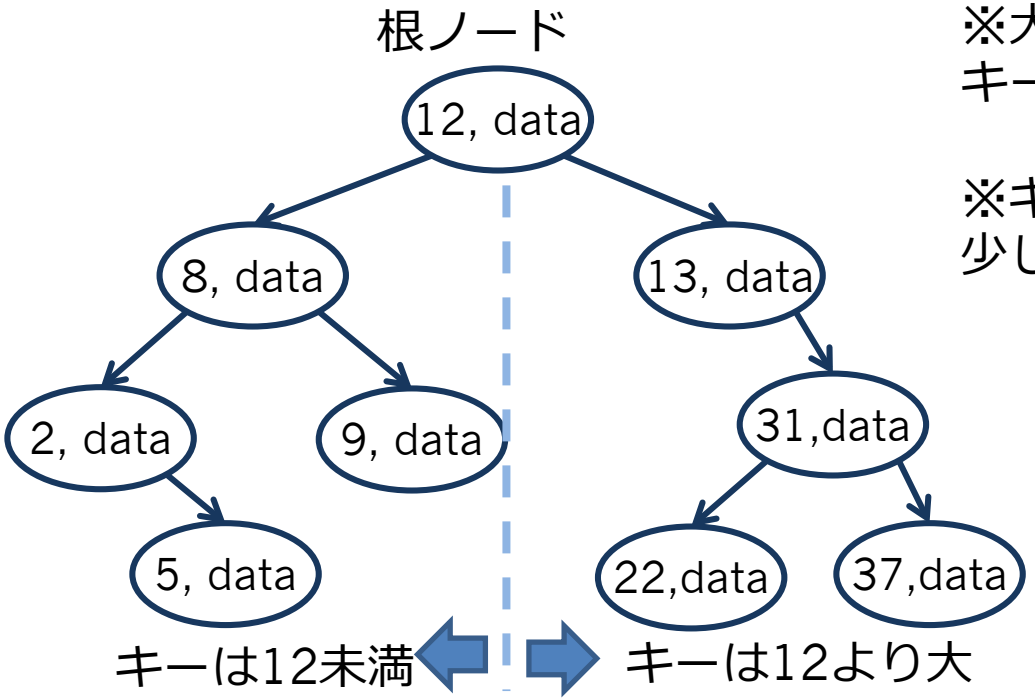


- 実現方法：二分探索木、
  ハッシュテーブルなど．

2分探索木

- 2分木：各ノードが2つ以下の子ノードを持つ.
- 探索木：各ノードが，1つの〈キー，データ〉に対応.

ただし，ノードのキーは
　左の子孫ノード　＜　親ノード　＜　右の子孫ノード
を満たしている.



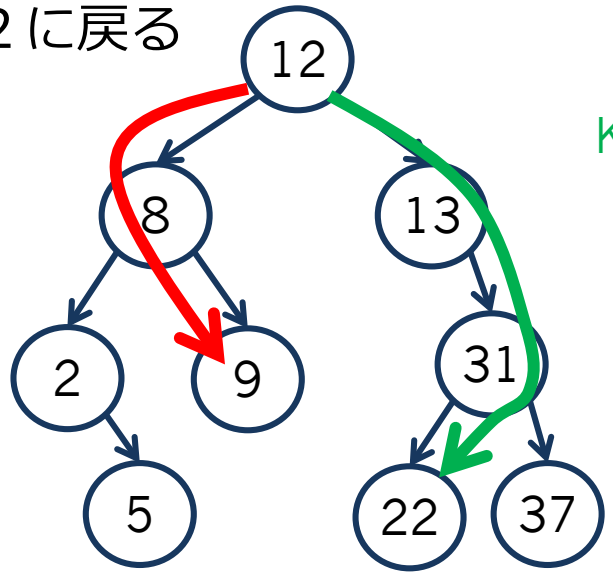
※大小比較できれば，
キーは数値でなくても良い

※キーに重複がある場合も、
少し変更すれば使える。

2分探索木

探索木を使った探索（Kを探索したいとき）

1. 根ノードに着目.
2. 着目ノードのキーとKを比較.
3. 等しい⇒　着目ノードのデータを得る．探索終了.
4. Kのほうが小さい⇒左の子ノードに着目する.
5. Kのほうが大きい⇒右の子ノードに着目する.
6. 着目ノード（右または左の子ノード）が存在しない⇒探索終了
   （対応するデータは存在しない）
7. 着目ノードが存在する⇒2に戻る



