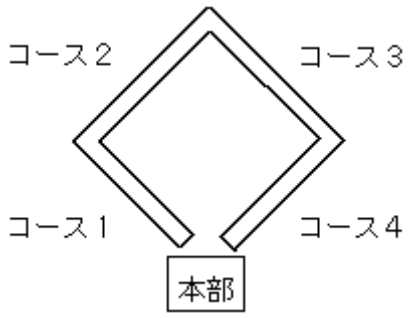
（3）情報処理の定式化とデータ管理

設定問題：この競技大会を行なう会場として、図のような 400 メートルのコースが使えることになった。



図のコース１～４はそれぞれ長さは 100m の直線である。100m走、100mハードル走はコース１～４のどこででも行なえる。200m走、200mハードル走はコース 1 とコース 2、コース 2 とコース 3 などのように、コース 2 つをつなげて行なう。ただしコース 1 とコース 4 は間に競技本部が設置されるので、つなげて使うことはできない。 300m走も同様に、例えばコース 2、3、4 のように 3 つのコースを使って行なうとする。

ここで以下の競技を行なうことを考える。

100m走A
100m走B
200m走
300m走
100mハードル走
200mハードル走

それぞれにかかる時間は次のように予定されている。

- **100m・200m・300m走はそれぞれ 30 分**
- **100m・200mハードル走は準備・後始末に時間がかかるのでそれぞれ 45 分と 60 分**

これをまとめると下のようになる。

100m走A	30 分
100m走B	30 分
200m走	30 分
300m走	30 分
100mハードル走	45 分
200mハードル走	60 分

それぞれの競技をどのコースでどのようなスケジュールで行なったらいいだろうか。

演習：それぞれの競技をどのコースでどのようなスケジュールで行なったらいいか考えてプランを作ってみよ。またそのスケジュールでは始めてから終るまで時間がどれだけ必要か考えてみよ。

いろいろなものが考えられるが、例えば方法１のようにすると、全体が終るのに２時間かかる。

```
//陸上競技のコース割り当て 方法１
//　コース１～４の４つのコースに各競技を割り当てる
//
コース１で 100mの競技全部を順次行なう。
コース２～４をつなげて 200m、300mの競技全部を順次行なう。
```

この競技場は使う時間が長いとそのぶん利用料を多く支払わなければならないので、始めてから終えるまでの時間はできるだけ短くしたい。競技の割当を工夫して、終る時間をもっと早くすることを考えてみよう。

ここで以下の方法に基づいて割り当てることを考えよう。（方法２－第１段階）

```
//陸上競技のコース割り当て 方法２－第１段階
//　コース１～４の４つのコースに各競技を割り当てる
//
コースが空いたら
　　空いたコースで行なうことのできる中で一番距離の長い競技を行なう
　　（同じ距離の競技がいくつもある場合は
　　　その中で時間が最もかかる競技を行なう）
```

この説明を、「実際にコースを割り当ててスケジュールを作る方法」の形に書きなおしてみよう。割り当ては、大会開始の時刻から始めて時間を順に追って行なうことが必要になる。すなわち

1.まず大会開始時点の割り当てをする。（コースは最初は全て空いているのでそこに上の方法に基づいて競技を割り当てる。）
2.その後は、開始から 15 分後の状況、30 分後の状況、と、15 分ごとの時刻を順次想定して状況を考えてゆく。それぞれの時刻において、その時終わった競技がありコースが空いたら、上の方法に従って別の競技を割り当る。
3.全ての競技が終わったら大会を終了する。

と進めていくことになる。これを考慮して手順を書いてみよう。（第２段階）

```
//陸上競技のコース割り当て 方法２－第２段階
//　コース１～４の４つのコースに各競技を割り当てる
//
すべての競技が終了するまで時刻を１５分づつ進めながら繰り返す：
　　この時刻に競技が終了したコースがあれば
　　　　そのコースのしるしを「空」に変える
　　（終了したコースがいくつもあれば上記を繰り返す）
　　空いた／空いているコースがあれば
```

実行できる中で最も距離の長い競技を選んで開始する
（距離が同じ競技がいくつもある場合は最もかかる時間の長い
競技を選ぶ）
（競技を開始できた場合は上記を繰り返す
－もう一つ同時に開始できる場合もあるので）
繰り返しはここまで

このように時間を順に追って割り当てを進めるためには「いま開始何分後の作業をやっているのか」を控えておくことが必要になる。すなわち紙の上などに「現在時刻」という場所を設けて

・最初にそこに０分と書く
・大会開始時点の競技割当を行う
・０分を 15 分に直す
・開始後 15 分の状況での割当作業を行う
....

のように進めていくことになる。

また、上記の 2 で、一つ競技が終わってコースが空いても開始できる競技が無く、コースをしばらく空いたままにせざるをえなかったり、またその後で再度そのコースで競技を開始することになる等の場合もある。そのため「このコースはいま空いている」というしるしをつけておく必要があるので、そのための場所も用意しておく。

演習：「一つ競技が終わってコースが空いても開始できる競技が無く、コースをしばらく空いたままにせざるをえない」のはどんな場合があるか。

これらの考え方を使って、この方法をより正確に書き直してみよう。（第３段階）

```
//陸上競技のコース割り当て 方法２－第３段階
//　コース１～４の４つのコースに各競技を割り当てる
//
繰り返す：
　　//この時刻に終了した競技があればそのコースのしるしを「空」に変える
```

```
        コース１～４それぞれについて：
                そのコースでその時競技が終了するなら
                        そのコースのしるしを「空」に変える
        //コースの印を「空」に変える処理はここまで

        //空いているコースがあれば割り当てる－可能な限り繰り返す
        繰り返す：
                //空いている最長コースを調べる
                //単独の空いているコースがあるかを調べる
                コース 1～4 それぞれについて：
                        そのコースが空ならば
                                コースが 1 つ空いていることと、コース番号を控えておく
                //２つつなげて使えるコースがあるかを調べる
                コース「1 と 2」「2 と 3」「3 と 4」それぞれの組について：
                        その両方のコースが空ならば
                                コースが 2 つ空いていることと、コース番号 2 つを控えて
                                おく
                //３つつなげて使えるコースがあるかを調べる
                コース「1 と 2 と 3」「2 と 3 と 4」それぞれの組について：
                        その 3 つのコースが空ならば
                                コースが 3 つ空いていることと、コース番号 3 つを控えて
                                おく
                //空いている最長コースを調べる処理はここまで

                //空いているコースがあり、割り当てられる競技があれば割り当てる
                空いた／空いているコースがあるならば
                        今空いているコースの長さで実行できる最も長距離の競技を選ぶ
                        そのような競技があるならば
                                そのような競技（最も長距離の競技）が２つ以上あるならば
                                        その中で必要時間が最大のものを選ぶ
                                //割り当てを行なう
                                空いているコース中の、競技に使うコースにだけ：
                                        使用中の印を付ける
                                        競技が終わりコースが空く時刻を控えておく
                                //割当の処理はここまで
                //空いているコースへの割り当て処理はここまで
        繰り返しはここまで－競技を割り当てたならもう一度繰り返す
        //　さらにまた別の競技も始められるかも知れないので繰り返し試みる
        「現在時刻」を 15 分進める
繰り返しはここまで－コースが一つでも使用中なら繰り返しを続ける
```

