2018 年度 情報領域演習第三
— 第 11 回 —

注意事項

- 今回から木構造データを扱う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，わからないことがあれば TA や教員に質問しよう．
- CED の開室状況を踏まえ，締切りが通常と異なるので，checker コマンドで確認しよう．
- 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/18jr3/11/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 8 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/18jr3/11/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/18jr3/11/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
```

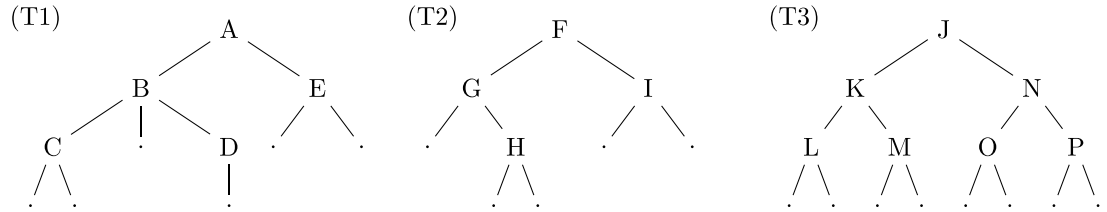
なお，一度「成功」となった問題に対し，実行テストに失敗するプログラムを送信してしまうと，結果が「失敗」になってしまうので注意すること．

1 木

前回，探索アルゴリズムとして線形探索と二分探索を扱ったが，二分探索は事前に整列されていることが前提となっている反面，線形探索よりはるかに高速に探索が終了する．ハッシュ法による探索も格納方法を工夫しておくことで，更に高速な探索を実現している．このように，データに対して前処理を行ったり格納するデータ構造を工夫したりすることで，その後の処理を効率化できることが多い．今回は，その基本的なデータ構造である**木**を扱う．まず，木についての基本用語を復習しておこう．

1.1 節点・根・葉

下の図のような構造のことを木という．



(T1)，(T2)，(T3) のそれぞれの木において，A から P までのアルファベットの部分を**節点** (node)[1] といい，特に節点 A，F，J を**根** (root)，最下部にある「.」の位置を**葉** (leaf) という．葉を節点に含めることもある．節点や葉を結ぶ線は親子関係を表しており，上にあるものが**親** (parent)，下にあるものが**子** (child) である．(T1) の木において節点 A には 2 つの子があり，節点 A の子である節点 B には 3 つの子がある．また，根には親がなく，葉には子がない．日常用語と同じく，親，親の親，親の親の親，…のことを**祖先** (ancestor)，子，子の子，子の子の子，…のことを**子孫** (descendant) という．さらに，同じ親をもつ子同士を**兄弟** (sibling) という．ある節点から別の節点まで，親子関係の線に沿ってたどる経路を**道** (path) という．たとえば，(T3) の木において，根 J から左から 2 番めの葉 (.) までの道は左，左，右と子をたどる経路である．

1.2 二分木

(T2) や (T3) のように (葉を除く) どの節点も子がちょうど 2 つであるような木を**二分木**といい[2]，特に (T3) のように根から葉までの道の長さ (道に含まれる節点の個数) がすべて等しいものを**完全二分木**という．二分木や完全二分木は多くのアルゴリズムで重要な役割を果たしており，二分木の節点 (葉を除く) に情報を持たせたデータ構造を扱うことで効率的に問題を解くことができる．

二分木のすべての節点を見る順序 (なぞり，traverse) には以下の 3 種類がある：

[1] リストにも節点があったが同じものと考えることができる．リストはすべての節点の子が 1 つしかない特殊な木である．
[2] 「どの節点も子が 2 つ以下であるような木」を二分木と呼ぶ流儀もある．

行きがけ順 (pre-order) 子より親を先に見るが，子同士なら左の子を先に見る．たとえば，(T2) の二分木なら，F，G，H，I の順，(T3) の二分木なら，J，K，L，M，N，O，P の順に見る．

通りがけ順 (in-order) 左の子は親より先に見るが，右の子よりは親を先に見る．たとえば，(T2) の二分木なら，G，H，F，I の順，(T3) の二分木なら，L，K，M，J，O，N，P の順に見る．

帰りがけ順 (post-order) 子は親より先に見るが，子同士なら左にある子を先に見る．たとえば，(T2) の二分木なら，H，G，I，F の順，(T3) の二分木なら，L，M，K，O，P，N，J の順に見る．

二分木でない場合でも行きがけ順や帰りがけ順は考えることができる．たとえば，(T1) の木において，A，B，C，D，E の順に見るのが行きがけ順，C，D，B，E，A の順に見るのが帰りがけ順である．

今回の演習では二分木に関する問題を解くプログラムをいくつか作成してもらう．C 言語において，二分木は以下の構造体 `node` を用いて定義できる：