K=9の場合,
K < 12　⇒左
K > 8　⇒右
K = 9　⇒終了

K=19の場合,
K > 12　⇒右
K > 13　⇒右
K < 31　⇒左
K < 22　⇒左
⇒ノード無し⇒終了

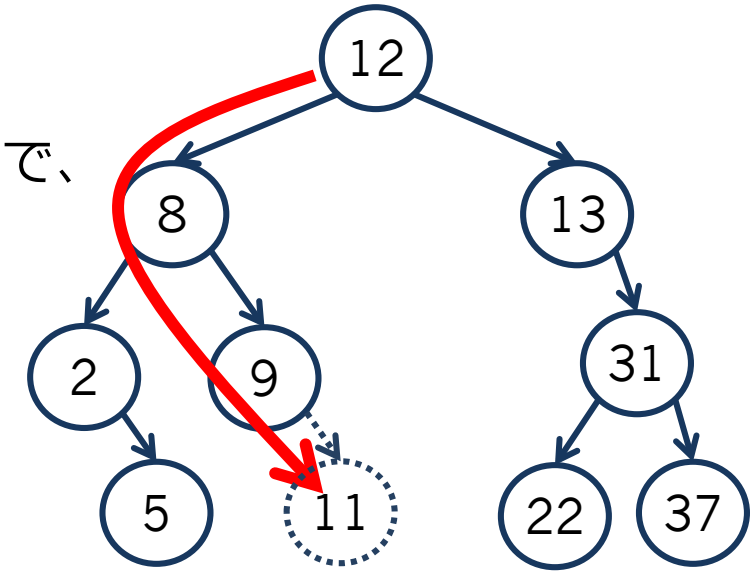
２分探索木（づつき）

データの追加：探索と同様に。

例）キーが K＝11 であるデータを追加する場合、
K=11で探索するのと同様に

　　K < 12　⇒左
　　K > 8　 ⇒右
　　K > 9　⇒右

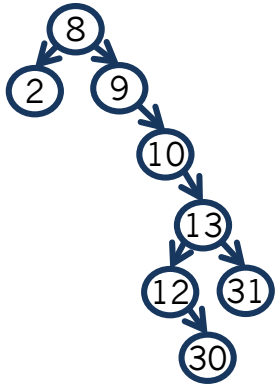
と進み、９の右の子ノードが存在しないので、
そこへ新しいノードを追加する。



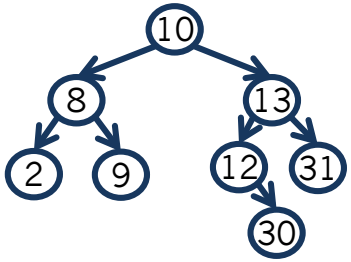
2分探索木の効率

検索時間 ∝ 木の高さ

- データ数Nのとき、木の高さは log_2 (N+1) 以上。
- データの入力順によっては、木の高さ≒データ数N　となってしまい、効率が悪い　**(O(N))**
- 平衡二分探索木：木の高さがO(log_2 N)の探索木。**検索時間は　O(log_2 N)**　となり、二分探索と同じ。



