2018 年度 情報領域演習第三
— 第 10 回 —

注意事項

- 今回は探索アルゴリズムについて実装を通じて学習する．問題文だけでなく，その前にある説明をよく読み，必要であれば参考書等で自習するとよい．わからないことがあれば TA や教員に質問しよう．
- 演習時間終了時点での提出状況についても評価の対象となるので，時間内にできるだけ多くの問題に挑戦するとよい．また，遅刻や途中退室は記録に残るので注意すること．
- プログラムの形式が指定されている場合には，それに従うこと．従わなくても「成功」と表示されることがあるが，得点にはならない．また，採点の際は `checker` とは異なる入出力例を用いて動作確認するので，「失敗」する反例に対する小手先の対策だけでは得点にならないことがあるので注意せよ．

はじめに

前回同様，CED において以下のコマンド (青字の部分) を実行することで出席を表明できる．ただし，N は所属するクラス名 1，2，3 で置き換え，N の前には空白を入れないこと．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/10/checkerN
提出開始: 12 月 xx 日 14 時 40 分 0 秒
提出締切: 12 月 yy 日 14 時 39 分 0 秒
ユーザ: p1610999, 出席状況: 2018-12-xx-14-58
 問題:    結果 | 提出日時            | ハッシュ値                        |
    1:  未提出 |                     |                                   |
    2:  未提出 |                     |                                   |
                        ⋮
```

出席を表明するには，授業の開始から 30 分以内に実行する必要がある．実行しても出席状況が「欠席」である場合は教員に伝えること．なお，上の出力は例であるので，実際の締切りは各自コマンドで確認せよ．

提出も同じコマンドを用いて以下のように実行する．ただし，N はクラス名 (1 から 3)，X は 1 から 6 のいずれかの問題番号であり，`prog.c` は提出する C プログラムのファイル名であり，X の前後には忘れずに空白を入れること．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/10/checkerN X prog.c
```

ここで，ファイル名として指定するのは，問題番号 X の問題を解く C プログラムのソースファイルであり，コンパイル済みの実行ファイルではない．ファイル名は特に指定しないが，「英数字からなる文字列 `.c`」などとすることが望ましい．このコマンドを実行することにより，指定されたファイルがコンパイルされ，実行テストプログラムが自動的に起動される．コンパイルに失敗した場合にはエラーとなり，提出されたことにはならないので注意すること．コンパイルが成功した場合には複数回の実行テストが行われ，実行テストにも成功すると「成功」と表示される．実行テストに失敗すると「失敗」と表示されるとともに反例となる入力と出力が表示されるので，失敗した理由を見つけるための参考にするとよい．なお，入出力が大きい場合やファイルとして入力したい場合には共に表示されるパスにあるファイルを使うと便利である．ただし，入力のない問題の場合は失敗しても反例は出力されない．

正しく提出できたか確認するためには以下のように出席の表明と同じコマンドを用いて確認できる．

```
[p1610999@blue00 ~]> /ced-home/staff/18jr3/10/checkerN
提出開始: 12 月 xx 日 14 時 40 分 0 秒
提出締切: 12 月 yy 日 14 時 39 分 0 秒
ユーザ: p1610999, 出席状況: 2018-12-xx-14-47
 問題:    結果 | 提出日時            | ハッシュ値                        |
```

```
  1:    成功 | 2018-12-xx-15-33  | 9b762c81932f3980cf03a768e044e65b |
                        ⋮
```

ハッシュ値は，提出した C プログラムのソースファイルの MD5 値である．CED では `md5sum` コマンドを用いて MD5 値を見ることにより，提出したファイルと手元のファイルが同じものであるかを確認することができる．

```
[p1610999@blue00 ~]> md5sum prog.c
9b762c81932f3980cf03a768e044e65b
```

なお，一度「成功」となった問題に対し，実行テストに失敗するプログラムを送信してしまうと，結果が「失敗」になってしまうので注意すること．

1 データの探索

今回は配列で与えられたデータ列から目的のデータを見つける探索アルゴリズムを実装する．入力となる配列の各要素は以下の構造体で与えられるものとする．

```
struct student { int id; char name[32]; int score; };
```