```
struct node { datatype data; struct node *left, *right; };
```

構造体 `node` は 1 つの節点を表し，メンバ `left` には左の子の節点のアドレス，メンバ `right` には右の子の節点のアドレスを保持している．メンバ `data` には二分木の節点に情報を格納するためのもので，汎用性のためにその情報の型は `datatype` としているので，問題に応じて `typedef` を用いてあらかじめこの型を定義しておく必要がある．たとえば，`int` 型の値を二分木に格納するなら，

```
typedef int datatype;
```

という宣言を構造体 `node` の宣言の前に記述すればよい．ここで扱う二分木においては，葉を `NULL` で表し，葉には情報を格納しないものとする．

今回扱う問題では，二分木を表す入力として複数行に渡る以下の形式を用いる (この入力は (T2) の二分木を表し，橙色の文字は入力に含まない)：

```
2345 ← (T2) の二分木の節点 F にあるデータ
123  ← (T2) の二分木の節点 G にあるデータ
.    ← (T2) の二分木の節点 G の左の子の葉
456  ← (T2) の二分木の節点 H にあるデータ
.    ← (T2) の二分木の節点 H の左の子の葉
.    ← (T2) の二分木の節点 H の右の子の葉
789  ← (T2) の二分木の節点 I にあるデータ
.    ← (T2) の二分木の節点 I の左の子の葉
.    ← (T2) の二分木の節点 I の右の子の葉
```

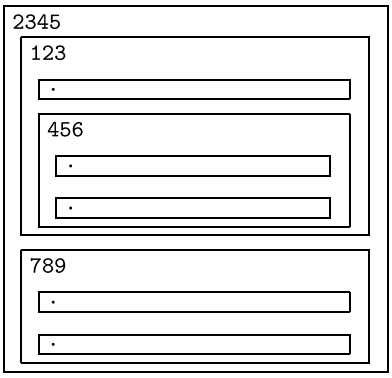
ここでは，型 `datatype` が `int` の場合を例として説明する．この入力は (T2) の二分木を葉も含めて行きがけ順に並べたものであり，. (ドット) は葉を表す．この形式では，葉を除く各節点から下の木が，

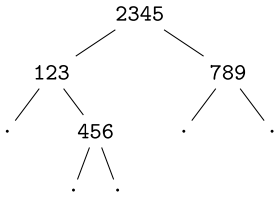
その節点の情報

左の子から下の木

右の子から下の木

という形で表されている．次に示すように，先ほどの入力に対して，葉や小さい木を表す部分を 1 つの塊と見て枠を補うと，(T2) の二分木と対応していることがよくわかるかもしれない．





このようにみると，2345 というデータを持つ節点には 2 つの子があり，左の子は 123 というデータを持つ節点，右の子は 789 というデータを持つ節点であることが一目でわかるだろう．この形式の入力から，構造体 node による二分木を生成するプログラムは，次のような再帰関数 get_tree として定義できる．この関数は,「標準入力から二分木を生成し，根の節点のアドレスを返す関数」である．

```
struct node* get_tree() {
  struct node *t;
  /* 標準入力から buf に文字列を読み込み,先頭が . なら葉とみなして NULL を返す */
  if(fgets(buf,sizeof(buf),stdin)==NULL || buf[0]=='.')
    return NULL;
  else {
      /* 先頭が . 以外なら,節点のためのメモリを確保 */
      t = (struct node*)malloc(sizeof(struct node));
      sscanf(buf,"%d␣",&t->data);  /* buf から数値として読み込み，メンバ data に保存 */
      t->left = get_tree();  /* 以降の入力で,左の子の節点から下の木を生成 */
      t->right = get_tree(); /* 左の木が生成できたら,右の子の節点から下の木を生成 */
      return t;
  }
}
```

ただし，buf は文字列のためのグローバル変数であり，十分なメモリが確保されているものとする．なお，. (ドット) の 1 行のみである場合は，葉だけからなる節点のない二分木を表すが，この関数はそのような入力にも対応している．

二分木の各節点において，左右の子の節点から下の構造もまたそれぞれ二分木とみなせるので，再帰関数で処理されることが多い．具体的には，

