大井

配布資料

1. **本資料**
2. **文献 1** Problem F: Young, Poor and Busy (ACM International Collegiate Programming Contest, Asia Regional Context, Hakodate, 2001-11-11)
3. **文献 2** 寺田実：どこで会える？ (情報処理, Vol.44, No.2, pp192–197, 2003.)

1 総合課題

文献 1 "Problem F: Young, Poor and Busy" のとおり。但し、チェックポイント 5 以降では、この問題を拡張する。

なお、この問題の解説、および、解答例の一部は、文献 2 にかなり詳しく書かれている。これを参考にしてよい。

2 チェックポイント

総合課題を解くために、次の 6 つのチェックポイントを設ける。これらのチェックポイントをひとつづつ解いていけば、最終的に、総合課題を解くことができる。

2.1 チェックポイント (1): 列車データの準備

ファイルから列車データを読み込むサブルーチンを作成する。

このサブルーチンが正しく動作することを、次のようなプログラムを作成して確認する。

1. 最も金額が高い列車を求める（出力する）。
2. 最も時間がかかる列車を求める（出力する）。
3. 特定の駅 { から出発する／に到着する } 列車を出力する。

2.2 チェックポイント (2): fr_h を求める

文献 2 に基づいて、fr_h を求めるサブルーチンを作成する。

このサブルーチンが正しく動作することを、次のようなプログラムを作成して確認する。

1. 函館を起点とした fr_h の表（文献 2 の p194 の表）を出力する。
2. 東京を起点とした fr_h の表を出力する。

2.3 チェックポイント (3): to_h を求める

文献 2 に基づいて、「フィルムの逆回し」のアイディアを使って、to_h を求めるプログラムを作成する。

このプログラムが正しく動作することを、次のようなプログラムを作成して確認する。

1. 函館を着点とした to_h の表を出力する。
2. 東京を着点とした to_h の表を出力する。

2.4 チェックポイント (4): 解を求める

これまで作成してきたサブルーチンを組み合わせて、解を求めるプログラムを作成する。

このプログラムが正しく動作することを、文献 1 のデータ (sample input) に対して適用し、sample output と同じ出力が得られることを確認する。

2.5 チェックポイント (5): 拡張

解を求めるプログラムを、2 つの駅、出発時刻、帰着時刻、面会時間、の 5 つを引数として与えることができるように拡張する。

ヒント：

- プログラムに与えた引数をハンドリングする方法を学ぶ (テキスト p66)。
- これまで作成してきたプログラムで、上記の 5 つの情報を使用しているところを把握し、それらを変数化する。

2.6 チェックポイント (6): 面会駅と面会時間の表示

もし、答が存在する場合は、その金額だけでなく、どの駅で、何時何分から何時何分まで会うことができるかを表示するようにする。

このチェックポイントは、余力がある場合に取り組むべきものとする。(必須ではない。)

ヒント：

- 最も安い費用の候補が求まったとき、その費用の金額だけでなく、(a) どの駅で、(b) 何時何分から何時何分まで会うことができるか、という情報を記憶する。

- 文献 2 に示されている関数 calc_cost は、抜本的に作り直す必要があるかもしれない。(理由を考えよ！)

- 答が存在した場合は、費用だけでなく、(a) と (b) の情報も出力するように変更する。文献 1 の sample input に対する出力例は、以下のとおり。

```
11000 Morioka: 13:35 - 14:05
0
11090 Morioka: 11:04 - 14:49
```

プログラミング及び演習

4 月 21 日

チェックポイント (1): 列車データの準備

ファイルから列車データを読み込むサブルーチンを作成する。

このサブルーチンが正しく動作することを、次のようなプログラムを作成して確認する。

1. 最も金額が高い列車を求める（出力する）。
2. 最も時間がかかる列車を求める（出力する）。
3. 特定の駅 { から出発する／に到着する } 列車を出力する。

1　攻略法

鉄則：やさしいプログラムから始めて、徐々に、希望の機能を実現していく。

1. 入力ファイルの形式の確認
2. 行単位でファイルを読み込む
3. データ数の行と列車データの行を判別する
4. データセット（ひとまとまりのデータ）を意識して読み込む
5. 列車データ行の解析
6. 列車データの保管
7. 作成したサブルーチンの確認

今週は 4 まで。

1.1　入力ファイルの形式の確認

- 入力ファイルの形式 — 文献 1 の sample input
- 各行の最大の長さは何文字？ — 文献 1 を良く読もう！

1.2 行単位でファイルを読み込む

ファイルから、行単位でデータを読み込み、そのまま出力するプログラムを作る。Unix コマンドの cat のようなもの。

- 入力には、fgets (教科書 p77) を使う。
- 入力の読み込みは、標準入力 (教科書 p76) から行なう。
- 出力には、fputs (教科書 p77) を使う
- 出力は、標準出力 (教科書 p76) へ書き出す。
- シェルのリダイレクションを使って、ファイルと標準入力を結び付ける。
- 基本的に、fgets を繰り返せばよい。繰り返しには while (教科書 5.1.2) を使う。
- fgets は、ファイルの末尾を読み込んだとき、NULL を返すので、これを使って、繰り返しを終了させる。
- fgets には、読み込んだ行（文字列）を格納するのに十分な記憶領域のアドレス (ポインタ) を、引数として渡す必要がある。
- そのような領域は、文字の配列として確保する。
- ポインタに関しては、教科書の 7 章を参照のこと。

```
1  #include <stdio.h>
2
3  /* 各行ごとに読み込んで、そのまま出力する */
4
5  main(void)
6  {
7    char buf[64];
8
9    while (fgets(buf, sizeof(buf), stdin) != NULL) {
10     fputs(buf, stdout);
11   }
12 }
```

- プログラムが正しく動作するかどうかをチェックする。

```
% a.out < sample.in > sample.out
% diff sample.in sample.out
```

1.3 データ数の行と列車データの行を判別する

- データセットの構成は、データ数 (整数) が書かれた行と、それに続くその行数分の列車データ。
- データ数の行は、

  - 1 行目
  - (1 行目に書かれた数字+1) 行目
  - ...