この構造体 `student` は学生の情報を表し，メンバ `id` は整数値で表される学籍番号，メンバ `name` は文字列で表される名前 (31 文字以下)，メンバ `score` は 0 から 100 までの整数値で表される得点である．

構造体 `student` を要素にもつ配列が `id` について昇順に並んでいるとき，以下の問いに答えよ．

問題 1

与えられた数値と同じ得点をもつ学生のうち，学籍番号が最も若い (小さい) 学生の学籍番号と名前を出力するプログラムを作成せよ．最も若い学籍番号を初めから順番に比較する線形探索を行う必要があることに注意せよ．たとえば，得点が S 点の学生を見つける問題を考えよう．配列の先頭 n 個 (0 番めから $n-1$ 番めまで) の中から線形探索を行うとき，最も単純な方法では添字を表す変数 i を 0 から 1 つずつ増やしながら,「配列の i 番めの学生の得点が S と異なるか，i が n 未満である間」繰り返す．しかしながら，この方法では毎回 $i < n$ を判定するため効率が悪い．配列の n 番めを利用できる場合には以下のようにすると効率がよくなる．

(1) n 番めに得点が S であるような構造体 `student` (番兵という) を入れる．

(2) 配列の i 番めの学生の得点が S と異なる間，i に 1 を足し続ける．

(3) (2) が終わった時点で，i が n であれば得点が S である学生が見つからなかったことになり，$i < n$ であれば学生の得点が S である最初の学生が i 番めであることを表す．

(2) で i に 1 を足し続けているが，$i < n$ を判定していないことに注意しよう．事前に n 番めに得点が S の要素を入れておくことで，$n-1$ 番めまでに得点が S の要素が見つからなかったとしても，$i = n$ になった時点で必ず `while` 文を抜けることができる．線形探索は最短で 1 回で目的の要素を見つけられるが，最悪すべての要素を見ることになるため，このアルゴリズムの時間計算量は $O(n)$ である．

作成すべきプログラムの形式は後に示す `main` 関数を含むものとし，上述のアルゴリズムによって出力を計算する．標準入力から与えられる入力は 2 行以上 1024 行以下で，最後の行を除くすべての行は学籍番号 (5 桁の整数) と名前 (カンマを含まない 31 文字以下の文字列) と得点 (0 から 100 までの整数) をカンマで区切った文字列とし，最後の行は 0 から 100 までの整数と . (ドット) からなる文字列である．最後の行を除き，学籍番号は重複なく昇順に並んでいると仮定してよい．また標準出力には，最後の行の整数と得点が一致する最初の学生の学籍番号と名前をカンマで区切った文字列を出力し，一致する学生がいなかったときは `Not found.` を出力する．

入力例

```
10123,Ichiro Suzuki,51
10345,Shohei Ohtani,17
10456,David Beckham,77
10567,Cristiano Ronaldo,7
17.
```

出力例

```
10345,Shohei Ohtani
```

入力例

```
10123,Ichiro Suzuki,51
10345,Shohei Ohtani,17
10456,David Beckham,77
10789,Cristiano Ronaldo,7
97.
```

出力例

```
Not found.
```

入力例

```
10123,Ichiro Suzuki,51
10234,Makoto Hasebe,17
10456,Shohei Ohtani,17
10789,Cristiano Ronaldo,7
17.
```

出力例

```
10234,Makoto Hasebe
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えること．

