プログラミング基礎演習
第10回

金（5限）

長谷川禎彦

連絡事項

- 提出者一覧はホームページ
- 前回課題のプログラム例もホームページ
  - 課題2は課題3のサブセットなので，載せていない

前回課題の注意点

```
int* f() {
    int a[10];
    return a;
}
```

局所変数は関数が実行中の間だけ存在する. 関数を出ると値がなくなる.

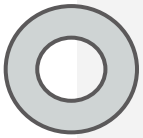
```
int* f() {
    int *a;
    a = (int*) malloc(sizeof(int)*10);
    return a;
}
```

mallocで確保した変数はfreeされるまで存在する. 関数を出ても値はなくならない.

前回課題の注意点

```
struct monomial* construct_monomial(int c,int n) {
    struct monomial m = {c,n};
    return &m;
}
```

これだと，関数を出るとなくなってしまう



```
struct monomial* construct_monomial(int c,int n) {
    struct monomial *m;
    m = (struct monomial*)malloc(sizeof(struct monomial));
    m -> c = c;
    m -> n = n;
    return m;
}
```

mallocで確保すれば，関数を出てもなくなることはない

双方向連結リストのメイン部分

```
while (fscanf(fp, "%[^,],%d,%lf,%lf\n", tmp_name, &tmp_age, &tmp_height, &tmp_weight) !=
EOF) {
     if (i==0) {
          head = (struct member*) malloc(sizeof(struct member));
          head->next = NULL;
          head->previous = NULL;
          current = head;
     } else {
          current->next = (struct member*) malloc(sizeof(struct member));
          current->next->previous = current;
          current = current->next;
          current->next = NULL;
     }
     strcpy(current->name,tmp_name);
     current->age = tmp_age;
     current->height = tmp_height;
     current->weight = tmp_weight;
     i++;
  }
```

連結リストに必要なのは先頭 (head)と，今の時点で一番最後の要素 (current)へのポインタ

新しく追加するときは，current->nextにアドレス設定する

今日の学習事項

- ハッシュ
- ハッシュテーブル

ハッシュ

- ハッシュ関数とは，データに対して整数値を計算する関数である
  - 2つのファイルが同一か確認する場合はそれぞれのハッシュ値が一致しているかどうかを確認すれば良い．ハッシュ値が異なればデータは必ず違う．同じであれば，同じデータである確率が非常に高い．
  - しかし，異なるデータであっても同じハッシュを返す場合がある（ハッシュの衝突）
- 良いハッシュは衝突が最小限であること
  - できるだけ一様に分布するようなハッシュ

簡単なハッシュの例

```
int hash(int x, int y) {
   return x + y;
}
```

これだと，(1,5)と(5,1)で同じハッシュになってしまう．データがある程度小さいなら以下の方が衝突が少なくなる．

```
int hash(int x, int y) {
   return 1000*x + y;
}
```