- データ数の行を読み込んだら、そこに書かれていた行数分だけ、列車データ行を読み込む。これを繰り返すように、プログラムを変更する。
- 読み込んだ行から、整数を一つ取り出すには、sscanf (教科書 p78) を使う。

```
#include <stdio.h>

/* データ数の行と列車データの行を判別する */

main(void)
{
  int count;
  char buf[64];

  count = 0;
  while (fgets(buf, sizeof(buf), stdin) != NULL) {
    if (count == 0) { /* データ数の行 */
      sscanf(buf, "%i", &count);
      printf("%i\n", count);
    }
    else { /* 列車データの行 */
      /*      fputs(buf, stdout);  */
      count--;
    }
  }
}
```

1.4 データセットを意識して読み込む

- 上記のプログラムは、＊ねじれている＊ので、より直接的な形式に書き直す。
  - 「データセットに対して繰り返す」という構造が、プログラムから読めない。
- データセットに対して繰り返すという構造を、プログラムの構造に直接反映させたプログラムに変更する。

```
#include <stdio.h>

/* データセットを意識して読み込む */

main(void)
{
  int count;
```

```
  char buf[64];

  while (1) {
    /* データセットの先頭(データ数)を読む */
    ....

    /* データ全体の終了の判定 */
    ....

    /* データセットの残り(列車データ)を読み込む */
    ....
  }
}
```

今週の演習

- 上記のプログラムを完成させよ。(提出は不要)

まとめ

この回に学んだ内容は、次のとおり。

変数、配列、データ型(文字型、整数型)、関数 fgets、関数 fputs、関数 sizeof、関数 sscanf、関数 printf、標準入力、標準出力、記憶領域のアドレス(ポインタ)、NULL、制御構造 (while, if)、比較演算子 (==, !=)、デクリメント (--)

プログラミング及び演習

4 月 28 日

チェックポイント (1): 列車データの準備

ファイルから列車データを読み込むサブルーチンを作成する。

このサブルーチンが正しく動作することを、次のようなプログラムを作成して確認する。

1. 最も金額が高い列車を求める（出力する）。
2. 最も時間がかかる列車を求める（出力する）。
3. 特定の駅 { から出発する／に到着する } 列車を出力する。

1　攻略法

鉄則：やさしいプログラムから始めて、徐々に、希望の機能を実現していく。

1. 入力ファイルの形式の確認
2. 行単位でファイルを読み込む
3. データ数の行と列車データの行を判別する
4. データセット（ひとまとまりのデータ）を意識して読み込む
5. 列車データ行の解析
6. 列車データの保管
7. 作成したサブルーチンの確認

今週は 5 から。

1.4　先週の演習の解答

```
1  #include <stdio.h>
2
3  /* データセットを意識して読み込む */
4
5  main(void)
6  {
7    int count;
```

```
  char buf[64];

  while (1) {
    /* データセットの先頭(データ数)を読む */
    if (fgets(buf, sizeof(buf), stdin) == NULL) {
      break; /* データセットの異常終了 */
    }
    sscanf(buf, "%d", &count); /* データセット中のデータ数 */
    printf("%d\n", count); /* データ数の書き出し */

    if (count == 0) { /* データ全体の終了の判定 */
      break;
    }

    /* データセット(列車データ)を読み込む */
    while (count-- > 0) {
      if (fgets(buf, sizeof(buf), stdin) == NULL) {
        break; /* データセットの異常終了 */
      }
      /* printf("%s", buf); */ /* 列車データの書き出し */
    }
  }
}
```

1.5 列車データ行の解析

- sscanf を使って、列車データ行から、各データを切り出す。

```
/* 列車データの解析 */
parse_connection(char *buf)
{
  char from[18], to[18];
  int  dpt[2], arv[2], fare;

  sscanf(buf, "%s %d:%d %s %d:%d %d",
         from, &(dpt[0]), &(dpt[1]), to, &(arv[0]), (&arv[1]), &fare);

  printf("%s %d %s %d %d\n",
         from, dpt[0]*60+dpt[1], to, arv[0]*60+arv[1], fare);
}
```

1.6 列車データの記憶

- 文献 2 に従って、列車データをプログラム中に記憶する。

  - （ひとつの）列車データは、構造体（教科書第 8 章）を使って表現（記憶）する。
  - 一連の列車データは、その配列を利用して記憶する。

- 駅名は、プログラム内部では、駅 ID によって表現する。
  - 読み込み時に対応表を作成する。

```
#include <stdio.h>
#include <string.h>

#define MAXCITY 100

/* 駅名の表 */
char city_name[MAXCITY][18];
int ncity; /* 駅数 */

#define MAXCONN 2000+1

/* 列車の情報 */
struct train {
  int from, to; /* 駅番号 */
  int dpt, arv; /* 0:00 からの分単位 */
  int fare;
} trains[MAXCONN];

int nconn; /* 列車数 */

...

parse_connection(char *buf)
{
  char from[18], to[18];
  int  dpt[2], arv[2], fare;

  sscanf(buf, "%s %d:%d %s %d:%d %d",
         from, &(dpt[0]), &(dpt[1]), to, &(arv[0]), (&arv[1]), &fare);

  trains[nconn].from = city_id(from);
  trains[nconn].to   = city_id(to);
  trains[nconn].dpt  = dpt[0]*60+dpt[1];
  trains[nconn].arv  = arv[0]*60+arv[1];
  trains[nconn].fare = fare;
  nconn++;

}

/* 駅名 -> 駅番号 */
```

```
41 int city_id(char *name)
42 {
43   int i;
44
45   for (i = 0; i < ncity ; i++) {
46     if (strcmp(name, &(city_name[i][0])) == 0) {
47       return i;
48     }
49   }
50
51   strcpy(&(city_name[ncity][0]), name);
52   return ncity++;
53 }
```