演習：上記の方法を忠実になぞるとどのように競技が割り当てられていくかやってみよ。始めてから終るまで時間は何分かかったか。

このようにあること（この場合は競技へのコースの割り当て）を行なう方法をきちんと書いたものをアルゴリズムという。

演習：上記の方法２も必ずしも最良の割当をするものではなく、行なう競技の個数・距離・時間を変えると無駄なコースの割り当てを行なうこともある。より無駄の少ない割当をするような別のアルゴリズムを考えてみよ。それによるコース割り当てをいろいろな場合について方法２でのものと比較してみよ。

上記のアルゴリズムはまだ細かい部分を省略して書いてある。同じアルゴリズムをより細かく書いたものを下に示す。（第４段階）

```
//陸上競技のコース割り当て アルゴリズム（方法）２－第４段階
//　コース１～４の４つのコースに各競技を割り当てる
//
//記憶場所を用意する
各競技種目の名前・必要距離・時間を覚えておく場所「競技」を用意し、
        各競技種目の名前・必要距離・時間を設定する
各コースが「使用中」か「空」か、および終了時刻を覚えておく場所
        「コース１」から「コース４」を用意し、最初は「空」にする
現在時刻を覚えておく場所「現在時刻」を用意し、最初は「０分」にする

//割り当てを行なう
繰り返す：

        //この時刻に終了した競技があればそのコースのしるしを「空」に変える
        ｉを１～４まで変えながら：
                もし コースｉが使用中で
                かつ 現在時刻＝コースｉの終了時刻 ならば
                        コースｉを空にする
        //コースの印を「空」に変える処理はここまで

        //空いているコースがあれば割り当てる－可能な限り繰り返す
        繰り返す：
                //空いている最長コースを調べる
                場所「最大コース長」（最初は０とする）と「候補コース」を用意する
                //単独の空いているコースがあるかを調べる
                ｊを１～４まで変えながら：
                        もし コースｊが空 ならば
                                最大コース長を１に、候補コースをｊにする
                //２つつなげて使えるコースがあるかを調べる
                もし （コース３が空 かつ コース４が空） ならば
                        最大コース長を２に、候補コースを[3,4]にする
                もし （コース２が空 かつ コース３が空）ならば
                        最大コース長を２に、候補コースを[2,3]にする
                もし （コース１が空 かつ コース２が空）ならば
                        最大コース長を２に、候補コースを[1,2]にする
```

```
                //３つつなげて使えるコースがあるかを調べる
                もし （コース２が空 かつ コース３が空 かつ コース４が空）ならば
                        最大コース長を３に、候補コースを[2,3,4]にする
                もし （コース１が空 かつ コース２が空 かつ コース３が空）ならば
                        最大コース長を３に、候補コースを[1,2,3]にする
                //空いている最長コースを調べる処理はここまで

                「開始競技」を「なし」に設定する

                //空いているコースがあり、割り当てられる競技があれば割り当てる
                もし 最大コース長 ＞ ０ ならば
                        「競技」の中の、必要距離が最大コース長以下のものの中で
                            最も長いものを選び「開始競技」とする
                        もし 「開始競技」がある ならば
                                もし 「開始競技」が２つ以上ある ならば
                                        その中で必要時間が最大のものだけ「開始競技」
                                        とする
                                //割り当てを行なう
                                「開始競技の必要距離」の回数繰り返す：
                                //距離 100m の競技なら１回、200m の競技なら２回繰り返
                                  す
                                        「候補コース」の先頭のコースを使用中にする
                                        「候補コース」の先頭のコースの終了時刻を
                                         現在時刻＋「開始競技の終了時刻」に設定する
                                        「候補コース」から先頭のコースを削除する
                                「競技」から開始競技を削除する
                                //割当の処理はここまで
                        //空いているコースがあっても、そこでできる
                        //　開始競技がない場合は割り当てはできない
                //空いているコースへの割り当て処理はここまで

        繰り返しはここまで－「開始競技」が「なし」でないなら繰り返しを続ける
        //　さらにまた別の競技も始められるかも知れないので繰り返し試みる

        「現在時刻」を 15 分進める
繰り返しはここまで－コースが一つでも使用中なら繰り返しを続ける
```

アルゴリズムをここまで詳しく正確に書けば、コンピュータもこれを理解できるようになる。ただしコンピュータに理解できる、「プログラミング言語」という決まった形式に従って書かなければならない。これをプログラムという。これをコンピュータに与えると、上の演習で手作業でやったのと全く同じことをコンピュータは忠実に超高速で行なってくれる。

上のアルゴリズムをプログラミング言語で書いたものを以下に示す。プログラミング言語には多くの種類がある。ここではＣ＋＋（シープラスプラス）言語でのものを示す。[1]