```
xxx f(struct node *t) {
  if(t==NULL) {
     ...
  } else {
    ...
    ... f(t->left) ...
    ... f(t->right) ...
    ...
  }
}
```

のような形である．再帰関数を使わずに，while 文を用いて，

```
while(t!=NULL) {
   ...
   if(...) {
     ...
     t = t->left;
   } else {
```

```
        ...
        t = t->right;
      }
  }
```

のように処理することもある．問題や目的に応じて使い分けるとよい．

問題 1

型 datatype は int とする．構造体 node で表現された二分木に対し，指定された形式で画面に表示する関数 print_tree を定義せよ．関数 print_tree の引数と返り値の型は以下の通りである：

```
void print_tree(struct node *t);
```

この関数は,「構造体 node のアドレス t の指す節点を根とする二分木を，指定された形式で標準出力に出力する関数」であり，その形式は

メンバ data に格納された整数 (左の子の節点から下の木 , 右の子の節点から下の木)

であり，葉は - で表す．たとえば，先ほど示した，根に 2345 が格納されていた二分木をこの関数に渡すと，

```
2345(123(-,456(-,-)),789(-,-))
```

が出力される．関数 print_treeは今後の問題を解く上でも重要な関数である．実行中の二分木の状態をこの関数を使って確認することで，効率よくプログラムをデバッグできるだろう [3]．

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，標準入力から与えられる入力は複数行に渡る文字列で int 型を格納する二分木を表すものとし，標準出力には括弧とカンマを用いた上記の形でその二分木を出力する．入力には必ず二分木に対応する文字列が与えられると仮定してよい (以下の問題でも同様)．

入力例

```
2345
123
.
456
.
.
789
.
.
```

出力例

```
2345(123(-,456(-,-)),789(-,-))
```

入力例

```
456
123
.
.
777
789
.
.
.
```

出力例

```
456(123(-,-),777(789(-,-),-))
```

[3] ただし，型 datatype が異なる場合はその都度修正が必要である．

入力例

```
123
.
456
.
789
.
.
```

出力例

```
123(-,456(-,789(-,-)))
```

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>
char buf[128]; /* 関数 get_tree で用いるグローバル変数 */

typedef int datatype;
struct node { datatype data; struct node *left, *right; };

struct node* get_tree() {
  /* ※ここは上で示した通り */
}

void print_tree(struct node *t) {
  /* ※ここを適切なプログラムで埋める */
}

int main() {
  struct node *t = get_tree();
  print_tree(t);
  printf("\n");
  return 0;
}
```

問題 2

型 `datatype` は `int` とする．構造体 `node` で表現された二分木 (根の節点のアドレス) を引数とする以下の関数をすべて定義せよ．

- `int size(struct node *t);`
  `t` の指す節点を根とする二分木に含まれるすべての節点 (葉を除く) の数を返す関数
- `int height(struct node *t);`
  `t` の指す節点を根とする二分木の高さを返す関数
- `int sum(struct node *t);`
  `t` の指す節点を根とする二分木に含まれるのすべての節点のデータ (整数) の合計を返す関数

ここで，木の**高さ**とは，根から最も遠い葉までの道の長さ (通る節点の個数) のことで，最初に示した (T1), (T2), (T3) の木の高さはいずれも 3 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，標準入力から与えられる入力は複数行に渡る文字列で `int` 型を格納する二分木を表すものとし，標準出力にはその二分木の節点の数，高さ，合計を順にそれぞれ改行とともに出力する．

入力例

```
2345
123
.
456
.
.
789
.
.
```

出力例

```
4
3
3713
```

入力例

```
456
123
.
.
777
789
.
.
.
```

出力例

```
4
3
2145
```

入力例

```
123
.
456
.
789
.
.
```

出力例

```
3
3
1368
```

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>
#include<stdlib.h>
char buf[128]; /* 関数 get_tree で用いるグローバル変数 */

typedef int datatype;
struct node { datatype data; struct node *left, *right; };

