# C プログラミング

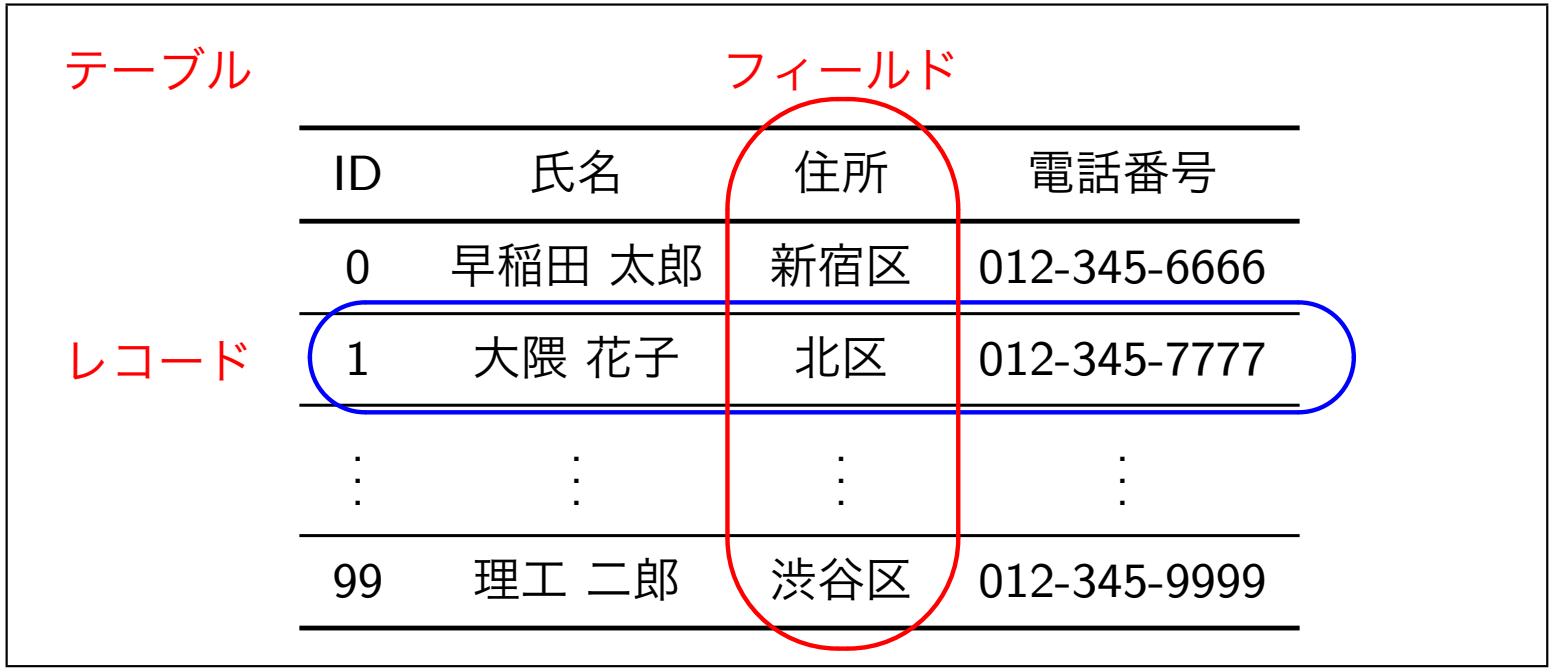
— 探索—

早稲田大学

# 本日の目標

- データ形式
- 探索
  - 逐次探索法
  - 二分探索法

# データ形式・操作




- データには様々な形式があるが，概念上は表形式でデータを扱うことが多い．
- 上図のような住所録を例にとると，表全体をテーブル，一人分のデータをレコード，ID や氏名などの個々の属性をフィールドまたはカラムと呼ぶ．
- 表形式の場合は，全てのレコードが同じフィールドを持つが，他のデータ形式では，あるレコードは住所フィールドはあるが，電話番号フィールドは持たない，といったことが許されるものもある．
- テーブルに対して行う主な操作は，レコードの追加挿入，修正，削除，探索，ソートなどがある．

# データ形式・操作

- データがどのようなデータ構造で格納されているかによって，レコードの追加挿入，削除，探索，ソート方法などが異なる．方法によって計算量も $\mathcal{O}(N)$，$\mathcal{O}(\log N)$，$\mathcal{O}(1)$ などと異なる．
  - 配列，線形リスト，双方向リスト，スタック，キュー，ハッシュなど．
- レコードの内容や順序に関しても色々な制約をかけることがある．
  - 一意性: テーブル内に同じレコードは [存在できない / 存在できる]．
  - 順序: レコードは常に特定の順番に [ソートされている / ソートされているとは限らない]．
- 探索条件によっても，探索方法が異なる．
  - 探すレコードはテーブル内に [必ず存在する / 存在するかどうか分からない]．
  - 探すレコードは存在するとすれば，[唯一である / 複数かもしれない]．
  - 探すレコードが複数ある場合，[どれか一つを探せればよい / 全部探し出したい]．

# 探索

- テーブル内の特定のレコードを見付けること.
- 今回扱う基本的な探索では，ある特定のフィールドについて，ある特定の値を指定することで，指定された値を持つレコードを探し出すことを行う.