```c
#include<stdio.h>

struct student { int id; char name[32]; int score; };

int main() {
  int i=0, n, v;
  char buf[128], c;
  struct student st[1024];
  while(fgets(buf,sizeof(buf),stdin)!=NULL&&i<1024) {
    sscanf(buf, "%d%c", &v, &c);
    if(c==',') {
      st[i].id = v;
      sscanf(buf, "%*d,%[^,],%d", st[i].name, &st[i].score);
      ++i;
    }
  }
  n = i;
  /* この時点で v に探索すべき得点が入っていて，n が読み込んだ要素の数である */
  /* ※ここを適切なプログラムで埋める                                          */
  /*   for 文や while 文において i<n のような添字による終了判定をしないこと   */
  return 0;
}
```

メモ 13 行めの関数 `sscanf` の引数 `"%*d,%[^,],%d"` は，3 つのフォーマット形式 `%*d` と `%[^,]` と `%d` をカンマで区切った形になっている．最初の `%*d` は整数を表す文字列にマッチするが変数に格納しないことを表す (`%*s` なら文字列にマッチするが変数に格納しないことになる)．次の `%[^,]` はカンマ以外の文字からなる文字列にマッチし，その文字列が引数のアドレスに格納される．最後の `%d` は整数を表す文字列にマッチし，その整数が引数のアドレスに格納される．たとえば，`buf` に `10123,Ichiro Suzuki,51` という文字列が入っていれば，13 行めによって，`10123`, は変数に格納されずに，文字列 `Ichiro Suzuki` が `st[i].name` に格納され, 整数 `51` が `st[i].score` に格納される．

問題 2

今度は学籍番号から対応する学生を見つけて名前と得点を出力するプログラムを作成しよう．入力となる配列は学籍番号順に並んでいるため，**二分探索**を用いるのが効率がよい．二分探索は，入力が整列されていることを利用して効率よく探索を行うアルゴリズムで，日常生活においても無意識に行うことがあるかもしれない．昇順に並んでいる配列の l 番め (左端) から r 番め (右端) の中から目的の要素を探す二分探索アルゴリズムは，以下のようにして実現される (見つかった場合はその要素の添字を返し，見つからなかった場合は -1 を返すものとする)

(1) r が l より小さければ，配列の中に目的の要素は存在しないので -1 を返す.

(2) そうでなければ，l と r の平均値 m (小数点以下切り捨て) を計算し，

(i) m 番めが目的の要素なら m を返す.

(ii) m 番めが目的の要素より小さければ，$m+1$ 番めから r 番めを探す.

(iii) m 番めが目的の要素より大きければ，l 番めから $m-1$ 番めを探す.

二分探索を実現する関数は，下線部が示すように同じような手続きで探索を進めているので，再帰関数の考え方を用いれば定義できる．なお，l と r の平均値の小数点以下を切り捨てた値は，C 言語なら `(l+r)/2` で計算できるが，`int` 型のオーバフローを考慮して `l+(r-l)/2` と計算する方が望ましい．二分探索では探索範囲が毎回半分になるため効率がよい．たとえば，配列の 0 番めから 1022 番めまで探すとき，線形探索では最悪 1023 個すべての要素と比較する必要があるが，二分探索では最悪でも 10 個の要素としか比較する必要がない．つまり，二分探索の時間計算量は $O(\log n)$ である．

作成すべきプログラムの形式は後に示す `main` 関数を含むものとし，関数 `binary_search` の定義を埋めることによってプログラムを完成させよ．関数 `binary_search` の引数と返り値の型は以下の通りである：

```
int binary_search(struct student st[], int id, int l, int r);
```

この関数は,「構造体 `student` を要素とする配列 `st` の `l` 番目から `r` 番目の中から，引数の `id` と一致するようなメンバ `id` を持つ要素の添字を見つける関数」であり，一致するものが見つからない場合は `-1` を返す．標準入力から与えられる入力は 2 行以上 1025 行以下で，最後の行を除くすべての行は学籍番号 (5 桁の整数) と名前 (カンマを含まない 31 文字以下の文字列) と得点 (0 から 100 までの整数) をカンマで区切った文字列とし，最後の行は学籍番号を表す 5 桁の整数と `.` (ドット) からなる文字列である．最後の行を除き，学籍番号は重複なく昇順に並んでいると仮定してよい．また標準出力の 1 行めには，入力の最後の行の整数と学籍番号が一致する学生の名前と得点をカンマで区切った文字列を出力し，一致する学生がいなかったときは `Not found.` を出力する．標準出力の 2 行めには，関数 `binary_search` を呼び出した回数も出力する．

入力例