struct node* get_tree() {
  /* ※ここは問題 1 と同じ */
}

int size(struct node *t) {
  /* ※ここを適切なプログラムで埋める */
}

int height(struct node *t) {
  /* ※ここも適切なプログラムで埋める */
}

int sum(struct node *t) {
  /* ※ここも適切なプログラムで埋める */
}

```

```
24
25 int main() {
26   struct node *t = get_tree();
27   printf("%d\n",size(t));
28   printf("%d\n",height(t));
29   printf("%d\n",sum(t));
30   return 0;
31 }
```

問題 3

型 datatypeは charとする．構造体 node で表現された二分木 (根の節点のアドレス) を引数とする以下の関数をすべて定義せよ．

- `void print_preorder(struct node *t);`
  t の指す節点を根とする二分木の各節点のデータ (文字) を行きがけ順に標準出力に出力する関数
- `void print_inorder(struct node *t);`
  t の指す節点を根とする二分木の各節点のデータ (文字) を通りがけ順に標準出力に出力する関数
- `void print_postorder(struct node *t);`
  t の指す節点を根とする二分木の各節点のデータ (文字) を帰りがけ順に標準出力に出力する関数

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，標準入力から与えられる入力は複数行に渡る文字列で char 型を格納する二分木を表すものとし，標準出力にはその二分木の (葉を除く) すべての節点のデータ (文字) を行きがけ順，通りがけ順，帰りがけ順に並べたものを順にそれぞれ改行とともに出力する．

入力例

```
A
B
.
.
C
.
.
```

出力例

```
ABC ← 行きがけ順
BAC ← 通りがけ順
BCA ← 帰りがけ順
```

入力例

```
F
G
.
H
.
.
I
.
.
```

出力例

```
FGHI
GHFI
HGIF
```

入力例

```
J
K
L
.
.
M
.
.
N
O
.
.
P
.
.
```

出力例

```
JKLMNOP
LKMJONP
LMKOPNJ
```

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>
char buf[128]; /* 関数 get_tree で用いるグローバル変数 */

typedef char datatype; /* ← 格納するデータは char 型 */
struct node { datatype data; struct node *left, *right; };

struct node* get_tree() {
  /* ※ここは問題 1 とほぼ同じだが，%d は %c で置き換える */
}

void print_preorder(struct node *t) {
  /* ※ここを適切なプログラムで埋める */
}

void print_inorder(struct node *t) {
  /* ※ここも適切なプログラムで埋める */
}

void print_postorder(struct node *t) {
  /* ※ここも適切なプログラムで埋める */
}

int main() {
  struct node *t = get_tree();
  print_preorder(t);  printf("\n");
  print_inorder(t);   printf("\n");
  print_postorder(t); printf("\n");
  return 0;
}
```

1.3 二分探索木

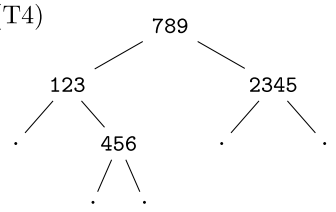
前回配列の中から目的の値を見つける探索アルゴリズムをいくつか復習したが，配列には，格納する要素数を柔軟に変更できないという制約がある．要素数を自由に変更できるデータ構造として連結リストがあるが，二分探索のように真ん中の節点の値をすぐに見つけることができないため，結局リンクを 1 つ 1 つたどることとなり，線形探索よりも効率が悪くなってしまう．

この問題を解決するのが**二分探索木**である．前回学習した配列の中から探索するアルゴリズムの二分探索とは (名前は似ているが) 異なるアルゴリズムであることに注意せよ．二分探索木とは，大小関係が定義されたデータを節点に格納した二分木のうち，葉を除くすべての節点のデータが以下の条件を満たすものである．

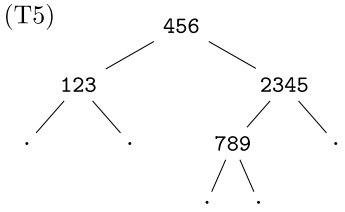
左の子とその子孫の節点のデータより大きく，
右の子とその子孫の節点のデータより小さい (か等しい)

たとえば，次の 3 つの木 (T4)，(T5)，(T6) は二分探索木である．

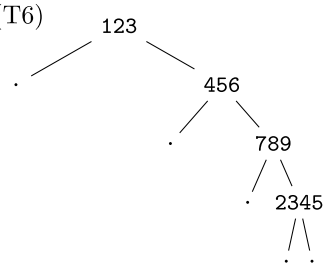
(T4)



(T5)

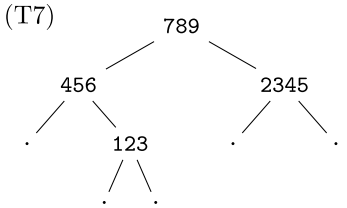


(T6)