  例: 「氏名が “早稲田 太郎” のレコードを探す.」

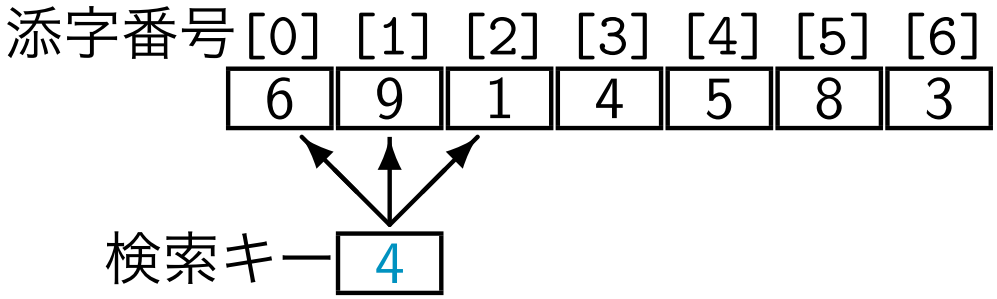
  この場合，探索対象となるフィールドをキー，指定された値を検索キーと呼ぶ.

# 探索

- 代表的な探索アルゴリズム
  - 逐次探索法
  - 番兵法
  - 二分探索法（バイナリサーチ）
  - ハッシュ法

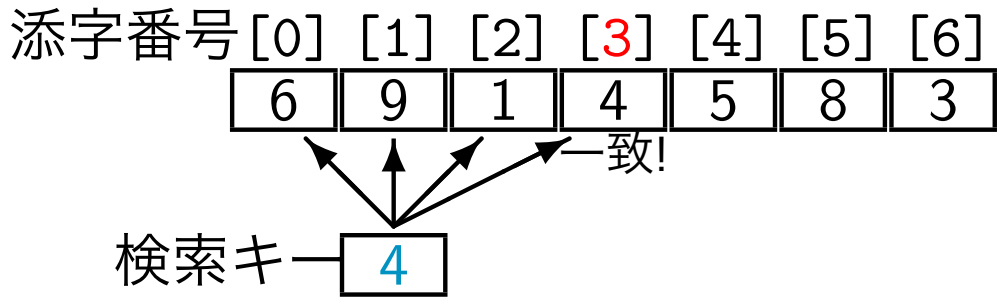
# 逐次探索法

- 簡単のため，フィールドが double 型一つだけのテーブルを考え，データ構造も単純な配列を用いることにする．
- 指定された値と一致するレコードを逐次探索して，その添字番号を表示せよ．
  - 一致するレコードが複数ある場合は，その中で最も小さい添字番号を表示することにする．
  - 一致するレコードが無い場合は，その旨表示することにする．



# 逐次探索法

- 先頭から順に値が一致するか調べて，見つかったらその添字番号を表示すればよい.




- 最良で 1 回の検査（目的のデータが先頭にあった場合），最悪で全レコードの検査（目的のデータが末尾にあった場合）が必要.
- $N$ 個のデータに対して平均約 $\frac{N}{2}$ 回の比較操作が必要．つまり $\mathcal{O}(N)$.

# 逐次探索法

- 次の構文はよく用いられるので，理解して覚えること．

```
for (i = 0; i < N; i++) {
  if (Data[i] == key) {
    break;     /* 一致したら中断 */
  }
}
if (i < N) {  /* for 文が中断された場合 */
  printf("found. index: %d\n", i);
} else {      /* for 文が最後まで回った場合 */
  printf("not found.\n");
}
```

- for 文の後の if 文で，for 文の条件文と同じ条件（i<N）を指定することで，for 文が中断されたか否かが分かる．

# 演習問題

## ファイル名: seqsearch.c

double 型配列 Data に 5, 3, 1, 4, 2 がこの順番で格納されているとする. このデータを逐次探索法を用いて指定された値の添字番号を表示するプログラムを作成せよ.