```
10123,Ichiro Suzuki,51
10345,Shohei Ohtani,17
10456,David Beckham,77
10567,Cristiano Ronaldo,7
10456.
```

出力例

```
David Beckham,77
2
```

入力例

```
10012,Hideki Matsui,55
10123,Ichiro Suzuki,51
10234,Makoto Hasebe,17
10345,Shohei Ohtani,17
10456,David Beckham,77
10567,Yuto Nagatomo,55
10789,Cristiano Ronaldo,7
10567.
```

出力例

```
Yuto Nagatomo,55
2
```

入力例

```
10012,Hideki Matsui,55
10123,Ichiro Suzuki,51
10234,Makoto Hasebe,17
10345,Shohei Ohtani,17
10456,David Beckham,77
10567,Yuto Nagatomo,55
10789,Cristiano Ronaldo,7
10777.
```

出力例

```
Not found.
4
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えること．

```
#include<stdio.h>

int count = 0; /* 関数 binary_search を呼んだ回数 */

struct student { int id; char name[32]; int score; };

int binary_search(struct student st[], int id, int l, int r) {
  ++count;
  /* ※ここを適切なプログラムで埋める */
}

int main() {
  int i=0, n, v;
  char buf[128], c;
  struct student st[1024];
  while(fgets(buf,sizeof(buf),stdin)!=NULL&&i<1024) {
    sscanf(buf, "%d%c", &v, &c);
    if(c==',') {
      st[i].id = v;
      sscanf(buf, "%*d,%[^,],%d", st[i].name, &st[i].score);
      ++i;
    }
  }
  n = i;
  /* この時点で v に探索すべき学籍番号が入っていて，n が読み込んだ要素の数である */
  i = binary_search(st, v, 0, n-1);
  if(i < 0) printf("Not found.\n");
  else      printf("%s,%d\n",st[i].name,st[i].score);
  printf("%d\n", count);
  return 0;
}
```

2 ハッシュ法

線形探索や二分探索は入力の各要素と目的の要素と比較しながら比較していたが，ハッシュ法はまったく異なる方法で探索を高速に実現している．各要素に対して「何らかの方法で」配列の添字と対応づけることができれば，探したい要素についても同じ方法で添字を計算してその要素の情報を見つけることが可能になる．たとえば，Ichiro Suzuki という文字列なら 1，Shohei Ohtani という文字列なら 3，Cristiano Ronaldo という文字列なら 7，というような計算をする方法が仮にあれば，それぞれの情報をその添字に格納した配列を用意すればよい．それに対して，Shohei Ohtani を探したければ，同じ方法で文字列から 3 という値を計算して，その添字の要素を見れば目的の情報を見つけることができる．

この「何らかの方法」を実現するのが**ハッシュ関数**である．ハッシュ関数の定義の方法には様々な方法

が考えられるが，今回は ASCII の文字列を 128 進数 (0 文字めを最も上の位とする) と見なして添字を計算することにしよう．たとえば，`uec` という文字列は `u` (ASCII コード [1] で 117 番)，`e` (ASCII コードで 101 番)，`c` (ASCII コードで 99 番) なので，$117 \times 128^2 + 101 \times 128 + 99 = 1929955$ となる．つまり，文字列 $c_0c_1c_2\ldots c_k$ に対して，$\displaystyle\sum_{i=0}^{i=k} c_i \cdot 128^{k-i}$ を計算する．もちろん，この数値をそのまま添字にして使うと非常に大きな配列が必要になってしまうので，この値を配列の大きさで割った余りを添字とする．たとえば，配列の長さが 100 であるとき，文字列 `uec` に対応する添字は 1929955 を 100 で割った余りの 55 となる．したがって，文字列 `uec` に対応する情報を配列の 55 番めに入れておけば，文字列 `uec` に対応する情報を見つけるために同様の計算をして，配列の 55 番めを見るようにすればよい．なお，配列の大きさが小さいときは違う文字列で同じ添字になってしまうことがあるが，この場合の対処は後述する．