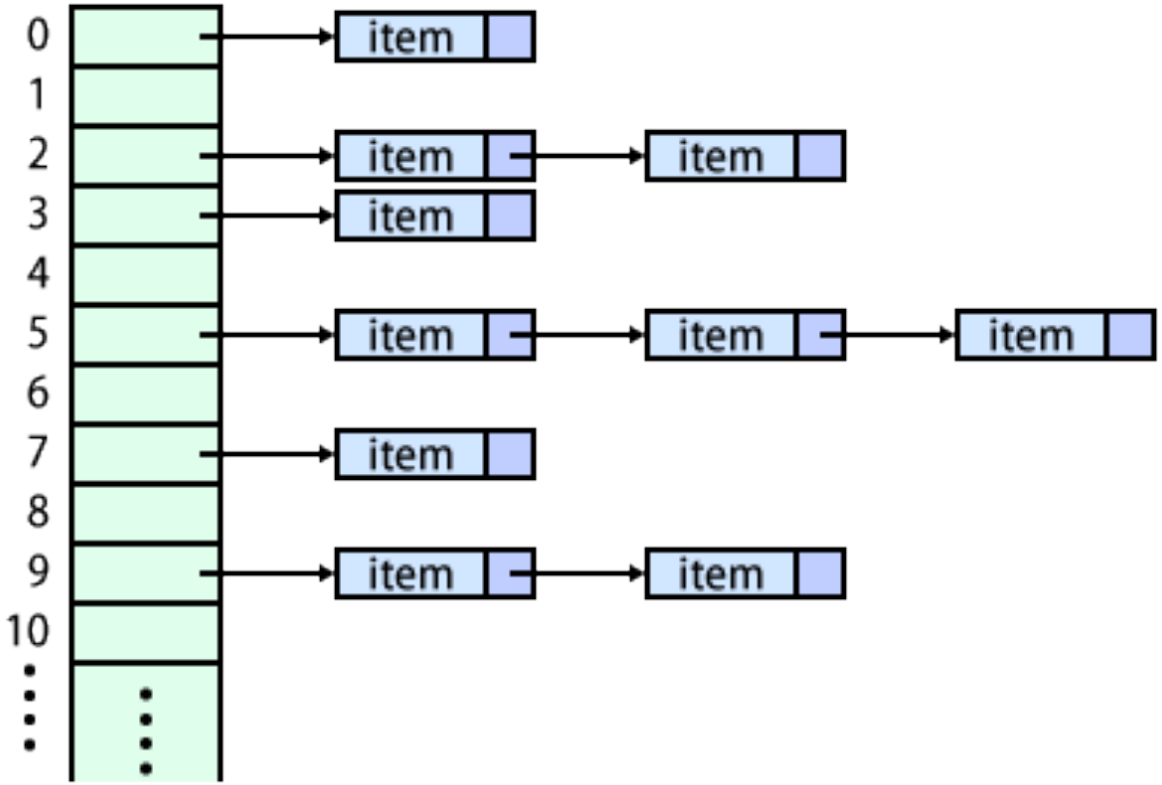
ハッシュテーブル

- 要素数 N の配列を用意する（ルート配列）．ルート配列の各要素は，リンクリスト（便宜上エントリリストと呼ぶ）である（またはリストへのポインタ）．
- エントリを格納する場合，エントリの「キー」に対してハッシュ値を計算する．ハッシュ値を n とするとき，ルート配列上の n 番目のエントリリストにこのエントリを格納する．ハッシュ値の衝突が発生したとき，それらのエントリは同一のエントリリストに格納される．

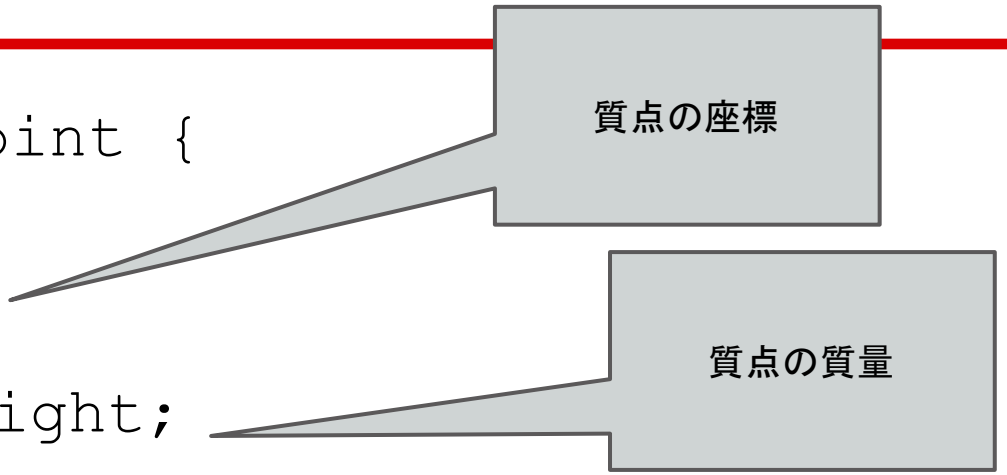
ハッシュテーブル

- あるキーをもつエントリを検索する場合，ハッシュ値を計算し，ハッシュ値の場所のエントリリストから探す．
- リストになっているので，目的のエントリが見つかるまでリストを走査する