バランスの悪い木は効率が悪い



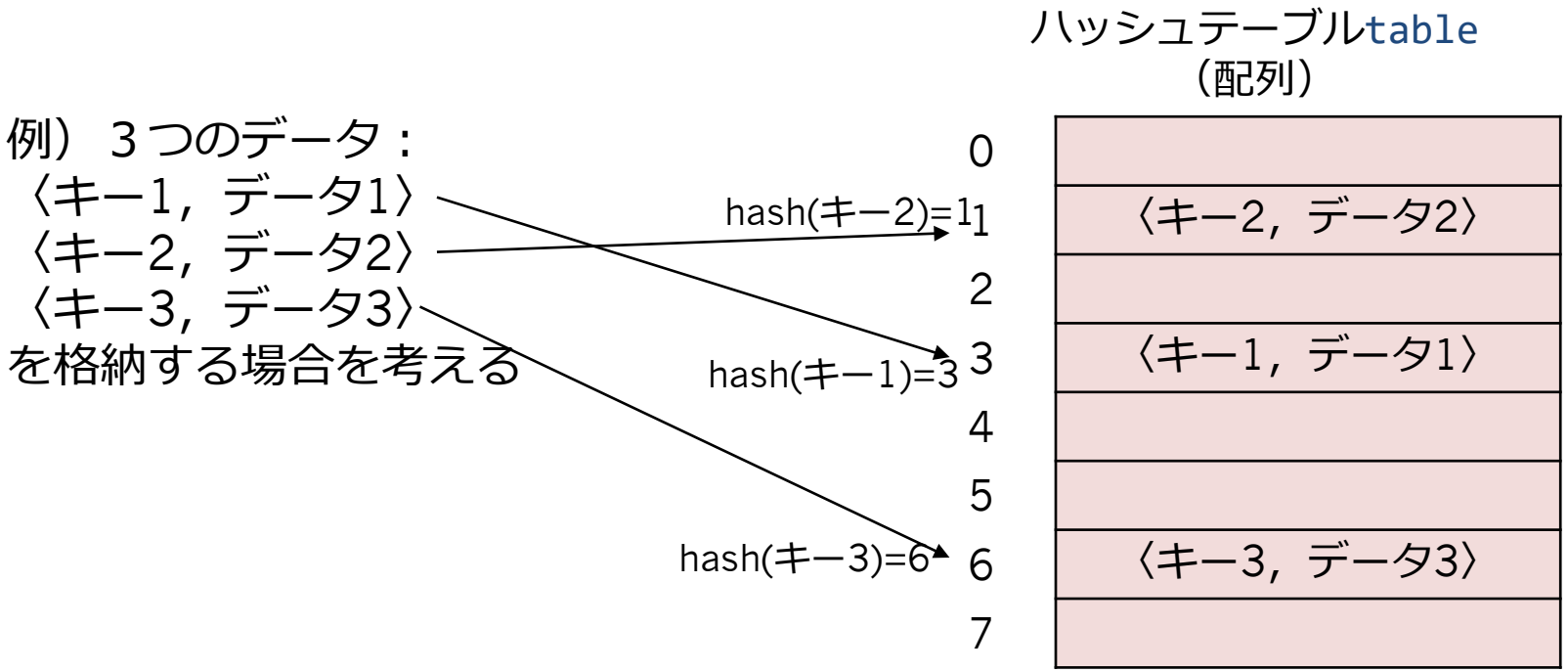
平衡二分探索期はバランスが良く、効率が良い

データの追加・削除：

- 配列を利用した二分探索：挿入位置・削除位置より後のデータを全てずらす必要がある ⇒**O(N)**
- 平衡二分探索木：データの追加・削除も**O(log_2 N)**で可能。（バランスの取り直しの手間を含む）

ハッシュテーブル

- サイズ SIZE の配列table（ハッシュテーブルと呼ぶ）と値域が{0,1,…,SIZE-1}であるハッシュ関数 hash( ) を用意.
- 〈キー，データ〉を array[hash(キー)] に格納.