問題 3

ハッシュ法に用いる配列の大きさ `SIZE` がマクロで定義されているとき，文字列を 0 番めを上位の桁とするような 128 進数と見なしてハッシュ値を計算するハッシュ関数 `hash` を定義せよ．関数 `hash` の引数と返り値の型は以下の通りである：

```
int hash(char *s);
```

この関数は,「文字列 `s` を 128 進数と見なして `SIZE` で割った余りを計算する関数」である．たとえば，`SIZE` の値が N のとき，文字列 $c_0c_1c_2\ldots c_4$ に対して，

$$\left(\sum_{i=0}^{i=4} c_i \cdot 128^{4-i}\right) \%N = \left(c_0 \cdot 128^4 + c_1 \cdot 128^3 + c_2 \cdot 128^2 + c_3 \cdot 128 + c_4\right) \%N$$

を計算すればよいが，このまま C 言語で計算しようとすると括弧の中が `int` 型の範囲を超えてしまう可能性があることに注意せよ．そこで，代わりに以下のように式変形 [2] したものを利用すればよい．

$$\begin{aligned}\left(\sum_{i=0}^{i=4} c_i \cdot 128^{i}\right) \%N &= ((((c_0 \times 128 + c_1) \times 128 + c_2) \times 128 + c_3) \times 128 + c_4)\%N \\ &= ((((c_0\%N \times 128 + c_1)\%N \times 128 + c_2)\%N \times 128 + c_3)\%N \times 128 + c_4)\%N\end{aligned}$$

こうすると，計算中の整数値が $128N$ を越えることがない [3]．すなわち，文字列が $c_0c_1c_2\ldots c_k$ なら，

$$\begin{cases} S_0 = c_0\%N \\ S_i = (S_{i-1} \times 128 + c_i)\%N \quad (i \geq 1) \end{cases}$$

によって定義される数列の S_k の値を計算すればよい．なお，結果となる添字がなるべくばらつくように N として素数を選ぶことが多い．

作成すべきプログラムの形式は後に示す `main` 関数を含むものとし，関数 `hash` の定義を埋めることによってプログラムを完成させよ．標準入力から与えられる入力は 1 行からなる文字列 (制御文字を除く ASCII 文字からなる 31 文字以内の文字列) で，標準出力にはその文字列に対するハッシュ値 (ハッシュ関数を適用した結果) を出力する．なお，入力とする文字列には最後の改行は含まれないものとする．プログラム内ではハッシュ法における配列の大きさ `SIZE` は 101 としている．また，文字 `c` に対応する ASCII コード (整数値) は `(int)c` とすれば計算できる．

入力例	出力例
uec	47

[1] 半角の英字やアラビア数字，記号・空白文字・制御文字などを規定した ASCII 文字には，ASCII コードという通し番号がついている． コマンド `man ascii` によってその数値を確認することができる．

[2] 等式 $(a+b)\%N = ((a\%N) + (b\%N))\%N$ や $(a \times b)\%N = ((a\%N) \times (b\%N))\%N$ を用いれば変形できる．

[3] 計算結果は N 未満であるが，計算の途中では N より大きくなることがある．ただ，$(\ldots)\%N$ が N 未満で，c_i が 128 未満なので，$(\ldots)\%N \times 128 + c_i$ は $128N$ 未満になる．