[1] このプログラムが動くには下記のヘッダ部分を先頭に付ける必要がある。

```
//陸上競技のコース割り当て アルゴリズム（方法）２ C++言語によるプログラム
//　コース１～４の４つのコースに各競技を割り当てる
void main()
{
    //記憶場所を用意する
    t_kyougi      kyougi;       //各競技種目の名前・必要距離・時間を覚えておく場所
    //各競技種目の名前・必要距離・時間の設定
    kyougi.settei("100m走A",          1,    30);
    kyougi.settei("100m走B",          1,    30);
    kyougi.settei("200m走",           2,    30);
    kyougi.settei("300m走",           3,    30);
    kyougi.settei("100mハードル走", 1,    45);
    kyougi.settei("200mハードル走", 2,    60);

    //各コースが「使用中」か「空」か、および終了時刻を覚えておく場所
    t_course course;
    //最初は「空」にする
    course.shiyou[1] = aki;
    course.shiyou[2] = aki;
    course.shiyou[3] = aki;
    course.shiyou[4] = aki;

    //現在時刻を覚えておく場所
    int genzaijikoku = 0;       //最初は「０分」にする

    //割り当てを行う
    do {
        //この時刻に終了した競技があればそのコースのしるしを「空」に変える
        for (int i = 1; i <= 4; ++i) {
            if (course.shiyou[i] == shiyoochuu
               && genzaijikoku == course.shuryoujikoku[i]) {
                course.shiyou[i] = aki;
            }
        }
        //コースの印を「空」に変える処理はここまで

        //空いているコースがあれば割り当てる－可能な限り繰り返す
        t_intset kaishikyougi;
        do {
            //空いている最長コースを調べる
            int saidai_course_chou = 0;
            t_courseset kouho_course;
            //単独の空いているコースがあるかを調べる
            for (int j = 1; j <= 4; ++j) {
                if (course.shiyou[j] == aki) {
                    saidai_course_chou = 1;
                    kouho_course.settei(j);
                }
            }
            //２つつなげて使えるコースがあるかを調べる
            if (course.shiyou[3] == aki && course.shiyou[4] == aki) {
                saidai_course_chou = 2;
                kouho_course.settei(3, 4);
            }
            if (course.shiyou[2] == aki && course.shiyou[3] == aki) {
                saidai_course_chou = 2;
                kouho_course.settei(2, 3);
            }
            if (course.shiyou[1] == aki && course.shiyou[2] == aki) {
```

```
                saidai_course_chou = 2;
                kouho_course.settei(1, 2);
            }
            //３つつなげて使えるコースがあるかを調べる
            if (course.shiyou[2] == aki && course.shiyou[3] == aki && course.shiyou[4] == aki) {
                saidai_course_chou = 3;
                kouho_course.settei(2, 3, 4);
            }
            if (course.shiyou[1] == aki && course.shiyou[2] == aki && course.shiyou[3] == aki) {
                saidai_course_chou = 3;
                kouho_course.settei(1, 2, 3);
            }
            //空いている最長コースを調べる処理はここまで

            kaishikyougi.clear();

            //空いているコースがあり、割り当てられる競技があれば割り当てる
            if (saidai_course_chou > 0) {
                kaishikyougi = kyougi.saidaichou_erabu(saidai_course_chou);
                if (kaishikyougi.num_elem >= 1) {
                    if (kaishikyougi.num_elem >= 2) {
                        kaishikyougi = kyougi.saidaijikan_erabu(kaishikyougi);
                    }
                    //割り当てを行う
                    for (int c = 1; c <= kyougi.kyori[kaishikyougi.value()]; c++) {
                    //距離 100m の競技なら１回、200m の競技なら２回繰り返す
                        course.shiyou[kouho_course.course_elem[0] ] = shiyoochuu;
                        course.shuryoujikoku[kouho_course.course_elem[0] ]
                          = genzaijikoku + kyougi.jikan[kaishikyougi.value()];
                        //割り当ての表示
                        printf("jikan=%i, kyogi=%s, course=%i, shuryojikoku=%i¥n",
                          genzaijikoku, kyougi.namae[kaishikyougi.value()],
                          kouho_course.course_elem[0],
                          course.shuryoujikoku[kouho_course.course_elem[0] ] );
                        kouho_course.sakujo();
                    }
                    kyougi.sakujo(kaishikyougi.value() );
                    //割当の処理はここまで
                }
                //空いているコースがあっても、そこでできる
                //　開始競技がない場合は割り当てはできない
            }
            //空いているコースへの割り当て処理はここまで
        } while(kaishikyougi.num_elem >= 1);
        //　さらにまた別の競技も始められるかも知れないので繰り返し試みる
        genzaijikoku += 15;
    } while (course.shiyou[1] == shiyoochuu || course.shiyou[2] == shiyoochuu
          || course.shiyou[3] == shiyoochuu || course.shiyou[4] == shiyoochuu);
}
```

コンピュータが動くには必ずプログラムが必要である。どんな時もコンピュータが動いている時には、その中にはかならず、誰かが作った上の例のようなプログラムが入っており、コンピュータはそれに忠実に従ったことを超高速に行なっているにすぎないのである。

[ヘッダ部分（編注）]

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
const SAIDAIMOJI = 64;
const SAIDAIKYOUGI = 32;

// t_使用：空きか使用中か
enum t_shiyou {aki, shiyoochuu};

// t_intset：整数の集合
class t_intset {
public:
    int         num_elem;                   //要素の数
    int         elem    [SAIDAIKYOUGI];

    t_intset()    {num_elem = 0;}
    void clear() {num_elem = 0;}
    // add：集合に要素を加える
    void add(int a){if (num_elem >= SAIDAIKYOUGI) exit(1); elem[num_elem] = a; ++num_elem; }
    // value：集合の要素が一つの時にその値を得る－複数要素が入っている時は使用不可
    int value() {if (num_elem != 1) exit(1); return (elem [0]);}
};

// t_競技：競技一覧の集合
class t_kyougi {
private:
    int       num_elem;                                     //要素の数
public:
    char      namae    [SAIDAIKYOUGI] [SAIDAIMOJI] ;    // 競技の名前
    int       kyori    [SAIDAIKYOUGI];                    // 必要距離
    int       jikan    [SAIDAIKYOUGI];                    // 時間
public:
    t_kyougi() {num_elem = 0;}
    void settei(char *a_namae, int a_kyori, int a_jikan);    // 設定：集合に要素を加える
    // 最大長選ぶ：最大コース長以内で距離が最長の競技を選ぶ
    t_intset saidaichou_erabu(int saidai_course_chou);
    // 最大時間選ぶ：インデックス集合ａから時間が最長のものを選びそのインデックスを返す
    t_intset saidaijikan_erabu(t_intset a);
    void sakujo(int a);    // 削除：競技集合からインデックスで与えられた要素を削除する
};

// 設定：競技の集合に要素を加える
void t_kyougi::settei(char *a_namae, int a_kyori, int a_jikan)
{
    if (num_elem >= SAIDAIKYOUGI) {
        printf("競技数が多すぎる¥n");
        exit(1);
    }
    strcpy(namae[num_elem], a_namae);
    kyori[num_elem] = a_kyori;
    jikan[num_elem] = a_jikan;
    ++num_elem;
}
// 最大長選ぶ：最大コース長以内で距離が最長の競技を選ぶ
t_intset t_kyougi::saidaichou_erabu(int c)
{
```

```
    //登録されている競技の中で最長の距離を求める
    int saidaichou = 0;
    for(int i = 0; i<num_elem; ++i) {
        if (kyori[i] <= c && kyori[i] > saidaichou) {
            saidaichou = kyori[i];
        }
    }
    //最長の距離を持つ競技を全て取り出しそのインデックスの集合を返す
    t_intset r;
    for(int j = 0; j<num_elem; ++j) {
        if (kyori [j] == saidaichou) {
            r.add(j);
        }
    }
    return r;
}

// 最大時間選ぶ：インデックス集合ａから時間が最長のものを選びそのインデックスを返す
t_intset t_kyougi::saidaijikan_erabu(t_intset a)
{
    int x_saidaijikan = a.elem[0];
    for (int i = 0; i<a.num_elem; ++i) {
        if (jikan[a.elem[i] ] > jikan[x_saidaijikan]) {
            x_saidaijikan = a.elem[i];
        }
    }
    t_intset r;
    r.add(x_saidaijikan);
    return r;
}

// 削除：競技集合からインデックスで与えられた要素を削除する
void t_kyougi::sakujo(int a)
{
    for (int i = a; i< (num_elem - 1); ++i) {
        strcpy(namae[i], namae[i+1]);
        kyori[i] = kyori [i+1];
        jikan[i] = jikan [i+1];
    }
    --num_elem;
}
// t_コース：４つのコースのそれぞれの状態を保持する（要素１～４のみ使用）
struct t_course {
    t_shiyou    shiyou        [5];    //コースがあいているかどうか
    int         shuryoujikoku [5];    //今そのコースで行っている競技の終了時刻
};

// t_コースセット：コースの集合を保持
class t_courseset {
public:
    int course_elem [4];
    t_courseset() {for (int i = 0; i<4; ++i) {course_elem[i] = 0;} }
    // 設定：集合をクリアし値を一度にセット
    void settei(int a0) {course_elem[0] = a0;}
    void settei(int a0, int a1) {course_elem[0] = a0; course_elem[1] = a1;}
    void settei(int a0, int a1, int a2)
        {course_elem[0] = a0; course_elem[1] = a1; course_elem[2] = a2;}
    void sakujo();
};
```