ハッシュテーブル


0
1
2
3
4
5
6
7
8
9
10
item
item
item
item
item
item
item
item
item
item

ハッシュテーブルへの格納
キーがpointの場合の例




```
struct point {
    int x;
    int y;
    int weight;
    struct point *next;
};
int hash(struct point p) {
    return (p.x + p.y) % 10;
} // ハッシュ値は座標の単純な和
// ルート配列は10個. ポインタの配列
struct point *root[10];
```

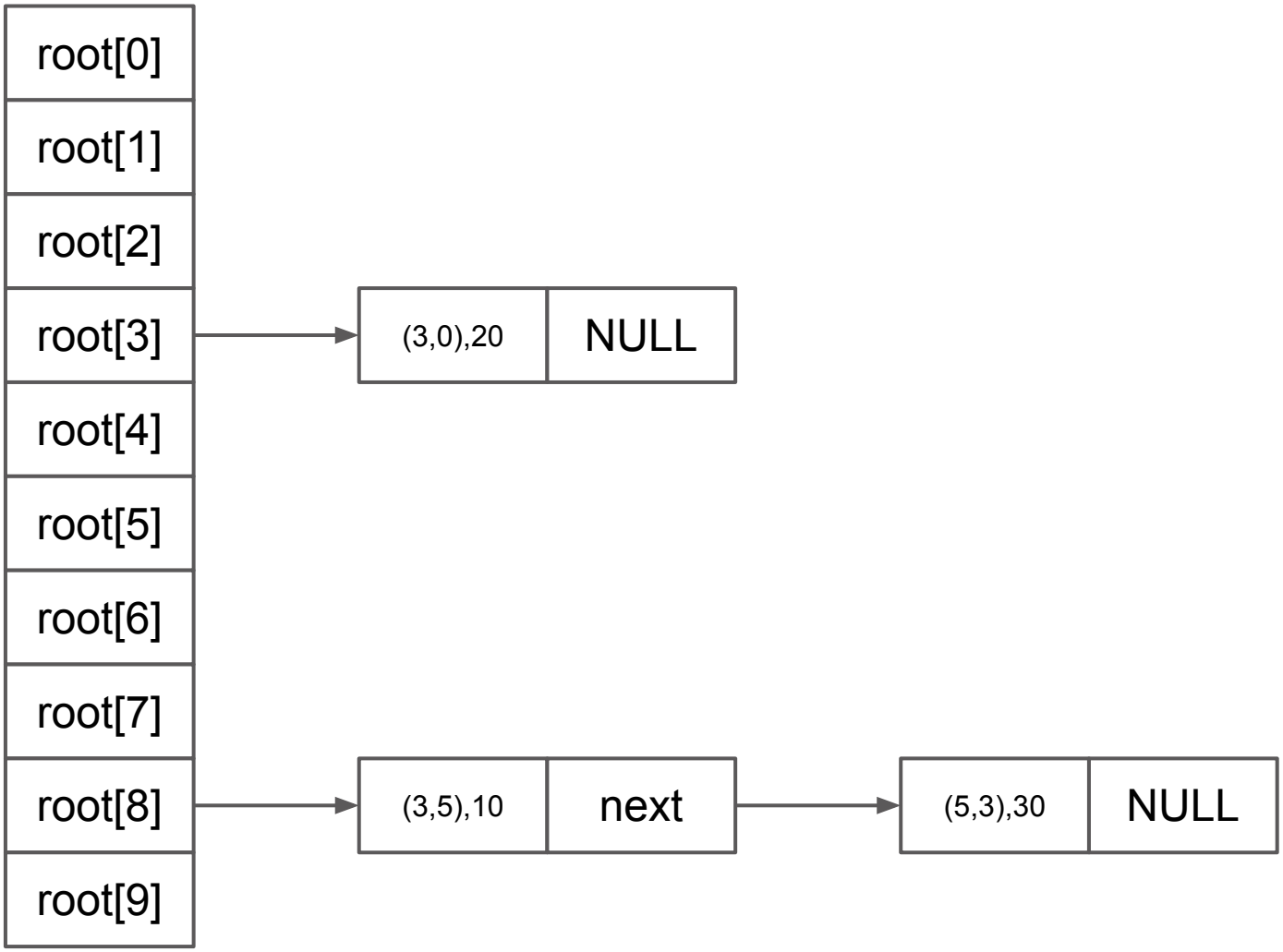
ハッシュテーブルへの格納
キーがpointの場合の例

(x,y,weight) = (3,5,10)を格納する場合，hashが8なので，root[8]にアドレスを格納する.

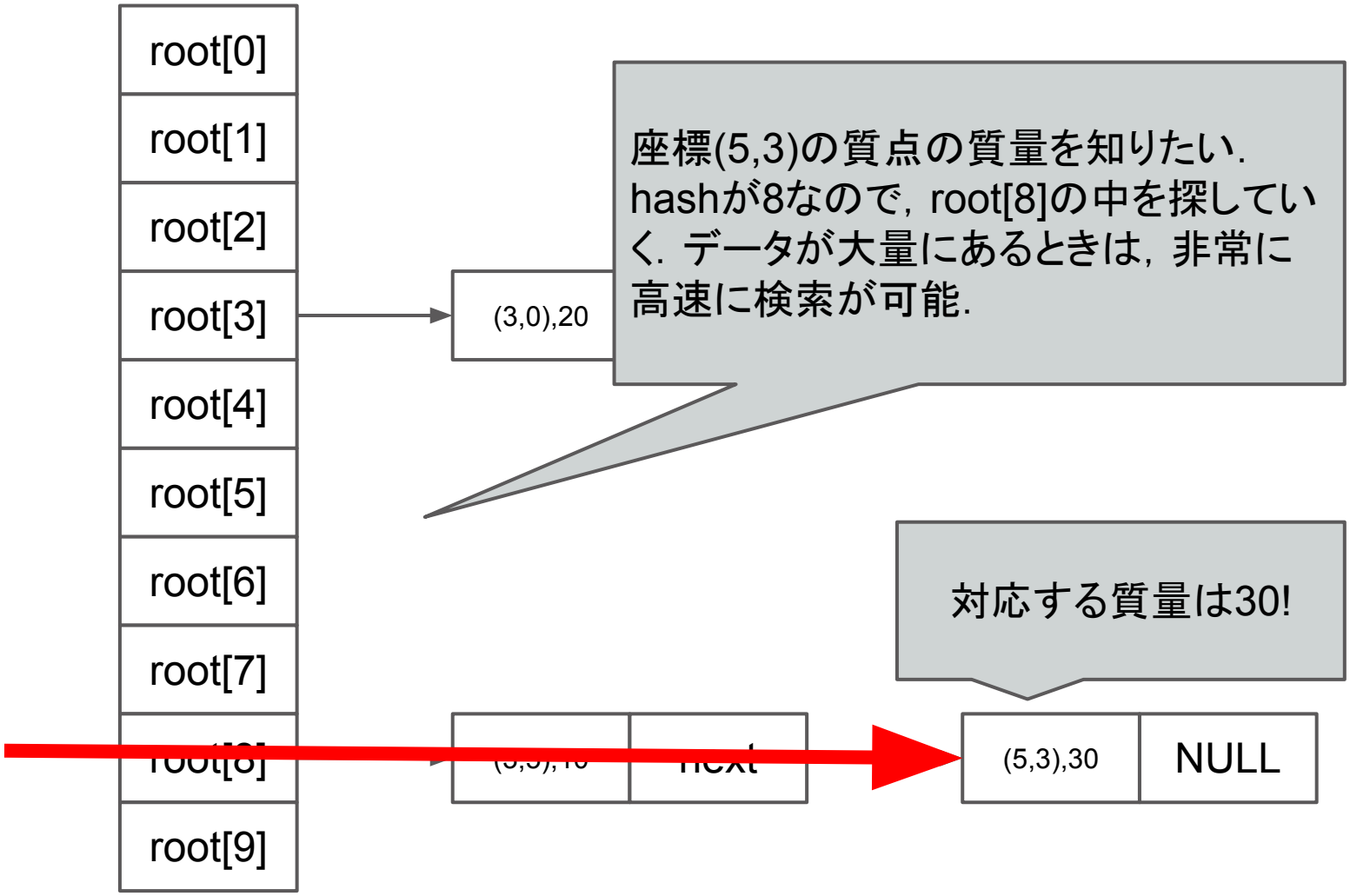
次に(x,y,weight) = (3,0,20)を格納する場合，hashが3なので，root[3]にアドレスを格納する.