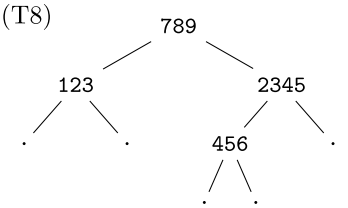


二分木 (T4) では，根のデータ 789 は，左の子とその子孫のデータ 123，456 よりも大きく，右の子のデータ 2345 よりも小さい．また，左の子のデータ 123 についても，右の子のデータ 456 より小さい．したがって，(T4) は二分探索木と言える．二分木 (T5) では，根のデータ 456 は，左の 123 よりも大きく，右の 2345 や 789 より小さい．また，右の子 2345 についても，左の子 789 より大きい．よって，(T5) も二分探索木である．二分木 (T6) では，すべての節点の左の子が葉であり，データをもつ節点が右に偏っているが，どの節点も右の子孫より小さいデータをもつので，(T6) も二分探索木である．二分木が二分探索木の条件を満たすとき，通りがけ順に節点のデータを並べると必ず昇順になる [4]．一方，以下の二分木はいずれも二分探索木ではない．

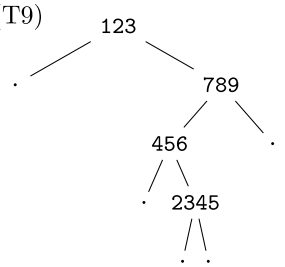
(T7)



(T8)



(T9)



二分木 (T7) では，456 をデータにもつ節点の右の子のデータが 123 であるため，二分探索木の条件を満たさない．二分木 (T8) では，根のデータが 789 であるが，右の子孫の節点のデータが 456 であるため，二分探索木の条件を満たさない．二分木 (T9) では，789 をデータにもつ節点の左の子孫の節点のデータが 2345 であるため，二分探索木の条件を満たさない．

二分木が二分探索木の条件を満たすとき，その二分木に目的のデータが格納されているかを探索することは容易である．根から始めて，その節点のデータより目的のデータの方が小さければ左の子をたどり，目的のデータの方が大きければ右の子をたどる，ということを繰り返せばよい．たとえば，(T4) の二分探索木から 456 というデータを見つけたい場合，まず，根の 789 と比較して小さいので左の子に行き，その節点の 123 と比較して大きいので右の子に行き，その節点が 456 と等しいということで目的のデータが見つかる．

見つからない場合には葉にたどり着いて終了するため，最も時間のかかる場合でも二分探索木の高さの分だけの繰り返しで必ず目的のデータの有無がわかる．二分探索木が完全二分木である場合には，$2^k - 1$ 個のデータから目的のデータを見つけるのに k 回の比較しか必要としない．すなわち，平均的には配列の二分探索と同じ $O(\log n)$ の時間計算量で探索問題を解くことができる．ただ，(T6) のように偏りがある場合には線形探索と同じ時間が必要になってしまうというのが，二分探索木の欠点である．

今回の演習では，二分探索木からデータを見つける関数，二分探索木に新たにデータを挿入する関数，二分探索木からデータを削除する関数を定義してみよう．最後には二分探索木からデータを見つける関数

[4] 逆は必ずしも成り立たない．同じ順位のデータが含まれている例を考えればわかるだろう．

を工夫したものも実装してもらう．二分木に格納するデータとして，前回扱った学生の情報を表す構造体 student を用いる：

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

メンバ id は学籍番号，メンバ name は名前，メンバ score は得点を表す．今回では，メンバ id (学籍番号) の大小関係に基づく二分探索木を扱うものとする．

問題 4

型 datatype は struct student とし，構造体 student を格納する二分探索木を構造体 node で表現する．このとき，二分探索木 (根の節点のアドレス) と学籍番号から，該当する学生の名前と成績を標準出力に出力する関数 find_info を作成せよ．関数 find_info の引数と返り値の型は以下の通りである：

```
void find_info(struct node *t, int id);
```

この関数は,「構造体 node のアドレス t の指す節点を根とする二分探索木の中から，学籍番号が id と一致する学生の名前と成績をカンマで区切った文字列を標準出力に出力する関数」である．一致する学生がいないときは，Not found. と出力する．

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 find_info の定義を適切に埋めることにより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複数行に渡る文字列で，問題 1 と同じ形式で二分木を表したものであるが，節点の情報は，学籍番号 (5 桁の整数値) と名前と得点 (0 から 100 までの整数値) をカンマで区切った文字列で表現され，入力の最後の行には探索すべき学籍番号が 5 桁の整数値が与えられる．与えられた学籍番号と一致する学生を上記のアルゴリズムで探索し，見つかった場合にはその名前と得点をカンマで区切った文字列を標準出力に出力し，見つからなかった場合には Not found. を出力する．入力には必ず二分探索木に対応する文字列と学籍番号が適切に与えられ，同じ学籍番号の学生は複数いないと仮定してよい．