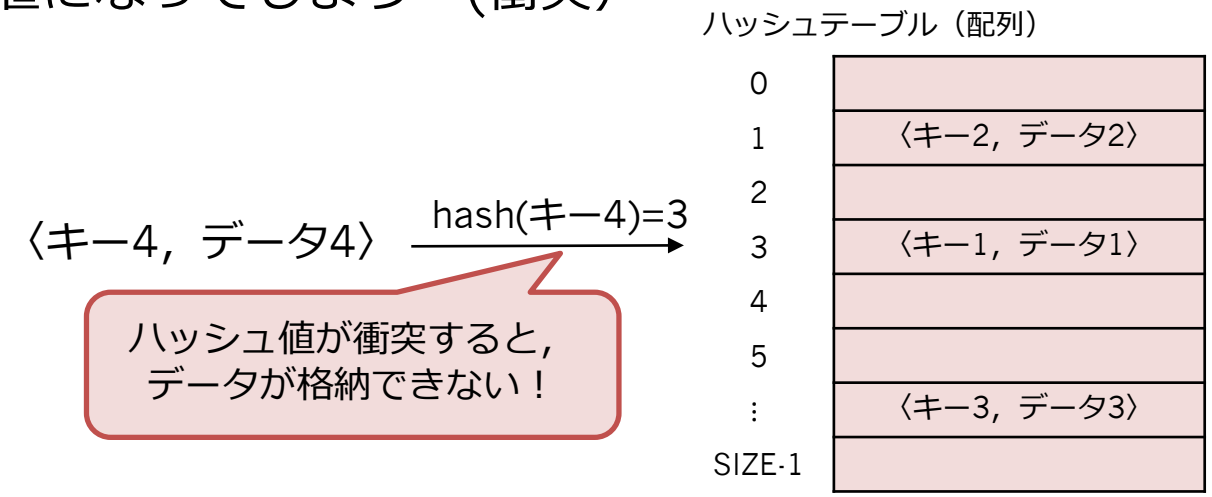


ある文字列strに対して，キー＝str となるデータを探索するとき：
table[hash(str)] にアクセスすればよい．配列のアクセスなので，O(1).

ハッシュテーブル

- データ数N と 配列長 SIZEの関係：

  SIZEがNより十分大きくないと，異なるキーに対して hash(キー) が同じ値になってしまう　(衝突)



  - 対策１：後から格納するデータは，別の空いているところに格納．例えば，table[hash(キー)+1] に入れる．

  - 対策２：配列には線形リストへのポインタを格納し，線形リストへ複数の〈キー，データ〉を格納する．

ハッシュテーブルの利用例（衝突対策なし）

const は、値が変更されないことを示す修飾子。つけなくてもOK

住所addrをキーとして
検索する場合を考える.
キー：addr
データ：zipcode

```
typedef struct zip_st {
        char addr[170];
        int zip;
    } zip_t;
zip_t data[N];       /* original data  */
zip_t table[SIZE]        /* hash table */
const zip_t init={"",0}; /* ハッシュテーブル初期化用 */

unsigned int hash(char *str);         // 適宜定義する

int main(){
/* ハッシュテーブルの初期化 */
    for (i=0; i<SIZE; i++)   table[i] = init;

/* 配列dataにすでに情報が格納されている仮定．これをハッシュテーブルに入れる */
    for (i=0; i<N; i++) {
        j = hash(data[i].addr);
        if (strcmp(table[j].addr, init.addr)==0) // 空ならば格納
            table[j]=data[i];
        else return 1;                  // 衝突が起これば残念.
    }
}
```