- 表示は各要素の比較操作が分かるようにすること.

```
key? 4
5, 3, 1, 4, 2,
Data[0]=5 is not equal to 4.
Data[1]=3 is not equal to 4.
Data[2]=1 is not equal to 4.
Data[3]=4 is equal to 4!
found. index: 3
```

# 解答例

```
#include <stdio.h>
void PrintData(double *Data, int N){
  int i;
  for (i = 0; i < N; ++i) {
    printf("%.0f, ",Data[i]);
  }
  printf("\n");
}

int main(void){
  double Data[] ={
    5, 3, 1, 4, 2
  };
  int i;
  int N = sizeof(Data)/sizeof(Data[0]);
  double key;
  printf("key? ");
  scanf("%lf",&key);

  PrintData(Data,N);

  // 逐次探索を開始する
  for (i = 0; i < N; i++){
    if (Data[i]==key){
      printf(...);
      break;
    }
    printf(...);
  }
  if (i < N){
    printf("found. index: %d\n",i);
  } else {
    printf("not found.\n");
  }
  return 0;
}
```

# 解答例

```
#include <stdio.h>
void PrintData(double *Data, int N){
  int i;
  for (i = 0; i < N; ++i) {
    printf("%.0f, ",Data[i]);
  }
  printf("\n");
}

int main(void){
  double Data[] ={
    5, 3, 1, 4, 2
  };
  int i;
  int N = sizeof(Data)/sizeof(Data[0]);
  double key;
  printf("key? ");
  scanf("%lf",&key);

  PrintData(Data,N);
  // 逐次探索を開始する
  for (i = 0; i < N; i++){
    if (Data[i]==key){
      printf(...);
      break;
    }
    printf(...);
  }
  if (i < N){
    printf("found. index: %d\n",i);
  } else {
    printf("not found.\n");
  }
  return 0;
}
```

# 解答例

```
#include <stdio.h>
void PrintData(double *Data, int N){
  int i;
  for (i = 0; i < N; ++i) {
    printf("%.0f, ",Data[i]);
  }
  printf("\n");
}

int main(void){
  double Data[] ={
    5, 3, 1, 4, 2
  };
  int i;
  int N = sizeof(Data)/sizeof(Data[0]);
  double key;
  printf("key? ");
  scanf("%lf",&key);

  PrintData(Data,N);
  // 逐次探索を開始する
  for (i = 0; i < N; i++){
    if (Data[i]==key){
      printf(...);
      break;
    }
    printf(...);
  }
  if (i < N){
    printf("found. index: %d\n",i);
  } else {
    printf("not found.\n");
  }
  return 0;
}
```

# 解答例

```
#include <stdio.h>
void PrintData(double *Data, int N){
  int i;
  for (i = 0; i < N; ++i) {
    printf("%.0f, ",Data[i]);
  }
  printf("\n");
}

int main(void){
  double Data[] ={
    5, 3, 1, 4, 2
  };
  int i;
  int N = sizeof(Data)/sizeof(Data[0]);
  double key;
  printf("key? ");
  scanf("%lf",&key);

  PrintData(Data,N);

  // 逐次探索を開始する
  for (i = 0; i < N; i++){
    if (Data[i]==key){
      printf(...);
      break;
    }
    printf(...);
  }
  if (i < N){
    printf("found. index: %d\n",i);
  } else {
    printf("not found.\n");
  }
  return 0;
}
```

# 二分探索法

- あらかじめ整列（ソート）されたレコードの中から，探索範囲を半分，半分と狭めて探索する方法．具体的には，検索キーを配列の中央のキーと比較して，目的のレコードが中央より前にあるか後にあるかを判断する．この一回の検査で，探索すべき範囲を半分に狭めることができる．
- $N$ 個のデータに対して約 $\log_2 N$ 回の比較操作が必要（つまり $\mathcal{O}(\log N)$）．
  100 万件のデータからでも約 20 回の検査で見つけることができる．

## 二分探索法のアルゴリズム

昇順にソートされている $N$ 個のデータ $x_0, x_1, ..., x_{N-1}$ $(x_0 \leq x_1 \leq ... \leq x_{N-1})$ の中から，与えられた検索キー $x^*$ に対して $x_i = x^*$ となるような添字番号 $i$ を答える．