入力例

```
10789,Cristiano Ronaldo,7
10123,Ichiro Suzuki,51
.
10456,David Beckham,77
.
.
12017,Osamu Dazai,50
.
.
10456
```

出力例

```
David Beckham,77
```

入力例

```
10456,David Beckham,77
10123,Ichiro Suzuki,51
.
.
12017,Osamu Dazai,50
10789,Cristiano Ronaldo,7
.
.
.
12017
```

出力例

```
Osamu Dazai,50
```

入力例

```
10456,David Beckham,77
10123,Ichiro Suzuki,51
.
.
12017,Osamu Dazai,50
10789,Cristiano Ronaldo,7
.
.
.
10777
```

出力例

```
Not found.
```

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>
#include<stdlib.h>
char buf[128]; /* 関数 get_tree で用いるグローバル変数 */

struct student { int id; char name[32]; int score; };
typedef struct student datatype; /* ← 格納するデータは構造体 student */
struct node { datatype data; struct node *left, *right; };

/* 葉を含む行きがけ順で表現された文字列を二分木に変換 */
struct node* get_tree() {
  struct node *t;
  if(fgets(buf,sizeof(buf),stdin)==NULL || buf[0]=='.')
    return NULL;
  else {
    t = (struct node*)malloc(sizeof(struct node));
    sscanf(buf,"%d,%[^,],%d",&t->data.id,t->data.name,&t->data.score);
    t->left = get_tree();
    t->right = get_tree();
    return t;
  }
}

void find_info(struct node *t, int id) {
  /* ※ここを適切なプログラムで埋める */
}

int main() {
  int id;
  struct node *t = get_tree();
  scanf("%d␣", &id);
  find_info(t,id);
  return 0;
}
```

問題 5

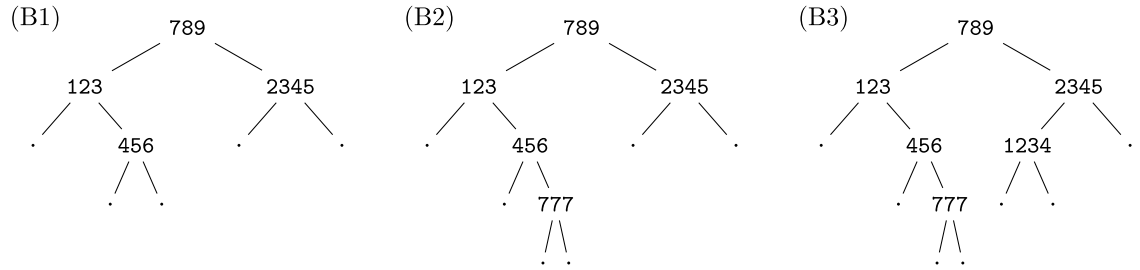
型 `datatype` は `struct student` とし，構造体 `student` を格納する二分探索木を構造体 `node` で表現する．このとき，二分探索木 (根の節点のアドレス) に構造体 `student` の値を追加する関数 `bst_insert` を作成せよ．bst は binary search tree (二分探索木) の略である．関数 `bst_insert` の引数と返り値の型は以下の通り：

```
struct node* bst_insert(struct node *t, struct student d);
```

この関数は,「構造体 node のアドレス t の指す節点を根とする二分探索木に，構造体 student の値 d をメンバ data とする節点を追加し，得られた二分探索木の根の節点のアドレスを返す関数」である．以下のアルゴリズムからもわかるように，t が NULL でない限り，返り値は t と等しくなる．t が NULL のときは，葉だけのデータのない二分探索木であるため，追加すべきデータを含む節点だけからなる二分探索木の根の節点のアドレスを返す.

二分探索木に新しいデータを追加するアルゴリズムは探索とほぼ同じである．二分探索木の根の節点のデータより，追加するデータが小さいときは左の子の節点に移り，大きいときは右の子の節点に移る．同様にして，移った先の節点のデータと比較し，追加するデータが小さいときは左に，大きいときは右に，と繰り返し，葉 (NULL) にたどり着いたら，追加するデータを含む新たな節点によってその葉を置き換えればよい．C 言語ではこの「葉を置き換える」という操作はポインタの付け替えによって行われるため，葉が入っていたメモリのアドレス (親の節点のメンバ left か right のアドレス) を憶えておく必要があることに注意しよう．連結リストの挿入の際に直前の節点を覚えておく必要があったこととよく似ている．

以下は，数値をデータとして節点に格納する二分探索木への追加の例である．二分探索木 (B1) に 777 を追加して得られる二分探索木が (B2) であり，二分探索木 (B2) に 1234 を追加して得られる二分探索木が (B3) 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 bst_insert と関数 print_bst の定義を適切に埋めることによりプログラムを完成させよ．関数 print_bst は，二分探索木を問題 4 の入力と同じ形式 (最後の行を除く) で出力する関数である．標準入力から与えられる入力は複数行に渡る文字列で，最後の行を除き，問題 4 と同じ形式で二分木を表したものであり，最後の行には追加すべき学生の学籍番号と名前と得点がカンマで区切った文字列が与えられる．標準出力には，与えられた学生の情報を上記のアルゴリズムによって追加した探索二分木を，入力と同じ形式 (葉を含む行きがけ順，最後の行を除く) で出力する．入力には必ず二分探索木に対応する文字列と追加する学生の情報が適切に与えられ，二分探索木の中には同じ学籍番号の学生は複数おらず，追加する学生は元の二分探索木には存在しないと仮定してよい.

入力例