```
//   削除：コースの集合から第０要素を削除
void t_courseset::sakujo()
{
      for (int i = 0; i<3; ++i) {
            course_elem[i] = course_elem[i+1];
      }
      course_elem[3] = 0;
}
```

第２節　データ管理

人間は何らかの知的活動をした後、その成果をそのまま保存しておきたいと考える。それは、文章であったり、絵であったり、楽譜であったり、マルチメディアであったりする。そして保存したものは、後から取り出して見たくなる。

人間の脳には無限の情報を蓄積できるような気がするが、必要なときに思い出せない。その癖、呼出しがかかっていないのに、昔の思い出が鮮明に蘇ることがある。ときには、意識の中に無いことまで引き出されてくる。

そこで、必要な情報を他所から引き出してきたり、大事な情報を後で使いやすくするために整理することを考える。コンピュータだって同じこと。きちんと整理しておかなければ、後で分からなくなってしまう。

能率良く仕事をする人は、きっと上手に整理しているに違いない。そこで、コンピュータに効率よくデータを保管し活用する方法について学習することにしよう。

１　データの整理と活用

これまでにコンピュータは大量のデータを処理するのが得意であることを学んできた。処理手順を示すのがアルゴリズムであることも知った。実は、このアルゴリズムの効率はデータの構造に非常に左右されるのである。そこに、データの整理・保管・活用と、その管理がかかわっている。

＜表の作成＞

第２章で扱った催し企画問題ではいくつかの名簿を作成する必要があった。郵便物を配布するのに必要な情報として、次のようなデータ[2]の種類が考えられる。

ア）相手の住所と郵便番号・所属・相手の氏名
イ）自分の住所と郵便番号・所属・自分の氏名
ウ）差出年月日
エ）郵便料金

以上のデータには二つの性質がある。つまり、不変的なデータ（しかし、永久的ではない）と可変的なデータである。（ア）と（イ）は不変データで、（ウ）と（エ）は郵便物ごとに異なる可変データである。

このうち不変データは何回も使うことがあるので、どのように整理しておいたら扱い易いかを考えることにしよう。（ア）のデータには、

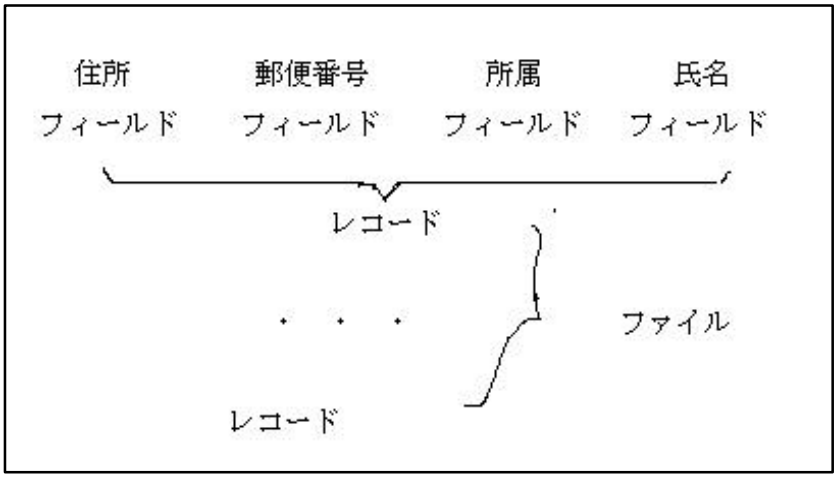
---

[2] 情報とデータ構造の定義は違う。情報処理ハンドブックによれば、「情報とは認知や思考の対象となる実体についての認識内容」で、「データとは情報を記述、表現するときの最小単位（あるいは、その集合）」である。本書は、この定義に従っている。

住所
郵便番号
所属
氏名

の四項目がある。このそれぞれをフィールドと呼ぶことにしよう。実は、（イ）のデータも同様のフィールドからなっている。このように一組のフィールドからなるデータの構造をレコードという。この例では、レコードは四つのフィールドから構成されている。

レコードは宛先の数だけあるから、宛先が 30 件あるとすれば、自分の分も合わせて 31 のレコード数が必要となる。この 31 レコードが一つのファイルを構成する。レコードは、メンバーの増減に合わせて、追加したり削除したりすることが可能である。



――――― 演習問題３－１ ――――――――

ワープロを使って、友達や知人の住所をこの表に記入してみよう。また、そこで気づいたことや疑問を述べなさい。個人の住所の場合には所属は不要であるので、その場合は空欄にしよう。

――――――――――――――――――――――――

実際に記入したことによって、いくつかの疑問が生まれたことであろう。たとえば、つぎのような疑問を想定しよう。

・住所の長さが違うが、自由なのか。
・一行に書き切れない場合には二行にまたがってもよいのか。
・郵便番号は数値[3]で記入するのか、文字列[4]で記入するのか。
・氏名は漢字、ひらがな、カタカナ、アルファベットのどれでもよいか。

[3] 数値とは四則演算ができる数字である。よく使用される数値に整数値と実数値がある。郵便番号に演算処理を施したい場合には数値として扱うが、小数点つきの数ではないので整数値である。

[4] 文字列は、数字、アルファベット、かな文字、漢字、その他の記号からなる列である。数字だけからなる文字列は、一般にコードとして使用される。郵便番号はコードであるから

・名前はいくら長くても良いのか。
・特殊記号を含んでもよいのか。
・氏と名は分けるのか続けるのか。

この他おそらく、いろいろな疑問を感じた人がいるであろう。それらの疑問は、フィールドの長さ（以後フィールド幅という）に関することと、文字列に関することに分類できるであろう。

このような表は、自由に記入できるように作成することも可能であるが、表が見にくいばかりでなく使い心地もよくない。また、コンピュータが処理するにしても、個々のフィールド幅を確認したり、いろいろな文字列と照合する必要があり、処理時間も大きくなる。

ここでは、予めフィールドごとに、フィールド幅（バイト数[5]で示す）や文字列の型（数字、アルファベット、かな文字、漢字、その他の記号など）を定義しておこう。フィールド幅の標準は決まっている訳ではないので、それぞれの交友範囲で上限を定めればよい。

――――― 演算問題３－２ ―――――――――

フィールド幅や文字列を定義して住所録ファイルを完成してみよう。見やすい表ができたであろうか。

――――――――――――――――――――――――

――――― 演算問題３－３ ―――――――――

住所録を拡張して、電話番号欄を追加してみよう。

――――――――――――――――――――――――

――――― 演算問題３－４ ―――――――――

さらに、電子メールアドレスを付加するとどうなるか。

――――――――――――――――――――――――

＜表の利用＞