次に(x,y,weight) = (5,3,30)を格納する場合，hashが8なのでroot[8]にアドレスを格納したいが，既に存在するので，root[8] -> nextにアドレスを格納する.

ハッシュテーブルへの格納
キーがpointの場合の例



ハッシュテーブルへの格納
キーがpointの場合の例



ハッシュテーブルへの格納
キーがpointの場合の例

つまり，

- ルート配列にまだなにも無いときはそのまま格納
- 重複があればnextに格納．nextもあるときはnext->nextに格納．以下同じ

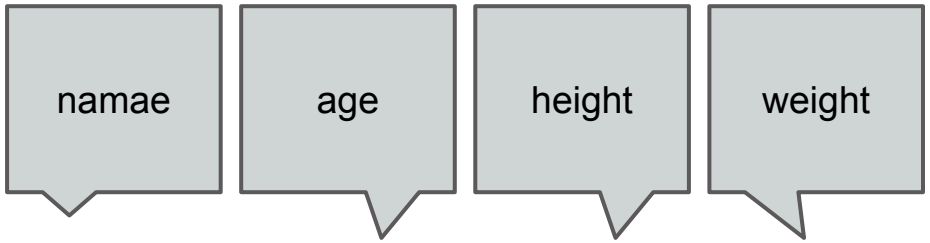
を繰り返す

- 座標から，対応する質点の質量を探すときは，ハッシュを計算して，ルート配列の対応する場所を探す

課題1

- 10000人分の名簿ファイル（namelist.txt）を読み込み，ハッシュテーブルに格納せよ（名簿ファイルのフォーマットは以前と同じ）．ハッシュのキー（ハッシュを計算する対象）は名前で，ルート配列の大きさは1000程度とする（意図的に重複を作る）．なお，名前に重複はない．
- 標準入力から入力された名前に対して，対応するデータを出力せよ

データフォーマット（前と同じ）




```
Dbrenlhsij,11,162.2,55.9
Ijnpwthy,14,163.8,62.6
Csaztvsy,30,176.2,57.1
Pgqfcwlx,22,161.2,57.3
Mgjwpmdxr,19,162.,65.2
...
```

文字列に対する簡単なハッシュの例

```
#define HASHSIZE 100
int hash(char *s){
  unsigned int val;
  for (val = 0; *s != '\0' ; s++) {
     val = *s + (31 * val);
  }
  return (int)(val % HASHSIZE);
}
```

ハッシュ値は0-99までとする

unsigned だとオーバーフローしても大丈夫

ハッシュテーブルの実装例

```
struct member {
   char name[20];
   int age;
   double height;
   double weight;
   struct member *next;
};
```

ハッシュテーブルの実装例

ルート配列の例

```
#define HASHSIZE 1000
struct member *root_array[HASHSIZE];
```

課題提出方法

- 課題xの答えを`kadai0x.c`ファイルに書く.
  - xは1
- **締め切りは火曜日の23:59**
- **ファイルをzipファイルにまとめる**
  - **ファイル名:学籍番号.zip**
  - 例:学籍番号が341234の場合，341234.zipとする.
- 学籍番号.zipファイルのみを提出する
  - 提出は http://goo.gl/hXsfLl
  - フォームの「課題」から「小課題」を選択