入力例

```
Shohei Ohtani
```

出力例

```
84
```

入力例

```
Cristiano Ronaldo
```

出力例

```
34
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えること．

```
1  #include<stdio.h>
2  #define SIZE 101
3
4  int hash(char *s) {
5    /* ※ここを適切なプログラムで埋める */
6  }
7
8  int main() {
9    char s[32];
10   scanf("%[^\n]", s);
11   printf("%d\n", hash(s));
12   return 0;
13 }
```

メモ　10 行めの関数 scanf の引数 "%[^\n]" は，改行以外の文字からなる文字列にマッチし，その文字列が引数のアドレスに格納されることを表す．

問題 4

この問題の目的は，構造体 student を格納するための**ハッシュ表** (hash table) を作ることである．ハッシュ表とは，ハッシュ関数に基づいて高速に参照するためのデータ構造で，C 言語であれば配列を用いて定義できる．たとえば，問題 3 で作成したハッシュ関数を用いる場合は，まず大きさが SIZE であるような配列を用意し，メンバ name が Shohei Ohtani であるような構造体 student であれば，対応する添字である 84 番めに格納すればよい．同様に，メンバ name が Cristiano Ronaldo であれば 34 番めに格納する．このような配列 (ハッシュ表) を用意しておけば，Cristiano Ronaldo の情報を得たいときには，ハッシュ関数を用いて添字を計算して配列の要素を見ればよい．

異なる文字列が同じ添字になってしまう場合はどうするべきだろうか．問題 3 で作成したハッシュ関数は SIZE (101) で割った余りを返すので，0 から 100 までの値しかないため，すべての文字列が異なる添字を返すことは考えにくい．実際，Shinji Kagawa と Yukio Mishima はどちらも 44 を返すため，配列の同じところに格納されることになってしまう．これをハッシュ値の**衝突**といい，いくつかの回避方法が提案されている．今回は，配列の各要素を連結リストで表すことによって複数個のデータを格納する方法を用いる．この問題では配列の各要素を以下の構造体 node (節点) のアドレスとして表現し，

```
struct node {
  struct student data;
  struct node *next;
};
```

メンバ data は構造体 student に対応し，メンバ node は次の節点のアドレスを表す．このような構造体のアドレスを要素とする配列をハッシュ表として用いることで衝突を回避することができる．先ほどの Shinji Kagawa と Yukio Mishima のような例であれば，これらを連結リストとして繋げて配列の 44 番めに格納すればよい．逆にハッシュ表から Shinji Kagawa を探索するときは，ハッシュ値 44 を計算して配列の 44 番めを見て，その連結リストの中から線形探索で探し出せばよい．今回は頭のない連結リストを配列の要素として格納するものとし，何も登録されていない添字の要素には NULL が入っているものとする．

作成すべきプログラムの形式は後に示す main 関数を含むものとし，関数 set_data の定義を埋めることによってプログラムを完成させよ．関数 set_data の引数と返り値の型は以下の通りである．

```
void set_data(struct node *table[], struct student st);
```

この関数は,「ハッシュ表 (構造体 node のアドレスを要素とする配列) table に，構造体 student 型の st を登録する関数」で，問題 3 で定義したハッシュ関数を用いて st のメンバ name から添字を計算し，その添字の要素の連結リストの先頭に st と同じ値を data にもつ節点を追加する．標準入力から与えられる入力は 1 行以上 1024 行以下で，各行は学籍番号 (5 桁の整数) と名前 (カンマを含まない 31 文字以下の文字列) と得点 (0 から 100 までの整数) をカンマで区切った文字列とする．また標準出力には，先頭から順に関数 set_data を用いて登録して得られるハッシュ表に含まれる構造体を，添字の小さい順に (添字が同じものは連結リストの先頭から順に)，それぞれ

(添字) 学籍番号, 名前, 得点

の形式で出力する (プログラムの雛型をそのまま使えばよい).

入力例

```
10123,Ichiro Suzuki,51
10345,Shohei Ohtani,17
10456,David Beckham,77
10567,Cristiano Ronaldo,7
```

出力例

```
(34)10567,Cristiano Ronaldo,7
(64)10123,Ichiro Suzuki,51
(84)10345,Shohei Ohtani,17
(98)10456,David Beckham,77
```

入力例

```
10012,Hideki Matsui,55
10123,Ichiro Suzuki,51
10234,Makoto Hasebe,17
10345,Shohei Ohtani,17
10456,David Beckham,77
10567,Yuto Nagatomo,55
10789,Cristiano Ronaldo,7
```

出力例