```
10789,Cristiano Ronaldo,7
10123,Ichiro Suzuki,51
.
10456,David Beckham,77
.
.
12017,Osamu Dazai,50
.
.
10777,Douglas Adams,42
```

出力例

```
10789,Cristiano Ronaldo,7
10123,Ichiro Suzuki,51
.
10456,David Beckham,77
.
10777,Douglas Adams,42
.
.
12017,Osamu Dazai,50
.
.
```

入力例

```
10789,Cristiano Ronaldo,7
10123,Ichiro Suzuki,51
.
10456,David Beckham,77
.
10777,Douglas Adams,42
.
.
12017,Osamu Dazai,50
.
.
11234,Sakae Tsuboi,24
```

出力例

```
10789,Cristiano Ronaldo,7
10123,Ichiro Suzuki,51
.
10456,David Beckham,77
.
10777,Douglas Adams,42
.
.
12017,Osamu Dazai,50
11234,Sakae Tsuboi,24
.
.
.
```

入力例

```
.
10123,Ichiro Suzuki,51
```

出力例

```
10123,Ichiro Suzuki,51
.
.
```

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>
char buf[128]; /* 関数 get_tree で用いるグローバル変数 */

struct student { int id; char name[32]; int score; };
typedef struct student datatype; /* ← 格納するデータは構造体 student */
struct node { datatype data; struct node *left, *right; };

struct node* get_tree() {
  /* ※ここは問題 4 と同じ */
}

struct node* bst_insert(struct node *t, struct student d) {
  /* ※ここを適切なプログラムで埋める */
}

void print_bst(struct node *t) {
  /* ※ここも適切なプログラムで埋める */
}

int main() {
  struct node *t = get_tree();
  struct student d;
  scanf("%d,%[^,],%d␣", &d.id, d.name, &d.score);
  t = bst_insert(t, d);
  print_bst(t);
  return 0;
}
```

問題 6

型 `datatype` は `struct student` とし，構造体 `student` を格納する二分探索木を構造体 `node` で表現する．このとき，二分探索木 (根の節点のアドレス) から構造体 `student` の値を削除する関数 `bst_delete` を作成せ

よ．関数 `bst_delete` の引数と返り値の型は以下の通り：

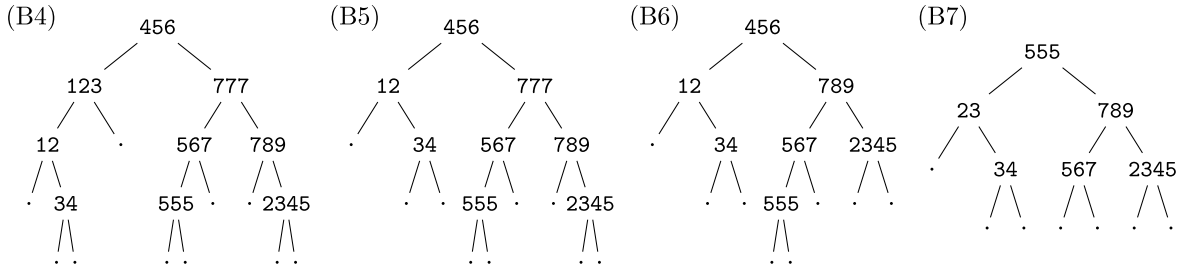
```
struct node* bst_delete(struct node *t, int id);
```