表ができたら、友達の住所を探してみよう。このように、既存の表からデータを探すことをデータ検索という。実際には、いずれか一つのフィールドに注目して検索することが多い。

検索してみると、不便なことや疑問が更にまた生まれるであろう。たとえば、
・フィールド順がよくないのではないか。
・たくさんのレコードの中から必要な情報を探し出すのは結構面倒だ。
・電話帳のように、レコード数が多くなったらどうするのだろう。
・実際には名前以外のキーワードで検索したいこともある。
といった問題が発生するかもしれない。

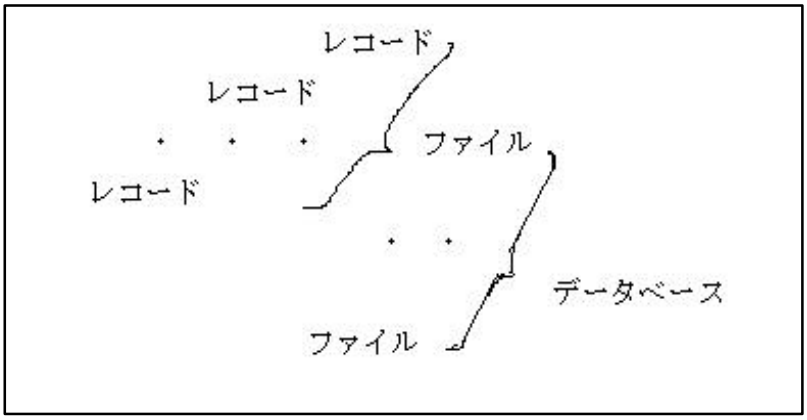
それならば、このファイルの構造をもっと使いやすい構造に設計し直してみよう。検索

文字列として扱うことが多い。
[5] ２進数の８けた分を１バイトと数える。

しやすさに注目するならば、先ずフィールド順番の入れ替えが考えられる。これは予め決めておくことができるので可能であろう。

名前が五十音順に並ぶようにレコードの順番を入れ替えるとどうなるか。それは便利である。しかし、初めから順番を決めてデータを入力する方がもっと大変ではないか。そこで、考えられるのは、一度入力したレコードの順番をいつでも自由に入れ替えられる仕組みを作ることである。このような仕組みをもった便利なパッケージソフトに、表作成ソフトがある。

しかし、電話帳のように膨大なレコード数を持つ表には、表作成ソフトでは少し負担が大きすぎる。実は、そのような大量データを管理できる仕組みが既に存在している。それはデータベースとよばれている。



２　データベースの仕組み

情報処理はコンピュータが出現する以前から行われていたが、コンピュータの登場によってさらに多くの人に関心を持たれるようになった。それは、コンピュータを使って大量のデータを蓄積したり速く加工したりすることが可能になったからである。

さらに、通信技術とコンピュータ技術の融合が遠隔地の情報処理をも可能にした。コンピュータによる今日の情報処理は、空間と時間の垣根を越えて行われているということができよう。

我々は多くのデータを使って情報処理をしているが、そのデータをどのように保管すると利用効率が良いのであろうか。

＜データと情報＞

この本には、「データ」と「情報」という言葉がしばしば出現する。これらの言葉はよく混同されるが、この２つの言葉は別のものである。「データ」とは、文字列や数値の形で形式的に表現された事象や概念である。これに対して、「情報」は、データが現す内容であり、事実に基づいた意味を持っている。

基本となるデータには数値や文字列の他、論理値による表現もある。論理値は、「真」または「偽」のいずれかの値で示される。この他、図形や画像、音声や楽音などをディジタル表現したデータもある。これらのデータは、すべて０と１の２進表記がなされているが、情報として再現できるようにするために、その型式やフィールド長など構造を示す情報も記録する。

＜ファイルとデータベース＞

データをある規則にしたがって保存し、必要に応じて参照できるようにした一つのまとまりをファイルとよび、それぞれのファイルは同じ書式にしたがってデータを記述する。データには型式などいろいろな制約があり、その論理的な構造を記述したものをスキーマとよぶ。

我々が参照の対象とするのは、ファイルを構成するレコードである。レコードを参照することをアクセスという。レコードを構成しているデータ項目はいつも同じ目的で参照される訳ではなく、同じデータがさまざまな目的で利用される。そのため、ファイルの検索や更新などの操作がしやすいように、登録簿を用意してファイルの保管場所がわかるようにしている。

プログラムとデータは相互に強い関係があるが、プログラムごとにデータを備えておくのはあまり効率の良いやりかたではない。それは、データを蓄積する労力からもみても、コンピュータの記憶領域から見ても無駄が多過ぎる。

同じデータがいくつかのプログラムに分散していると、あるデータの修正が必要になったときに、関係する全てのファイルのありかを探してデータを書き換えるという膨大な作業が発生する。また、重複したデータの間で矛盾が起こる危険性もある。

このような欠点を改善するために、いくつものファイルを一つにまとめて、いろいろな利用目的に対応できるようにすることが考えられた。このファイルの集まりをデータベースという。データベースとは、プログラムとは独立に整理したデータを蓄積しておき、条件に合致するデータを検索して抽出できるような仕組みを備えた大きなデータの基地である。

＜データベースの特徴＞

多目的利用のためには、

・データと利用するプログラムとは独立であること（データ独立という）
・データに重複がないこと
・データベースが安全かつ完全であること

が必要とされる。この特徴を、データベースの独立性、一貫性、安全性・完全性という。このような特徴を維持・管理し、利用者とのインタフェースを備えたシステムをデータベース管理システム（ＤＢＭＳ[6]という）という。また、データベース管理システムとデータの集合を合わせてデータベースシステムという。データベース管理システムには次のような機能が必要とされている。

DBMS の主な機能
・データ構造を定義するための機能
・データを検索、挿入、変更、削除するための機能
・データを維持する機能
・データを安全に守るための機能

[6] Data Base Management System の略である。

・スキーマの情報を管理する機能

＜データベースの構造＞

データベースのデータはいろいろな形式で表現される。その表現は利用者からみて使いやすい構造であること、コンピュータからみれば少ない記憶容量で格納できることが望ましい。そのため、データベースは、外部レベル（利用者の視点）、概念レベル[7]（論理的な記述の視点）、内部レベル（コンピュータの視点）の３つの見方に分けて構造化されている。この構造をデータベースの３層構造という（図３－１）。この構造において、外部レベルのモデルは複数あり、それぞれの利用者やプログラムの都合に合わせて自由に作ることができる。実際には、概念レベルで一つのデータベースであっても、内部レベルでは多数のディスクなどに分かれて格納されている。