1.7 作成したサブルーチンの確認

- チェックポイント (1) の達成。
  - 必要ならば、サブルーチン化を行なう。
  - 各データセットに対して、以下を実行するプログラムを作成する。
    1. 最も金額が高い列車を出力する。
    2. 最も時間がかかる列車を求める（出力する）。
    3. 特定の駅 { から出発する／に到着する } 列車を出力する。

今週の演習課題

- 各データセットに対して、最も時間がかかる列車を出力するプログラムを作成し、sample input に対して実行した結果を求めよ。
  - sample input に含まれるデータセットは 3 つなので、出力は 3 行となる。
    - subject は、Kadai0428
    - 本文は、プログラムの出力 (3 行)
  - 締切は、5 月 10 日 17 時。

まとめ

この回に学んだ内容は、次のとおり。

制御構造 (break)、構造体、構造体の配列、マクロ、string.h、関数 strcmp、関数 strcpy、デクリメント (++)

プログラミング及び演習

5 月 12 日

チェックポイント (2): fr_h を求める

文献 2 に基づいて，fr_h を求めるサブルーチンを作成する．

このサブルーチンが正しく動作することを，次のようなプログラムを作成して確認する．

1. 函館を起点とした fr_h の表 (文献 2 の p194 の表) を出力する．
2. 東京を起点とした fr_t の表を出力する．

1 問題の考え方

1.1 fr_h・fr_t・to_h・to_t の定義

文献 2 p.193「考え方」参照.

$$
\begin{aligned}
fr\_h(st,t) &= \text{午前 8 時以降に函館を出発して，時刻 } t \text{ に駅 } st \text{ にいるための最小金額} \\
fr\_t(st,t) &= \text{午前 8 時以降に東京を出発して，時刻 } t \text{ に駅 } st \text{ にいるための最小金額} \\
to\_h(st,t) &= \text{時刻 } t \text{ に駅 } st \text{ にいた後，午後 6 時までに函館に帰着するための最小金額} \\
to\_t(st,t) &= \text{時刻 } t \text{ に駅 } st \text{ にいた後，午後 6 時までに東京に帰着するための最小金額}
\end{aligned}
$$

総合課題の目的は，駅 st において 2 人が時刻 t から 30 分会うための総費用 $cost(st,t)$ を最小とする st と t を求めること．

$$
cost(st,t) = fr\_h(st,t) + fr\_t(st,t) + to\_h(st,t+30) + to\_t(st,t+30)
$$

1.2 時刻の離散化

- 文献 2 p.193「fr_h を求める」参照.
- ある駅までの到達費用に変化が生じるのは，その駅に列車がついた時だけ.
  全ての列車を到着時刻順に並べて，$T_0, T_1, \ldots, T_{nt-1}$ とする．ただし，nt は列車の総数．列車 T_i の到着時刻を A_i，8:00 を A_{-1} とする．
  連続的な時刻 t の代わりに，離散的な列車到着事象の番号 (T_i の添字 i) を用いることができる．

1.3 fr_h の再帰的定義

文献 2 p.193「fr_h を求める」，p.194「fr_h の再帰的定義」参照.

- 8:00 における値 $fr\_h(st, -1)$
  - すでに函館にいるので 0 円
  - その他の駅では無限大
- 番号 i の列車 T_i の着駅 TO_i が st でなければ，番号 $i-1$ の値 $fr\_h(st, i-1)$ から変化なし
- 番号 i の列車 T_i の着駅 TO_i が st と一致していれば，以下の小さい方
  - 番号 $i-1$ の値 $fr\_h(st, i-1)$
  - 番号 $change(i)$ を，列車 T_i に乗り継げる最も近い列車の着時刻 A_j の添字 j とすると，

    番号 $change(i)$ の時刻に列車 T_iの発駅 FR_iにいる最小金額 $fr\_h(FR_i, change(i))$

    と

    列車 T_iの運賃 $FARE_i$

    を足したもの

    $$FARE_i + fr\_h(FR_i, change(i))$$

$$fr\_h(st,i) \quad = \quad \begin{cases} 0 & i=-1,\ st=\text{函館} \\ \infty & i=-1,\ st\neq\text{函館} \\ fr\_h(st,i-1) & i\geq 0,\ st\neq TO_i \\ \min[\ fr\_h(st,i-1),\ FARE_i + fr\_h(FR_i, change(i))\ ] & otherwise \end{cases} \tag{1}$$

1.4 動的計画法

- **最適性の原理**: 最適解は，その一部分だけに注目しても，それ自身が (対応する小問題の) 最適解となっている.
- **動的計画法**: 最適性の原理が成り立つ場面において，小問題の最適解を積み重ねることによって大問題の最適解を作ることにより，最適化問題を解く技法.
- $fr\_h(st, i)$ の計算における動的計画法の適用
  - 番号 i の時刻にある駅 ST にいるための最小金額 $fr\_h(ST, i)$ の計算においては，番号 i より前の時刻 $j = -1, 0, 1, \ldots, i-1$ における，あらゆる駅 st での $fr\_h(st, j)$ の値が求まっていれば，式 (1) の一回の計算で $fr\_h(ST, i)$ が求まる.
  - $i = 0, 1, \ldots, nt-1$ の順に $fr\_h(st, i)$ を計算していけばよい.