```
(34)10789,Cristiano Ronaldo,7
(41)10012,Hideki Matsui,55
(64)10123,Ichiro Suzuki,51
(66)10567,Yuto Nagatomo,55
(73)10234,Makoto Hasebe,17
(84)10345,Shohei Ohtani,17
(98)10456,David Beckham,77
```

入力例

```
10012,Hideki Matsui,55
10123,Ichiro Suzuki,51
10234,Makoto Hasebe,17
10345,Shohei Ohtani,17
10456,David Beckham,77
10567,Yuto Nagatomo,55
10678,Shinji Kagawa,23
10789,Cristiano Ronaldo,7
10890,Yukio Mishima,100
```

出力例

```
(34)10789,Cristiano Ronaldo,7
(41)10012,Hideki Matsui,55
(44)10890,Yukio Mishima,100
(44)10678,Shinji Kagawa,23
(64)10123,Ichiro Suzuki,51
(66)10567,Yuto Nagatomo,55
(73)10234,Makoto Hasebe,17
(84)10345,Shohei Ohtani,17
(98)10456,David Beckham,77
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えること．

```
1  #include<stdio.h>
2  #include<stdlib.h>
3  #define SIZE 101
4
5  struct student { int id; char name[32]; int score; };
6
7  struct node { struct student data; struct node *next; };
8
9  int hash(char *s) {
10   /* ※ここは問題 3 と同じ */
11 }
12
13 void set_data(struct node *table[], struct student st) {
14   /* ※ここを適切なプログラムで埋める */
```

```
15 }
16
17 int main() {
18   int i;
19   char buf[128];
20   struct student st;
21   struct node *table[SIZE], *p;
22   /* ハッシュ表の配列の各要素を NULL で初期化 */
23   for(i=0;i<SIZE;++i) table[i] = NULL;
24   while(fgets(buf,sizeof(buf),stdin)!=NULL) {
25     sscanf(buf, "%d,%[^,],%d", &st.id, st.name, &st.score);
26     set_data(table, st);   /* ハッシュ表に st を登録 */
27   }
28   for(i=0;i<SIZE;++i) {
29     p = table[i];
30     while(p!=NULL) {
31       st = p->data;
32       printf("(%d)%d,%s,%d\n", i, st.id, st.name, st.score);
33       p = p->next;
34     }
35   }
36   return 0;
37 }
```

問題 5

この問題の目的は問題 4 で作成したハッシュ表に対して，特定の名前の学生の情報を見つけ出すことである．具体的には以下の関数を定義する．

```
int find_score(struct node *table[], char *s);
```

この関数は,「ハッシュ表に登録されている構造体 `student` 型の要素のうち，メンバ `name` が `s` であるような学生の得点 (メンバ `score`) を返す関数」で，その名前の学生がいなければ`-1` を返す．アルゴリズムは単純で，まずハッシュ関数を用いて `s` から添字を計算し，その添字にある連結リストからメンバ `name` が `s` であるものを探すだけである．なお，同じ文字列か比較するには関数 `strcmp` を用いてよい．2 つの文字列 `s1`, `s2` に対し，`strcmp(s1,s2)` は，`s1` と `s2` が等しければ `0` を返し，等しくなければ `0` 以外の値を返す．ただし，ハッシュ表の中に同姓同名の学生は含まれていないと仮定してよい．

作成すべきプログラムの形式は後に示す `main` 関数を含むものとし，関数 `find_score` の定義を埋めることによってプログラムを完成させよ．標準入力から与えられる入力は 2 行以上 1025 行以下で，最後の行を除く各行は学籍番号 (5 桁の整数) と名前 (カンマを含まない 31 文字以下の文字列) と得点 (0 から 100 までの整数) をカンマで区切った文字列とし，最後の行は，`0` と名前をカンマで区切った文字列である．また標準出力には，入力の最後の行の名前と一致する学生の得点を出力し，一致する学生がいなかったときは `Not found.` を出力する．

入力例

```
10123,Ichiro Suzuki,51
10345,Shohei Ohtani,17
10456,David Beckham,77
10567,Cristiano Ronaldo,7
0,David Beckham
```