個々のプログラムに対応する外部レベルのスキーマを外部スキーマ、データ格納に対応する内部レベルの構成を内部スキーマという。またデータベース全体の論理的な記述レベル（概念レベル）に対応するスキーマを概念スキーマとよぶ。このそれぞれのスキーマの設計のよしあしがデータ操作の効率に大きな影響を及ぼす。

データベースのスキーマを記述することをデータ定義といい、データ定義を容易にするためにデータ定義言語[8]が開発されている。

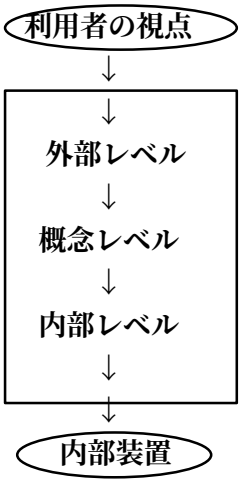


図３－１　データベースの３層構造

＜データベースソフトウェア＞

データベースシステムは、そもそも大量のデータを蓄積するために考えられたものであるが、今日では、大規模コンピュータで使うものからパソコン上で使えるものまでいろいろある。

これらのデータベースソフトウェアの多くは、次の機能を持っている。

---

[7] 概念レベルのモデルの設計には、実体関係図（２章参照）などが利用される。

[8] たとえば、問い合わせ／操作言語などがよく知られている。

（1）データベースの構造を設計する機能
（2）データを入力する機能
（3）データを編集する機能
（4）データベースに登録する機能
（5）テーブルを複写する機能
（6）データベースに登録されているデータを修正する機能
（7）条件に合致するデータをデータベースから検索する機能
（8）検索したデータを印刷する機能

このような機能を備えたデータベースソフトウェアの中でよく利用されるものに、リレーショナルデータベースがある。図３－２はリレーショナルデータベースソフトウェアを利用したデータベース作成から利用までの流れを示している。

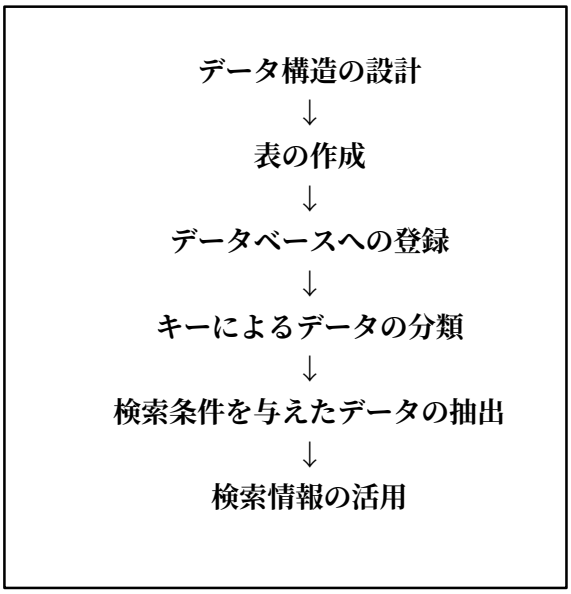


図３－２　リレーショナルデータベースの利用

また、一枚のカードで資料を整理する感覚で利用するカード型データベースがある。たとえば、企業ごとの情報や名刺の管理などに使われている（図３－３）。

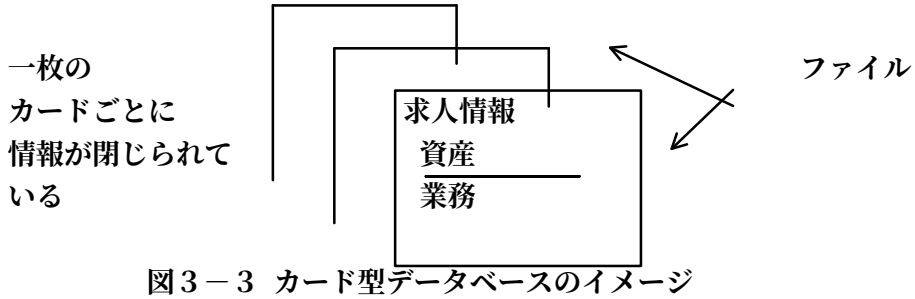


図３－３ カード型データベースのイメージ

――――― 演習問題３－５ ―――――――

クラブ活動の案内を、クラブごと一枚のカードで整理することになった。カードのレイアウトを考えてみよう。
また、カードを分類するために、どのような項目を備えておくと便利か。
――――――――――――――――――――――――

３　データベースの設計

パソコン用のデータベースソフトウェアを利用する場合でも、データベースの設計は大事な作業である。利用に先立って、まず、データを集めて整理しなければならない。この基本的な作業は、表を作成するときと同じように、データ項目を選定しその構造と型式を決めてレコードを構成する。

さらに、使用目的に対応できるようにデータ項目の相互関係を分析する。データ項目を関連づけたものをデータモデルといい、このモデルを設計することがデータベースを設計する基本作業となる。つまり、データベースの設計とはデータモデルの設計を意味している。代表的なデータモデルとして、階層モデル、ネットワークモデル、関係モデル、オブジェクト指向モデルがある。

パソコン用のデータベースソフトは、対応するデータモデルが予め決まっているため、データ項目の設計ができたところで使用するソフトウェアを選択することになる。

＜階層モデル＞

データを会社の組織のような階層的な構造で整理したもので、たとえば図４のような事例で示すことができる。この構造は逆さにすると枝をはった木のように見えるために、木構造とも呼ばれている。木の枝が交わらないのと同じように、階層モデルも交わることはない。レコードに親子の関係を持たせるが、子は一つの親しか持てない。処理能力は優れているが、木構造よりも複雑な関係を表現できないのが難点である。

［事例］クラブ活動のモデルを階層構造で表現しよう。

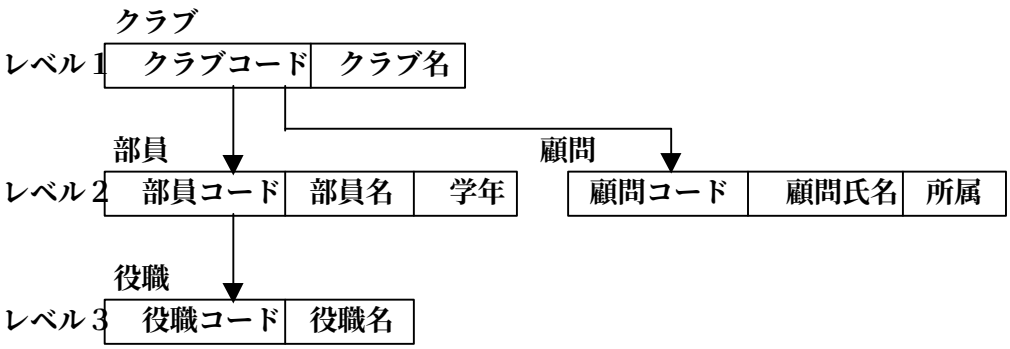


図３－４　階層型データモデルのスキーマ例

クラブ活動には各クラブの名前があり、部員がいて、顧問の先生がいる。部員には部長や副部長などの役職がある。この関係は図３－４のような３階層で表現できる。

「クラブ」、「部員」、「顧問」、「役職」はレコードを示している。枠内はそれぞれのレコードを構成するフィールド（データ項目）である。

――――― 演習問題３－６ ―――――――

卒業生のデータベースを作成したい。データ項目を考えて階層型データモデルで設計しなさい。

――――――――――――――――――――――――

――――― 演習問題３－７ ―――――――

レンタル会社の商品管理をデータベースで行いたい。品物を設定してどのようなデータ項目で商品を管理したら良いか考えなさい。

――――――――――――――――――――――――

＜ネットワークモデル＞

階層モデルをより一般化したモデルで、親が複数の子を持つばかりでなく子が複数の親レコードを持つこともできるようになっている。つまり、レコード間で親子を相互に関連づけられる構造モデルである。上下左右に関連する網のような構造をしていることからネットワークモデルと呼ばれている。データ構造が複雑になりすぎるのが難点である。

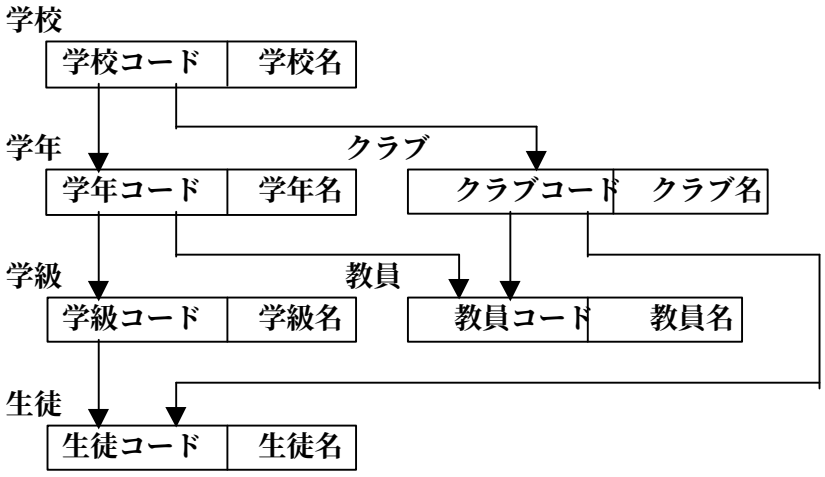


図３－５ ネットワーク型データモデルのスキーマ例

図３－５のネットワークモデルでは、教員が学年にもクラブにも所属し、生徒は学級にもクラブにも所属しているという関係を示している。このモデルでは、このように子が親を 2 つ以上もっている。

――――― 演習問題３－８ ―――――――