住所xxx での検索は，table[hash(xxx)].addr と xxx が等しいかチェック

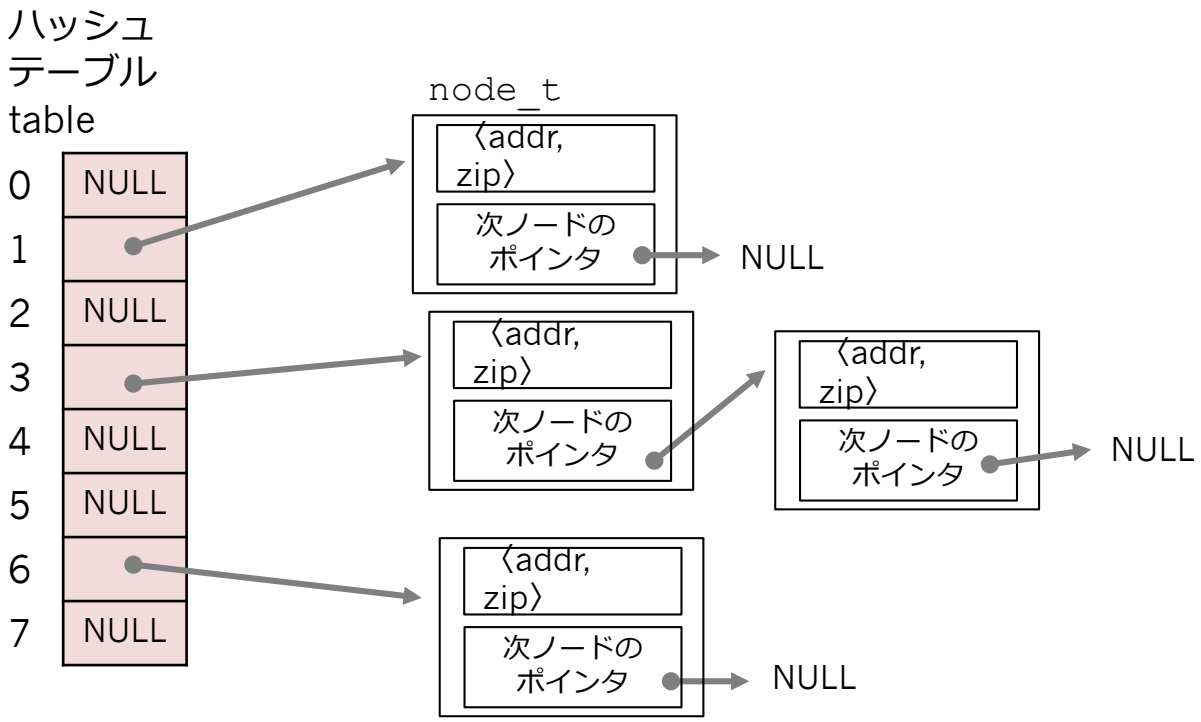
ハッシュテーブルの利用例（衝突対策２）

```
typedef struct node_st {
        char addr[170];
        int zip;
        struct zipnode_st *next;
    } node_t;

node_t *table[SIZE];   /* hash table */
```

データ本体.