2 実装法

1. データ構造
2. 列車データの準備
3. 表の作成
4. 作成したサブルーチンの確認

2.1 データ構造

- ライブラリ関数 qsort を使うために，ライブラリのヘッダファイル stdlib.h を取り込む (教科書 6.3.3 節).
- 駅 st，列車到着事象の番号 i における表の値 $fr\_h(st, i)$，あるいは，$fr\_t(st, i)$ を格納するために二次元配列 (教科書 5.2 節) を用いる.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAXCITY 100

/* 駅名の表 */
char city_name[MAXCITY][18];
int ncity; /* 駅数 */

#define MAXCONN 2000+1

/* 列車の情報 */
struct train {
  int from, to; /* 駅番号 */
  int dpt, arv; /* 0:00 からの分単位 */
  int fare;
} trains[MAXCONN];

int nconn; /* 列車数 */

#define INFINITE 99999999

int from_hakodate[MAXCITY][MAXCONN],  /* fr_h */
  from_tokyo[MAXCITY][MAXCONN];       /* fr_t */
```

2.2 列車データの準備

- 列車の配列 trains 中の各列車を到着時刻の早い順に並べ替えるために，ライブラリ関数 qsort を用いる.
- qsort で用いる大小比較関数 cmp_arv の引数は，train 型構造体へのポインタ (教科書 8.3 節)．大小比較関数の返す値の正負により，比較対象の大小を判定.

```
/* 列車到着時刻の大小比較関数の定義 */
int cmp_arv(struct train *t1,
            struct train *t2)
{
  return (t1->arv - t2->arv);
```

```
}

void prepare_data(void)
{
  /* 到着時刻の早い順に列車の配列の要素を並べ替え */
  qsort(/* 並べ替え対象の配列 */ trains,
        /* 配列の要素数 */ nconn,
        /* train 型構造体の大きさ */ sizeof(struct train),
        /* 並べ替えで用いる大小比較関数 */ cmp_arv);
}
```

2.3 表の作成

```
/* 列車の配列 tv 中で，dptime 以前に駅 st に到着する列車を探す
   p: 探索を開始する列車到着事象の番号                    */
int change(struct train tv[], int p, int st, int dpttime)
{
  while (p >= 0) {
    if ((tv[p].to == st) &&
        (tv[p].arv <= dpttime)) break;
    p--;
  }
  return p;
}

/* 列車の配列 tv を参照して，駅 org を起点として，
   fr_h または fr_t を二次元配列 v に作成する */
void make_table(int v[MAXCITY][MAXCONN], int org, struct train tv[])
{
  int ti;
  int i;
  int a;

  for (i = 0; i < ncity ; i++) {
    v[i][0] = INFINITE;               /* 式 (1) 2 行目 */
  }
  v[org][0] = 0;                /* 式 (1) 1 行目 */

  /* 列車の到着時刻の早い順に繰返し */
  for (ti = 0; ti < nconn ; ti++) {
    for (i = 0; i < ncity ; i++) {
      v[i][ti+1] = v[i][ti];    /* 式 (1) 3 行目 */
    }

    /* 列車の配列 tv 中で，tv[ti].dpt 以前に駅 tv[ti].from に到着する列車を探す
```

```
33          = 列車 tv[ti] に乗り継げる列車を探す
34          ti-1: 探索を開始する列車到着事象の番号  */
35      a = change(tv, ti-1, tv[ti].from, tv[ti].dpt);
36
37      /* 式 (1) 4 行目 */
38      v[tv[ti].to][ti+1] = min(v[tv[ti].to][ti],
39                               tv[ti].fare + v[tv[ti].from][a+1]);
40    }
41  }
```

2.4 作成したサブルーチンの確認

- チェックポイント (2) の達成。
  - 以下の表を出力するために，main 関数においてサブルーチン make_table を呼び出すプログラムを作成し，各データセットに対して実行する．
    1. 函館を起点とした fr_h の表
    2. 東京を起点とした fr_t の表

今週の演習課題

- 各データセットに対して，函館を起点とした fr_h の表，および，東京を起点とした fr_t の表を出力するプログラムを作成し，sample input に対して実行した結果を求めよ．
  - sample input に含まれるデータセットは 3 つなので、出力は 3 つの表 $\times 2$ となる。
    - subject は、Kadai0512
    - 本文は，プログラムの出力
  - 締切は、5 月 18 日 17 時。

まとめ

この回に学んだ内容は、次のとおり。

stdlib.h，関数 qsort，二次元配列，構造体へのポインタ，動的計画法

プログラミング及び演習

5月19日

チェックポイント (3): to_h を求める

文献 2 に基づいて,「フィルムの逆回し」のアイディアを使って，to_h を求めるサブルーチンを作成する.

このサブルーチンが正しく動作することを，次のようなプログラムを作成して確認する.

1. 函館を着点とした to_h の表を出力する.
2. 東京を着点とした to_t の表を出力する.

1 実装法

1. データ構造
2. 列車データの準備
3. 作成したサブルーチンの確認

1.1 データ構造

- 「フィルムの逆回し」のアイディアに基づいて，時間軸を逆転した列車データを格納する配列 `rtrains` を用いる.
- 駅 st，列車到着事象の番号 i における表の値 $to\_h(st, i)$，あるいは，$to\_t(st, i)$ を格納するために二次元配列を用いる.

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <stdlib.h>
4
5  #define MAXCITY 100
6
7  /* 駅名の表 */
8  char city_name[MAXCITY][18];
9  int ncity; /* 駅数 */
10
11 #define MAXCONN 2000+1
12
13 /* 列車の情報 */
```

```
struct train {
  int from, to; /* 駅番号 */
  int dpt, arv; /* 0:00 からの分単位 */
  int fare;
} trains[MAXCONN], rtrains[MAXCONN];

int nconn; /* 列車数 */

#define BIAS 24*60   /* 24h*60m */

#define INFINITE 99999999

int from_hakodate[MAXCITY][MAXCONN],  /* fr_h */
  from_tokyo[MAXCITY][MAXCONN],       /* fr_t */
  to_hakodate[MAXCITY][MAXCONN],      /* to_h */
  to_tokyo[MAXCITY][MAXCONN];         /* to_t */
```