模擬試験の成績を整理するデータベースを考えるとき、どのようなデータ項目が必要か。さらに、受験に必要な複数大学の情報や他の高校の情報を取り入れるならば、どのようなデータ構造モデルが考えられるか。

――――――――――――――――――――――――

＜関係モデル＞

学年テーブル

学年コード	学年名	・

学級テーブル

学級コード	学級名	教員

生徒テーブル

生徒コード	生徒氏名	性別
98001	安藤三郎	男
98002	伊藤陽子	女
98003	小川勝	男

教科テーブル

教科コード	教科名	教員

評価テーブル

生徒コード	数学	英語
98001	85	70
98002	58	95
98003	100	85

図３－６　関係データモデルのスキーマ例

生徒コード	生徒氏名	性別	数学	英語
98001	安藤三郎	男	85	70
98002	伊藤陽子	女	58	95
98003	小川勝	男	100	85
・・				

図３－７　テーブルの結合

データをいくつかの２次元表の形に整理するという非常に単純な構造をしている。リレーショナルデータモデルともいう。

行をレコード、列を項目と考え、関係（リレーション）という概念を導入してスキーマを定義するので関係モデルと呼んでいる。このモデルに基づいて構成するデータベースをリレーショナルデータベースと呼んでいる。図３－６のように２次元の表を複数作成するのが普通である。たとえば、生徒テーブルと評価テーブルを生徒コードで関係づけると図３－７が得られる。

――――――　演習問題３－９　――――――――

関係モデルを利用して、学級名簿のデータベースを作成してみよう。具体的なデータ項目を挙げてから表を設計しよう。

――――――――――――――――――――――――

＜オブジェクト指向モデル＞

オブジェクト指向モデルは他の３つのモデルと比べて、まだ体系化が遅れている。現実に見えているもの（オブジェクト）どうしの関係を、直観的にコンピュータ画面上に表現しようという発想から生まれたモデルである。その構造は分かりやすく、あるオブジェクトから別のオブジェクトに作業を依頼し合うという関係で示される。たとえば、パソコンというオブジェクトＡがあって、製品番号、製品名、製造工場、利用者という属性値をも

っているとする。この属性の表現で他のモデルと違うところは、メソッド[9]とよばれる操作手続きを一体化していることである。また、部品というオブジェクトBがあって、部品名、部品番号、使用機種、製造工場などの属性値とそのメソッドが一体化しているとしよう。オブジェクトAとオブジェクトBの間ではメッセージを通して関連づけが行なわれる。

４　データベースの検索

検索の問題とは、情報をどのように記憶装置内に格納し、その中の必要なデータをどのようにして見つけ出すかということである。

データが少ない場合には、計算方法に合わせてデータを用意することができる。しかし、データが大きくなり、データと処理プログラムが分離されると計算式に合わせてデータを準備することは不可能になる。

その結果、データを作る側は個々の計算にとらわれないでデータを表現し、データを使う側は多くのデータの中から必要なものだけを選び出して使うことになる。そこで、多くのデータから必要なデータを探し出すための方法が必要となる。データの検索とは、特定の内容と一致しているものを見付け出すことである。そのために、使う側は条件を与えてデータベースに問い合わせをする。

問い合わせる条件が一つだけの場合に単純条件、複数の条件を組み合わせて指定する場合に複合条件という。

単純条件の与え方には次のような場合がある。

あるデータと等しいものを探す
あるデータと等しくないものを探す
あるデータより小さいものを探す
あるデータより大きいものを探す
あるデータ以上のものを探す
あるデータ以下のものを探す
ある文字列と同じものを探す
ある文字列を含んでいるものを探す

以上を組み合わせたのが複合条件である。

「 AND（かつの意味)」 または 「 OR（またはの意味)」をつかって条件を結合して問い合わせる。

たとえば探したい本の名前を図書目録のデータベースで検索するとしよう。その本には「情報」という言葉と「コンピュータ」という言葉が含まれていると考えるときには、情報とコンピュータの両方に関係する書物を検索すればよいから、AND を使用して、

情報　AND　コンピュータ

というキーワードを与えることができる。あるいは「情報」か「コンピュータ」かいずれ

[9] オブジェクトに付随する操作をメソッドという。

かの言葉を含む本を探そうとするならば、

情報　OR　コンピュータ

で検索すればよい。

　このようなデータベースの検索を効率よく行うためには、優れたデータの探索、文字列の照合、データの整理の方法が重要になる。

――――――　演習問題３－１０　――――――――
電車の時刻表のデータベースを作成したい。どのような検索をしたいか考えよう。
――――――――――――――――――――――――

＜データの探索＞

　ファイルから必要なレコードを探し出す処理をでーたの探索という。すなわち、レコードのあるフィールドとその値を指定して、それに合致するデータやその格納場所を求める処理である。たとえば、住所録ファイルから、住所がＡ市で年令が１７歳のレコードを探す処理が考えられる。

［例題３－１］大きさの順[10]に整列されているデータのファイルXがあり、その中にＰというデータがあるか探したい。

　昇順に並んでいるデータが、Ｘの１番目からｎ番目に、Ｘ(１)、Ｘ(２)、..、Ｘ(ｎ)という名前で格納されているとしよう。Ｘ(１)から順にＰと比較しながら同じものがあるか探す方法がある。逐次探索または順探索とよばれる方法である。

　ｋ番目のデータがＰと同じであればＸ(ｋ)が探し当てたデータであり、ＰがＸ(ｋ)より大きくてＸ(ｋ＋１)より小さい場合には、データを探し当てられなかったことになる。

　データが非常に少ない場合にはこの方法が単純で使いやすい。しかし、データが多い場合には非常に時間がかかる。そこで、データを２つのグループに分けてＰがどちらのグループに入っているかを探しながら、目標の範囲を狭めていく方法がある。