出力例

```
77
```

入力例

```
10012,Hideki Matsui,55
10123,Ichiro Suzuki,51
10234,Makoto Hasebe,17
10345,Shohei Ohtani,17
10456,David Beckham,77
10567,Yuto Nagatomo,55
10678,Shinji Kagawa,23
10789,Cristiano Ronaldo,7
0,Yukio Mishima
```

出力例

```
Not found.
```

入力例

```
10012,Hideki Matsui,55
10123,Ichiro Suzuki,51
10234,Makoto Hasebe,17
10345,Shohei Ohtani,17
10456,David Beckham,77
10567,Yuto Nagatomo,55
10678,Shinji Kagawa,23
10789,Cristiano Ronaldo,7
10890,Yukio Mishima,100
0,Shinji Kagawa
```

出力例

```
23
```

作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えること．

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h> /* 関数 strcmp を使うために必要 */
#define SIZE 101

struct student { int id; char name[32]; int score; };

struct node { struct student data; struct node *next; };

int hash(char *s) {
  /* ※ここは問題４と同じ */
}

void set_data(struct node *table[], struct student st) {
  /* ※ここも問題４と同じ */
}

int find_score(struct node *table[], char *s) {
  /* ※ここを適切なプログラムで埋める */
}

int main() {
  int i, v;
  char buf[128], name[32];
  struct student st;
  struct node *table[SIZE];

  for(i=0;i<SIZE;++i) table[i] = NULL;
  while(fgets(buf,sizeof(buf),stdin)!=NULL) {
    sscanf(buf,"%d,",&v);
    /* 各行の最初の数値によって分岐 */
```

```
32      if(v > 0) {
33        sscanf(buf,"%d,%[^,],%d",&st.id,st.name,&st.score);
34        set_data(table, st);
35      } else
36        sscanf(buf,"0,%[^,\n]",name);
37    }
38    v = find_score(table, name);
39    if(v < 0) printf("Not␣found.\n");
40    else      printf("%d\n", v);
41    return 0;
42  }
```

メモ　36 行めの関数 sscanf の引数 "%[^,\n]" は，カンマと改行以外の文字からなる文字列にマッチし，その文字列が引数のアドレスに格納されることを表す．

3　応用問題

問題 6

これまでの問題の応用として，学生の名前と得点のデータから学生の合計点を出力するプログラムを作成せよ．この問題の入力には学籍番号がないので，学生の情報を表す構造体を

```
struct student { char name[32]; int score; };
```

とするとよい．この問題は，まず空のハッシュ表を用意して，入力ごとに「その名前がハッシュ表にあればその値を更新，なければ新たに追加する」を繰り返せば解くことができる．

作成すべきプログラムの形式は問わないが，以下の仕様を満たすものとする．標準入力から与えられる入力は 1 行以上 10000 行以下で，各行には名前 (カンマを含まない 31 文字以下の文字列) と得点 (0 以上 100 以下の整数値) が与えられ，標準出力には，各学生の得点の合計を入力と同様の形式[4] で出力する．出力する順序については，問題 3 で定義したハッシュ関数 (SIZE は 101 とする) によるハッシュ値が小さいものを先に出力し，同じハッシュ値であれば先に出現した名前ほど先に出力するものとする (ハッシュ表にない場合の登録の仕方を工夫すれば，この順序で出力することは容易である)．

入力例

```
Ichiro Suzuki,51
Yukio Mishima,100
Shohei Ohtani,17
Cristiano Ronaldo,7
Shinji Kagawa,23
Shohei Ohtani,95
Shinji Kagawa,52
Yukio Mishima,100
Cristiano Ronaldo,7
```

出力例

```
Cristiano Ronaldo,14
Yukio Mishima,200
Shinji Kagawa,75
Ichiro Suzuki,51
Shohei Ohtani,112
```

[4] 出力の場合は，得点が 100 以上になりうるが，int 型の範囲は超えないと仮定してよい．