1.2 列車データの準備

- 「フィルムの逆回し」を実現するために，列車の配列 trains に対して，
  - 始発駅と終点の交換
  - 時間軸の反転と発時刻・着時刻の反転

  を行なった列車の配列 rtrains を用意する．

```
/* 列車到着時刻の大小比較関数の定義 */
int cmp_arv(struct train *t1,
            struct train *t2)
{
  return (t1->arv - t2->arv);
}

void prepare_data(void)
{
  int i;

  for (i=0; i<nconn; i++) {
    rtrains[i].from = trains[i].to;                /* 始発駅と終点の交換 */
    rtrains[i].to   = trains[i].from;                /* 終点と始発駅の交換 */
    rtrains[i].dpt  = BIAS - trains[i].arv;        /* 時間軸反転と発時刻・着時刻の交換 */
    rtrains[i].arv  = BIAS - trains[i].dpt;        /* 時間軸反転と着時刻・発時刻の交換 */
    rtrains[i].fare = trains[i].fare;                /* 運賃のコピー */
  }

  /* 到着時刻の早い順に列車の配列の要素を並べ替え */
```

```
  qsort(/* 並べ替え対象の配列 */ trains,
        /* 配列の要素数 */ nconn,
        /* train 型構造体の大きさ */ sizeof(struct train),
        /* 並べ替えで用いる大小比較関数 */ cmp_arv);

  /* 到着時刻の早い順に列車の配列の要素を並べ替え */
  qsort(/* 並べ替え対象の配列 */ rtrains,
        /* 配列の要素数 */ nconn,
        /* train 型構造体の大きさ */ sizeof(struct train),
        /* 並べ替えで用いる大小比較関数 */ cmp_arv);
}
```

1.3 作成したサブルーチンの確認

- チェックポイント (3) の達成。
  - 以下の表を出力するために，main 関数においてサブルーチン make_table を呼び出すプログラムを作成し，各データセットに対して実行する．
    1. 函館を着点とした to_h の表
    2. 東京を着点とした to_t の表

今週の演習課題

- 各データセットに対して，(チェックポイント (2) の課題とあわせて)，函館を起点とした fr_h の表，東京を起点とした fr_t の表，函館を着点とした to_h の表，東京を着点とした to_t の表を出力するプログラムを作成し，sample input に対して実行した結果を求めよ．
  - sample input に含まれるデータセットは 3 つなので、出力は 3 つの表 $\times 4$ となる。
    - subject は、Kadai0519
    - 本文は，プログラムの出力
  - 締切は、5 月 25 日 17 時。

プログラミング及び演習

5 月 26 日

1 チェックポイント (4): 解を求める

これまで作成してきたサブルーチンを組み合わせて，解（$cost(st,t)$ の最小値）を求めるプログラムを作成する．

このプログラムが正しく動作することを，文献 1 のデータ (sample input) に対して適用し，sample output と同じ出力が得られることを確認する．

1.1 $cost(st,t)$ の計算

前回までの講義で，$cost(st,t)$ の計算に必要な表 fr_h, to_h, fr_t, to_t の作成方法について説明を行った．今回は，これらの作成した表を使って $cost(st,t)$ を計算する方法について説明する．

チェックポイント (2) で説明があったように，このプログラムでは次の関数の最小値を見つければよい．

$$cost(st,t) = fr\_h(st,t) + fr\_t(st,t) + to\_h(st,t+30) + to\_t(st,t+30) \tag{1}$$

fr_h などからこれを計算すればよいが，注意すべき点は列車の到着・発車時刻が離散化されていることである．到着時刻 a を指定したときに，このちょうど 30 分後の時刻は表には無いことがあり，30 分以上を満たす最小の離散時刻を探さなければならない．したがって，文献 2 のプログラムの関数 `calc_cost` 内では，到着時刻 a と発車時刻 d を 30 分以上という条件を満たしながら最短の間隔を開けて並行に動かしている．

```
1  int calc_cost(int city)
2  {
3    int a=0, d=nconn-1;
4    int stay;
5    int c, min_c=INFINITE;
6
7    while(1){
8      stay = (BIAS - rtrains[d].arv) - trains[a].arv;
9      if(stay < 30){
10       d--;    /* stay が 30 分未満ならば駅 st を出発する時刻 d を後ろへずらす */
11       if(d < 0){
12         break;
13       }
14       continue; /* while ループ内の先頭に戻る */
15     }
16     c = from_hakodate[city][a+1] +
17       from_tokyo[city][a+1] +
```

```
      to_hakodate[city][d+1] +
      to_tokyo[city][d+1];   /* コストの計算 */
    if(c < min_c){
      min_c = c;
    }
    a++;  /* 駅 st への到着時刻を後ろへずらす */
    if(a >= nconn){
      break;
    }
  }
  return min_c;
}
```

1.2 最小の $cost(st,t)$ を探し出す

どの駅で会うか (st)，および何時から会うか (t) を動かしながら，最小の $cost(st,t)$ を探し出す．式で書けば以下のようになる．

$$min\_cost = \min_{st} \min_{8:00 \le t \le 17:30} cost(st,t) \tag{2}$$

これは，チェックポイント (1) で行った「最も金額が高い，最も時間がかかる … 」を探し出す問題と同様である．今は，最も単純なやり方として，すべての可能な (st,t) について調べることにする．ただし，動かす変数が 2 つあるため，