　ｎ個のデータのうち、中央のデータＸ(ｋ) [11]とＰを比較して、それより小さい場合には下の方のグループを更に探索し、大きい場合には上の方のグループを探索するという手順を繰り返す。Ｘ(ｋ)がＰと一致すれば探索は成功である。この探索方法を二分探索と呼ぶ。図３－８は二分探索によるチェックの過程を示している。図中の①は始めにＰと比較する位置であり、その大小によって次に探す位置を左右の一方に決める。②と③の数字は、何回目に比較する位置であるかを示している。

　目的のものが見つかる前に探索できる範囲がなくなってしまえば、探索は失敗である。

---

[10] 小さい方から大きい方へ並べる昇順と逆に並べる降順がある。

[11] ｎが奇数の場合には、ｋ＝（ｎ＋１）／２ であり、ｎが偶数の場合には、ｋ＝ｎ／２ とすればよい。

――――― 演習問題３－１１ ――――――――

１，５，８，１３，２９，３８，３９，４３，６１，７８，８０，９２，９３，９９
のデータ列がある。二分検索法をつかって、Ｐ＝５０、Ｐ＝９２ をそれぞれ探索してみよう。何回探索を繰り返すか。

――――――――――――――――――――――

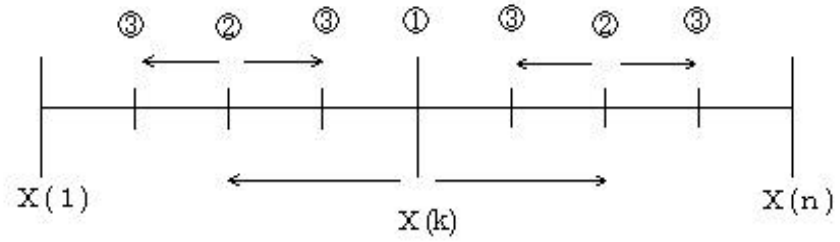


図３－８　二分探索法

＜文字列の照合＞

　文字の長い並びの中から、ある指定したパターンを探すという処理は、データベースやワープロの探索でよく使われる。

［例題３－２］ｎ個の文字が並んだテキストＴＸＴ［１・・ｎ］の中に、ｍ個の文字が並んだパターンＰＴＮ［１・・ｍ］が含まれているとき、その位置を探す。ただし、テキストに同じパターンが複数ある場合には、その最初に現れる位置を示すことにする。

テキストＴＸＴ

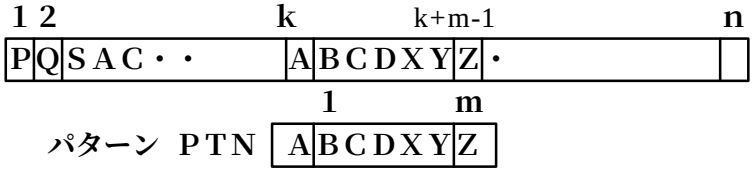


図３－９　文字列の照合

　テキストＴＸＴのｋ番目からｍ文字がパターンＰＴＮと合致したとすると、図３－９のような関係が得られる。このとき、ｋが求める位置となる。

　ここでは、最も単純な照合方法を説明しよう。ＴＸＴの１番目の文字とＰＴＮの１番目の文字から照合を始める。ＴＸＴ（１）とＰＴＮ（１）が同じ場合には次の桁すなわちＴＸＴ（２）とＰＴＮ（２）を照合する。もし、照合に失敗したらＰＴＮを一つ右にシフトして、ＴＸＴ（２）とＰＴＮ（１）を比較する。以下この手順を繰返し、ＰＴＮの全桁同じであったら、ＰＴＮ（１）に対応するＴＸＴの位置が求めるものである。

この方法を単純照合法という。

――――― 演習問題３－１２ ―――――――
テキストの文字列
ＰＲＯＪＥＣＴＳ ＡＮＤ ＰＲＯＧＲＡＭＳ，ＰＲＯＧＲＡＭＭＩＮＧ ＬＡＮＧＵＡＧＥ ＧＵＩＤＥ，において、ＰＲＯＧＲＡＭ という文字列のパターンを照合しなさい。ただし、スペースも一文字とする。照合に成功するまで文字の比較は何回行われたか。
――――――――――――――――――――――

＜データの整列＞
データの探索の例では、データが大きさの順に並んでいた。このようにデータを一定のルールにしたがって並べ替える処理をデータの整列（ソート）という。

［例題３－３］一組８枚のカードに異なる数字が書いてあり、次のように並べてある。このカードを昇順に並べ替えてみよう。
９，１５，３，５３，６，２１，３０，６０

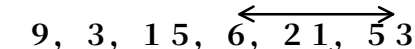
このようにデータが少ない場合には、容易に最小のカードを探すことができる。一番小さいカードを選んで、一番目の位置のカードと入れ替える。次に、二番目に小さいカードを選んで、二番目の位置のカードと入れ替える。この手順を繰り返すとカードは次のように置きかわっていく。

始めの状態　９，１５，３，５３，６，２１，３０，６０
　　３，１５，９，５３，６，２１，３０，６０
　　３，６，９，５３，１５，２１，３０，６０
　　３，６，９，１５，５３，２１，３０，６０
　　３，６，９，１５，２１，５３，３０，６０
　　３，６，９，１５，２１，３０，５３，６０

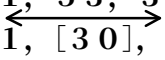
以上のように、このデータでは５回の入れ替えで整列ができた。このようにデータを選択して順番の位置に入れ替えていく方法を単純選択法という。

隣り合ったデータを順次比較して大小の順序が逆転している場合に位置を入れ替えるバブルソートという方法もある。例題３－３を、この方法で解決してみよう。この方法も、比較を何回か繰り返す。ここでは、入れ替えを行ったところを矢印で示すことにする。最後に入れ替えを行った所に印をつけておくと、次回はその先を比較しなくてもよい。ここでは、［　］を付けることにしよう。

始めの状態　　９，１５，３，５３，６，２１，３０，６０
１回目のパス　９，３，１５，５３，６，２１，３０，６０
　　９，３，１５，６，５３，２１，３０，６０

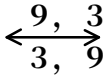


９，３，１５，６，２１，５３，３０，６０

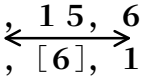


９，３，１５，６，２１，［３０］，５３，６０

２回目のパス　９，３，１５，６，２１，３０，５３，６０

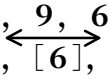


３，９，１５，６，２１，３０，５３，６０



３，９，［６］，１５，２１，３０，５３，６０

３回目のパス　３，９，６，１５，２１，３０，５３，６０



３，［６］，９，１５，２１，３０，５３，６０

４回目のパスは入れ替えがないので、これで整列は終了する。

――――― 演習問題３－１３ ―――――――

データ列

４０，８８，３５，７０，５５，１５，１０，２５，９５，６０

を単純選択法とバブルソート法で整列してみよう。

――――――――――――――――――――――