Step 1. $l = 0$，$r = N-1$ とする．

Step 2. $l < r$ でないならば Step 4 に移る．

Step 3. $m = (l + r)/2$（小数点以下切捨て）を計算する．

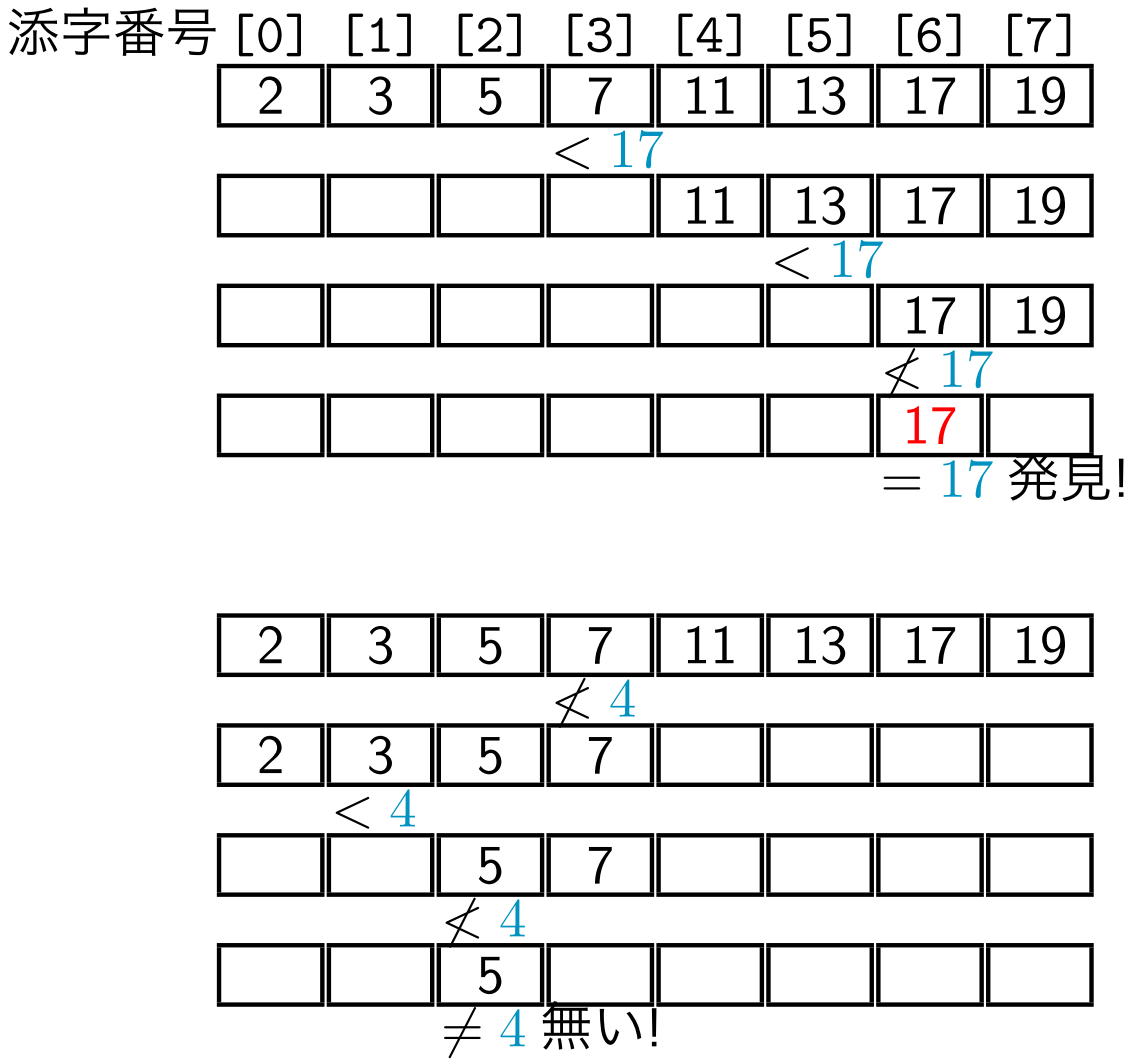
- $x_m < x^*$ ならば，$l \leftarrow m + 1$ とする．
- そうでないならば，$r \leftarrow m$ とする．

Step 2 に戻る．

Step 4. $x_l = x^*$ ならば，$l$ を答える．
そうでないならば，一致するものはない．

# 二分探索法

- 要素数 $N = 8$ での二分探索例
- 中央の要素が検索キー未満かどうかを比較．そうであれば，中央より右半分を，そうでなければ，中央より左半分（中央を含む）を調べる．
- 残り 1 個になったら，検索キーと等しいかどうかを比較．等しければ発見できたことに，等しくなければ該当レコードは無かったことになる．
- クイックソートと異なり，再帰関数にしなくとも実装できることに注意．



# 本日のまとめ

- データ形式
- 探索
  - 逐次探索法
  - 二分探索法