1. 目的駅 st をひとつ決めたときに全ての到着時刻 t を調べ，最小の $cost(st,t)$ を見つける
2. 1 を全ての駅 st について調べる

という手順を踏むことにする．文献 2 では，1 は関数 `calc_cost` の中で，2 は関数 `solve` の中で行っている．もちろんひとつの関数でこれらの処理を行ってもよいし，もっと細かい処理で関数に分けてもよい．どの処理をまとめて関数にするかは非常に重要である．まとめすぎてもよくないし，分割しすぎてもよくない．プログラムの見通しをよくする，すなわち**他人や未来の自分がそのプログラムを読んだときに，手順の流れを容易に理解できる**[1] ようにするためには，ちょうどいいまとまりでサブルーチン化することが重要であるが，そのための指針としては，

- ひとつの関数がひとつの処理を行っているか
- あとで関数を再利用できるか

などを考えて関数を作っていくとよい．

```
int solve(void)
{
  int c, a_i;
  int cost, min_cost;

  prepare_data();
  make_table(from_hakodate, city_id("Hakodate"), trains);
  make_table(from_tokyo, city_id("Tokyo"), trains);
```

[1] 未来の自分は他人であるという言葉があるが，過去に自分が書いたプログラムを読むとこれを実感することができる．

```
  make_table(to_hakodate, city_id("Hakodate"), rtrains);
  make_table(to_tokyo, city_id("Tokyo"), rtrains);

  min_cost = INFINITE;
  for(c=0; c < ncity; c++){
    cost = calc_cost(c);
    if(cost < min_cost){
      min_cost = cost;
    }
  }
  if(min_cost >= INFINITE){
    min_cost = 0;
  }
  return min_cost;
}
```

1.3 これまで作成したサブルーチンをくみ上げる

これまで作ってきたサブルーチンをくみ上げる際のプログラムの全体像を以下に示す．この説明における ひとつの処理がそれぞれひとつの関数に対応していることが分かるだろう．

1. *main* 関数ではひとつの時刻表のセットを読み込むたびに *solve* を呼び出す
2. *solve* 関数では
   (a) 読み込んだデータの整列など（表を作る準備）行う（prepare_data）
   (b) 表の作成を行う（make_table）
   (c) 駅 st を変えながら最小の $cost(st, t)$ を探す（calc_cost）
3. *calc_cost* 関数では
   (a) 駅 st は固定しておき，到着時間 t を変えながら最小の $cost(st, t)$ を探す

サブルーチンをくみ上げていくときの注意点として，関数で計算した値の取り出し方を以下で説明する．

関数で計算した値の取り出し方

関数内で計算した値を外に取り出すには，

- 関数の戻り値を用いる
- 関数の引数に，変数へのアドレスをポインタで渡す
- グローバル変数を用いる

などがある．ひとつの変数を取り出したいならば，戻り値を用いる方法が分かりやすい．

```
int i;
i = func();
```

複数の変数を取り出したい場合などは，関数の引数に変数のアドレスをポインタで渡す必要がある．

```
int i, j;
func(&i, &j);
```

渡していく変数が多くなる場合は，構造体を利用して変数をまとめるのがよい．もちろん，構造体を引数や戻り値にする方法もあるが，内部に配列を持つような大きな構造体では，このようにポインタで渡すほうが効率がよい[2]．

グローバル変数を利用すれば，関数内でも関数を呼ぶ側でも変数にアクセスできる．

```
void calc_min(void); /* プロトタイプ宣言 */
int min_f; /* グローバル変数 */

int main()
{
  calc_min(); /* 最小値 min_f を計算する */
  printf("min: %d\n", min_f);
  return 0;
}
```

ただし，グローバル変数の利用はできるだけ避けるべきである[3]．グローバル変数を用いずに，関数に変数のアドレスをポインタで渡すようにすると，先のコードは次のようになる．

```
void calc_min(int *fp); /* プロトタイプ宣言 */

int main()
{
  int min_f;
  calc_min(&min_f);
  printf("min: %d\n", min_f);
  return 0;
}
```

【注意】本格的なプログラムを作る場合は，from_hakodate などの表もグローバルな変数の配列ではなくローカル変数にしておき，必要/不要になったときにメモリ確保/開放を行うべきである．しかし，これは malloc/free を多用してプログラムが煩雑になるため，今回ぐらいの規模のプログラムや，ちょっとテストをしたい場合などはしばしばグローバル変数を用いる．

[2]変数を関数の引数や戻り値で受け渡しを行うときに，見えないところでコピーが行われているからである．変数そのものを渡す場合を「値渡し」，変数そのものでなくそのアドレスをポインタで渡す場合を「アドレス渡し」と呼ぶことがある．アドレス渡しの場合は，アドレス値のコピーだけで済むため高速である．

[3]ある関数を別のプログラムで流用したい場合がよく起こる．このとき，関数の中でグローバル変数を使っていると，必要なグローバル変数をもれなく別のプログラムでも宣言しておく必要がある．ところが，必要なグローバル変数が多い場合は，全て探すのが面倒であり，さらに流用先のプログラムで変数名が衝突する危険性が高くなる．したがって，再利用性を高くするには，あるルーチンでのみ用いる変数をローカル変数とし，他のルーチンからアクセスできないようにするべきである．