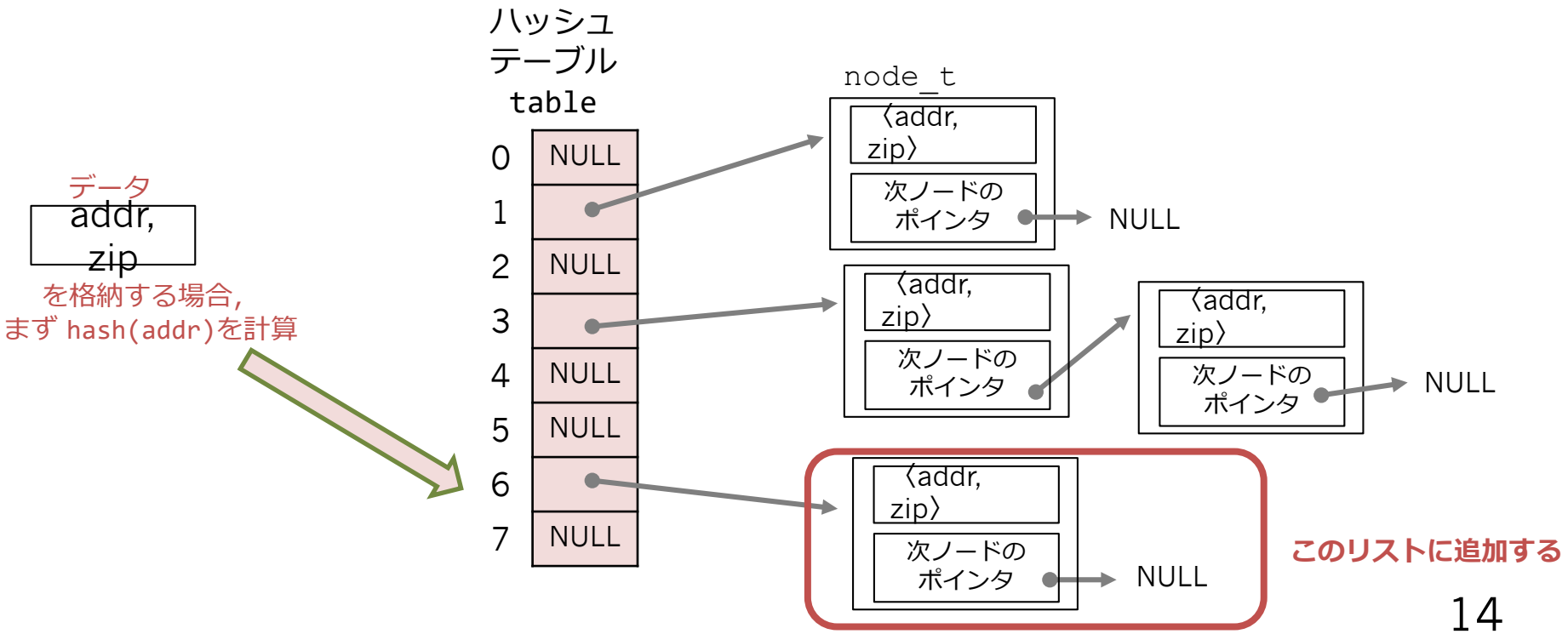
ハッシュテーブルには，ノードへのポインタを格納する。
始めに、NULLで初期化しよう



ハッシュテーブルの利用例（衝突対策２）

addrをキーとする場合，

- 格納：table[hash(addr)]をheadとするリストにノードを追加
- 検索：table[hash(addr)]をheadとするリスト内を検索



Ex8-1

ハッシュテーブルを使ってみよう.
名前、性別、年齢の入ったファイル “personal_data.txt” からデータを読み込み，名前で検索するプログラムを作成しなさい．仕様は以下の通り.

- ファイルから１行ずつ読み込み、名前をキーとしてハッシュテーブルを作成する．テーブルサイズはSIZE=128とする.
- 衝突処理はせず、もしハッシュテーブル作成時に衝突が起こったら、その時点で“collision”と出力して終了するようにする.
- 標準入力から名前を読み込み，ハッシュテーブルで検索して，名前，性別，年齢を出力する.
- 名前として `exit` が入力されるまで、無限に入力を受け付けるようにすること.
- ハッシュ関数は次ページのものを用いてよい.

提出物：
・ソースファイル（ex8_1.c ）

personal_data.txtの例

```
Alice 2 17
Bob 1 15
Carol 2 12
R2D2 0 999
```

Ex8-1　（つづき）

※名前は，空白を含まない文字列と仮定してよい.

※ハッシュ関数は，以下を使ってよい：（へなちょこですが）

```
unsigned int hash(char *str) {
    int hashval = 0;
    while (*str != '¥0') {
        hashval = hashval + *str;
        str++;
    }
    return (hashval % SIZE);
}
```

※衝突を人為的に起こしたければ、
personal_data.txtの任意の行を複製すればよい。

（発展課題）　性別を　1,2ではなく，
male, female で表示させよう.
1,2以外は other などとする.

実行例

```
Name? > Bob
(Bob 1 15)

Name? > Alice
(Alice 2 17)

Name? > Bill
No data

Name? > R2D2
(R2D2 0 999)

Name? > exit
```

ファイルから「あるだけ」データを読み込む

- fscanf で読み込む場合、ファイルの最後まで来るとfscanfが 戻り値としてEOFを返すので、
  while (fscanf(fp,"・・・",・・・)!=EOF)
  のようにして読み込むこともできるが,
  fscanfは読み込みエラーが起きたときにもEOFを返すので,
  エラーとファイル終了を区別できない.
  fgets と sscanf の組み合わせがお勧め.