2　チェックポイント (5): 拡張

解を求めるプログラムを，2 つの駅，出発時刻，帰着時刻，面会時間，の 5 つを引数として与えることができるように拡張する．

いままでは，駅 1: Hakodate, 駅 2: Tokyko, 出発時刻: 8:00, 帰着時刻: 18:00, 面会時間: 30 分を固定していた．これをプログラムの実行時に指定できるようする．

プログラムへ具体的なデータを埋め込むことは，できるだけ避けることが望ましい．例えば面会時間が最低 30 分でなく 60 分にしようと思ったときに，もう一度コンパイルしなおす必要があるからだ．このように,「処理」と「データ」を分ける代表的な方法としては,

- プログラムのコマンドライン引数でデータを渡す
- プログラム内でデータの入った表を読み込む

などがあるが，ここでは引数を用いてデータを渡すことにする．これは，今までのプログラムを以下のように修正することになる．

1. プログラムにコマンドライン引数で情報を渡せるように修正する．
2. 上記の 5 つの情報を変数化しておく．
3. 渡した引数を必要な型に変換し変数へ代入する．
4. `parse_connection` で出発時刻・帰着時刻を指定し，時間外の列車情報は，配列 `trains` に入れないように修正する．
5. `calc_cost` で面会時間を指定できるように修正する．到着列車に対応する出発列車を探す時に用いる．
6. `solve` でも引数で面会時間を渡せるようにする．

2–6 はこれまでの知識で拡張することが可能である．したがって，以下では 1 についてのみ説明を行う．基本的なコマンドライン引数の渡し方について述べた後，必要な知識として

- 引数を解析する際の定石
- 文字列から必要な型への変換

を説明する．

2.1　プログラムのコマンドライン引数（基礎）

テキスト p.66 を参考にして次のプログラムの動作を予想せよ．実際に次のプログラムを作成し，コマンドラインで引数を渡して実行せよ．

```
1  int main(int argc, char *argv[])
2  {
3    int i;
4    printf("num of arguments: %d\n", argc);
5    for(i=0; i < argc; i++){
6      printf("argv[%d] : %s\n", i, argv[i]);
7    }
8
9    return 0;
10 }
```

```
./prog abc hoge xyz
```

【注意】argv[0] にはプログラムの実行ファイル名自身が入る．argv[1] 以降に引数が文字列として入る．

2.2 引数解析の定石

プログラムへの引数の渡す方法は，仕様が決まっている場合はそれに従う必要があるが，今回は自由に決めてよい．よく用いられる渡し方は次の 2 つである[4]．

1. 順番が決まっている場合

最も単純な方法．簡単なプログラムを作る際にはよく用いられる．

```
./prog 駅 1 駅 2 出発時刻 帰着時刻 面会時間
```

（例 1-1）引数の個数が固定しているときは次のように引数を解析できる．

```
1  if(argc != 6){
2    fprintf(stderr, "usage : .... ");
3    return 1;  /* 慣例で，0 以外を return すると正常終了しなかったことを意味する */
4  }
5  (第 1 引数 (駅 1) を変数へ代入)
6  (第 2 引数 (駅 2) を変数へ代入)
7  ....
```

ここでは，fprintf を標準エラーにメッセージを出力するために用いている[5]．

（例 1-2）後の引数を省略したい場合は次のように引数を解析できる（switch でも可）．

```
1  for(i=1; i < argc; i++){
2    if(i==1){
3      (第 1 引数 (駅 1) を変数へ代入)
4    }else if(i==2){
5      (第 2 引数 (駅 2) を変数へ代入)
6    }else if
7      ...
8    }else{
9      fprintf(stderr, "usage : ...");
10     return 1;
11   }
12 }
```

[4] man というコマンドを用いて普段つかってるコマンドの引数を確認してみると参考になる．省略できる引数は [] で囲まれている．引数の指定の仕方はプログラマによって好みがあるようである．ちなみに，GNU 系のコマンドはある程度引数の指定の仕方を共通にしている．

[5] printf でもよいが，printf では標準出力（通常はコンソール）にメッセージが表示される．エラーメッセージの表示先を標準出力でなく標準エラーにしておくことで，エラー出力とプログラムの処理結果の出力を分離できるため，fprintf を用いて出力先を切り替えることが一般的である．

2. 順序を入れ替えられるようにする場合

オプションによって引数を指定することで，引数で渡す順序を自由に変えられるようになる．最もよく用いられる方法で，UNIX のコマンドもほとんどがこのタイプである．

```
./prog -a 駅 1 -m 面会時間 -b 駅 2
```

```
for (i=1; i < argc; i++) {
  if (argv[i][0] == '-') {
    switch (argv[i][1]) {
    case 'a':
      i++;
      (argv[i] (駅 1) を変数へ代入)
      break;
    case 'b':
      i++;
      (argv[i] (駅 2) を変数へ代入)
      break;
    case 'm':
      break;
    ...
    default:
      fprintf(stderr, "usage : ...");
      return 1;
    }
  }
}
```

【注意】(例 1-2) や (例 2) のように引数で全ての情報を指定するとは限らない場合は，引数の解析に際だって，あらかじめデフォルト値を変数へ代入しておく必要がある．

2.3 文字列から必要な型への変換

文字列から整数型などの別の型への変換はよく用いられるため，ここでまとめておく．

sscanf を用いる方法

```
  char str[] = "30";
  int i;
  float f;
  double d;
  sscanf(str, "%d", &i);  /* int 型の変数 i に 30 が入る */
  sscanf(str, "%f", &f);  /* float 型の変数 f に 30 が入る */
  sscanf(str, "%lf", &d); /* double 型の変数 d に 30 が入る */
```

チェックポイント (1) で説明があったように，ひとつの文字列から一度に型変換を行うこともできる．

```
char str[] = "18:00";
int h, m;
sscanf(str, "%d:%d", &h, &m);
```

atoi/atof などを用いる方法

```
int i;
float f;
double d;
i = atoi(argv[1]);
f = atof(argv[1]);
d = atof(argv[1]);
```

ただし，変換元の文字列はひとつの数値であること．ascii 文字列から integer や floating point 型に変換するため，このような関数名になっている．

2.4 文字列への変換

参考までに数値や文字列から文字列への変換・コピーをまとめておく．

sprintf を用いる方法

非常に汎用的なのでよく用いる．ただし，バッファオーバーフローが起こるため，文字列のサイズを指定できる snprintf の方を用いることが推奨されている．

```
char str[256];
int i = 30;
float f= 30;
double d = 30;
sprintf(str, "%d", i); /* str[0]='3',str[1]='0',str[2]='\0' */
sprintf(str, "%f", f);
sprintf(str, "%f", d);
```

printf と同じように書式で指定するので，一度に複数の値から文字列を作ることも可能．

```
char str[256];
char name[] = "cpro";
int year = 9;
sprintf(str, "I am %s, %d yeas old", name, year);
```

strcpy を用いる方法

```
char str_dst[256];
char str_src[] = "ab";
strcpy(str_dst, str_src);  /* str_src が str_dst へコピーされる */
```

3　チェックポイント (6): 面会駅と面会時間の表示

もし，答えが存在する場合は，その金額だけでなく，どの駅で，何時何分から何時何分まで会うことができるかを表示するようにする．

このチェックポイントは，余力がある場合に取り組むべきものとする（必須ではない）．
ヒント：

- 最も安い費用の候補が求まったとき，その費用の金額だけでなく，(a) どの駅で，(b) 何時何分から何時何分まで会うことができるか，という情報を記憶する．
- 文献 2 に示されている関数 `calc_cost` は，抜本的に作り直す必要があるかもしれない．（理由を考えよ！）
- 答えが存在した場合は，費用だけでなく，(a) と (b) の情報も出力するように変更する．文献 1 の sample input に対する出力例は，以下の通り．

```
11000 Morioka: 13:35 - 14:05
0
11090 Morioka: 11:04 - 14:49
```

3.1　必要な情報の記憶

最小値 (金額) とともに，最小値を与える変数 (どの駅で，何時何分から何時何分まで) を取り出したい．ちなみに，このことを式で書くと以下のようになる．

$$(min\_st, min\_t) = \arg\min_{st} \min_{8:00 \le t \le 17:30} cost(st, t) \tag{3}$$

参考までに

$$min\_x = \arg\min_{0 \le x < 10} f(x) \tag{4}$$

を求めるプログラムを紹介する．

```
1  void calc_min(void)
2  {
3    int x;
4    float f; /* cost */
5    int min_x;
6    float min_f=10000;
7
8    for(x=0; x < 10; x++){
9      f = (x-3.3)*(x-3.3);
10     if(f < min_f){
11       min_f = f;
12       min_x = x;  /* ここで x も記憶しておく必要がある */
13     }
14   }
15   printf("min : f = %f at x = %d\n", min_f, min_x);
16 }
```

3.2 calc_cost の修正

チェックポイント（5）までで作成した calc_cost をそのまま利用することはできない．calc_cost を修正することもできるが，流れが読みにくい場合は抜本的に作り直してもよい．

4 今回の演習課題

総合課題を解くこと．チェックポイント（5）までは必須だが,（6）は余力があればでよい．

一度に大きなプログラムを作成するよりも，各サブルーチンを作成する毎に，それが正しく動くかをチェックすること．このように，個々のモジュールのみを対象としたテストを単体テストと呼ぶ[6]．これまでの講義で出題された課題は，各サブルーチンの単体テストを含んでいるため，確実にクリアしていくこと．おかしな動作がある場合には，どこまでが正しく動いているかを調べ，**問題の箇所を絞り込んでいくことが非常に重要**である．

以下に，チェックポイント（4）,（5）,（6）をテストするための課題を出題する．これらの問題はテストのごく一部であるため，各自様々な課題を考えてテストを行い，確実に動作することを調べること．必要に応じてデータを自分で作成するとよい．

4.1 チェックポイント（4）：calc_cost のテスト

各駅における最小の費用（calc_cost の戻り値）を出力するように，solve を一時的に修正せよ．特に，sample.txt の 3 番目のデータセットに対しては以下の出力となることを確認せよ．

```
Hakodate: 99999999
Akita: 11700
Morioka: 11090
Tokyo: 12060
```

4.2 チェックポイント（5）：コマンドライン引数の解析のテスト

2 つの駅，出発時刻，帰着時刻，面会時間，の 5 つを引数として与え，変数に代入した後に，これらの値を表示するようにプログラムを修正せよ．

4.3 チェックポイント（5）：拡張のテスト

sample.txt を入力として，以下の値を引数で指定すること．

1. 駅 1: Tokyo, 駅 2: Hakodate, 出発時刻: 8:00, 到着時刻: 18:00, 面会時間: 30 分

2. 駅 1: Tokyo, 駅 2: Hakodate, 出発時刻: 8:00, 到着時刻: 23:00, 面会時間: 30 分

1 を指定したときは，チェックポイント（4）までと同じ結果が得られることを確認せよ．2 を指定したときは，sample.txt の 3 番目のデータセットに対する費用が 9780 円になることを確認せよ．

4.4 チェックポイント（6）のテスト

4.3 節の 2 を指定したときに，2 人が出会う駅が盛岡 (11:04 - 14:05) から秋田 (16:09 - 20:36) に変化することを確認せよ（括弧内は面会時間を表す）．

[6]複数のモジュールを組み合わせたテストは結合テスト，システム全体を対象に行うテストはシステムテストと呼ぶことがある．