※ fscanfはセキュリティの問題も持っているので,
避けた方が良い.

ファイルから１行分まとめて読み込み
char buff[] へ格納.
ファイルが終わったらNULLが返る.

```
while (fgets(buff, sizeof(buff), fp) != NULL){
    n=sscanf(buff, "%s %d %d", data.name, &data.gender, &data.age);
    if (n != 3) {
        printf("Input error¥n");
        return 1;
    }
    << 必要な処理 >>
}
```

buffから scanfと同様に変数を読み込む.
戻り値は読み込めた変数の個数.

Ex8-2

Ex7_2_4.c を改変し，ハッシュテーブルと線形リストを用いて，郵便番号検索を行うプログラムを作りなさい。仕様は以下のとおり。

1. まず、tokyo_all_dat.txt からデータを読み込み、各データを、住所をキーとしてハッシュテーブルへ格納する。
   - ハッシュテーブルのサイズはSIZE=1024とし、ハッシュテーブルには線形リストへのポインタを格納すること。関数 add_node はそのまま使えるはず。
   - 関数read_from_csvは変更する必要がある.
2. 関数print_n_nodeを使って，hash(fulladdr)が0であるデータ、1であるデータ、2であるデータを順に出力する.
3. 次に，住所を入力として受付け、入力された住所をハッシュテーブルを用いて検索し、対応する郵便番号を出力する．住所として"exit"が入力されるまで検索を繰り返す。
   - 関数search_nodeは変更する必要がある.
   - プログラム終了時には、すべての線形リストのメモリを開放することを忘れずに。

Ex8-2

★提出物：ソースファイル（ ex8_2.c ）

出力例

```
Read 3809 data
data with hash(fulladdr)=0:
1980211 : TOKYO TO NISHITAMA GUN OKUTAMA MACHI NIPPARA
 . . .
data with hash(fulladdr)=1:
1940021 : TOKYO TO MACHIDA SHI NAKAMACHI
 . . .
data with hash(fulladdr)=2:
1650032 : TOKYO TO NAKANO KU SAGINOMIYA
 . . .

full address ? > TOKYO TO MACHIDA SHI NAKAMACHI
1940021

full address ? > DOKOKA
no dat

full address ? > exit
Bye!
```

Ex8-3

Ex8-2の課題のプログラムを変更して，住所での検索と郵便番号での検索の両方ができるようにしたい.
２セットの〈ハッシュテーブル＋線形リスト群〉を用意すれば可能だが，もっとメモリ効率を良くするには，どのようにデータを保管すればよいだろうか？
（ヒント：データ本体をリストに格納するのは避けよう．）

提出物：解答をpdfファイルで提出。
実装する必要はありません。（ ex8_3.pdf ）

Ex8-4

２分探索木を実現するには，どのような構造体を使えばよいだろうか？（キーを最大30文字の文字列、データをint型の値とする。）線形リストで用いた構造体を参考に考えてみること.

提出物：説明をpdfファイルで提出。
実装する必要はありません。（ ex8_4.pdf ）

今日の提出物 まとめ

- Ex8-1：ソースファイル（ ex8_1.c ）
- Ex8-2：ソースファイル（ ex8_2.c ）
- Ex8-3：説明を書いたファイル（ ex8_3.pdf ）
- Ex8-4：説明を書いたファイル（ ex8_4.pdf ）

注意！

- ハッシュテーブルは適切に初期化し、「何も対応するデータがない」ことを明確にしておくこと。
- 特に、ex8-2では、ハッシュテーブルにはポインタが保存されるため、初期化しないとメモリの不適切利用につながる
- ex8-2では、多数の線形リストを使うので、 mallocで確保したメモリ領域をすべてfreeしてからプログラムを終了させること。

コピペレポートについて

プログラムや考察などが他の提出者と重複している場合、不正とみなして減点および問い合わせをすることがあります