この関数は,「構造体 `node` のアドレス `t` の指す節点を根とする二分探索木から，学籍番号が `id` と一致する学生の節点を削除し，得られた二分探索木の根の節点のアドレスを返す関数」である．削除する節点のメモリは関数 `free` で解放するものとする．

二分探索木からデータを削除するアルゴリズムは以下の通りである．まず，問題 4 と同じアルゴリズムで削除すべきデータの位置を探し，その節点を n とする．このとき，次の 3 通りの場合に分けて考える．

- n の右の子が葉であるとき，n があったところに n の左の子がそのまま入る．
- n の右の子の左の子が葉であるとき，n があったところには n の右の子の節点が入り，その左の子として n の左の子が，右の子として，n の右の子の右の子が入る．
- そうでなければ n の右の子から始めて，左の子，左の子の左の子，左の子の左の子の左の子，... と葉になる直前までたどり，そのデータを n があったところに入れ，そのデータがあったところにはその右の子を入れる．

たとえば，二分探索木 (B4) から 123 を削除する場合を考えよう．問題 4 と同じ要領で，まず 123 の場所を探し，右の子が葉なので，左の子 (12) をそのまま入れて (B5) の二分探索木を得る．さらに (B5) から 777 を削除する場合，まず，777 の場所を探し，右の子の左の子が葉なので，777 のあったところに，右の子 (789) が入り，その左の子として 777 の左の子 (567) が，右の子として，777 の右の子 (789) の右の子 (2345) が入り，(B6) の二分探索木を得る．さらに (B6) から 456 を削除する場合，まず，456 の場所を探し，右の子 (789) から左の子をたどり続け，葉にたどり着く直前の 555 を 456 のあったところに入れ，555 のあったところにその右の子 (.) を入れ，(B7) の二分探索木を得る．



C 言語では，この置き換えはポインタの付け替えによって実現される．連結リストで節点を削除する場合と同様に，置き換える場所のリンク元を憶えておく必要があることに注意しよう．

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 `bst_delete` の定義を適切に埋めることにより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複数行に渡る文字列で，最後の行を除き，問題 4 と同じ形式で二分木を表したものであり，最後の行には削除すべき学生の学籍番号が与えられる．標準出力には，与えられた学生の情報を上記のアルゴリズムによって削除した探索二分木を，入力と同じ形式 (葉を含む行きがけ順) で出力する．ただし，入力には必ず二分探索木に対応する文字列と削除する学籍番号が与えられ，削除する番号は二分探索木の中にあり，同じ学籍番号の学生は複数いないと仮定してよい．

入力例

```
10456,David Beckham,77
10123,Ichiro Suzuki,51
10012,Hideki Matsui,55
.
10034,Akira Kurosawa,34
.
.
.
10777,Douglas Adams,42
10567,Yuto Nagatomo,55
10555,Tetsuro Tamba,75
.
.
.
10789,Cristiano Ronaldo,7
.
12017,Osamu Dazai,50
.
.
10123
```

出力例

```
10456,David Beckham,77
10012,Hideki Matsui,55
.
10034,Akira Kurosawa,34
.
.
10777,Douglas Adams,42
10567,Yuto Nagatomo,55
10555,Tetsuro Tamba,75
.
.
.
10789,Cristiano Ronaldo,7
.
12017,Osamu Dazai,50
.
.
```

入力例

```
10456,David Beckham,77
10012,Hideki Matsui,55
.
10034,Akira Kurosawa,34
.
.
10777,Douglas Adams,42
10567,Yuto Nagatomo,55
10555,Tetsuro Tamba,75
.
.
.
10789,Cristiano Ronaldo,7
.
12017,Osamu Dazai,50
.
.
10777
```

出力例

```
10456,David Beckham,77
10012,Hideki Matsui,55
.
10034,Akira Kurosawa,34
.
.
10789,Cristiano Ronaldo,7
10567,Yuto Nagatomo,55
10555,Tetsuro Tamba,75
.
.
.
12017,Osamu Dazai,50
.
.
```

入力例

```
10456,David Beckham,77
10012,Hideki Matsui,55
.
10034,Akira Kurosawa,34
.
.
10789,Cristiano Ronaldo,7
10567,Yuto Nagatomo,55
10555,Tetsuro Tamba,75
.
.
.
12017,Osamu Dazai,50
.
.
10456
```

出力例

```
10555,Tetsuro Tamba,75
10012,Hideki Matsui,55
.
10034,Akira Kurosawa,34
.
.
10789,Cristiano Ronaldo,7
10567,Yuto Nagatomo,55
.
.
12017,Osamu Dazai,50
.
.
```

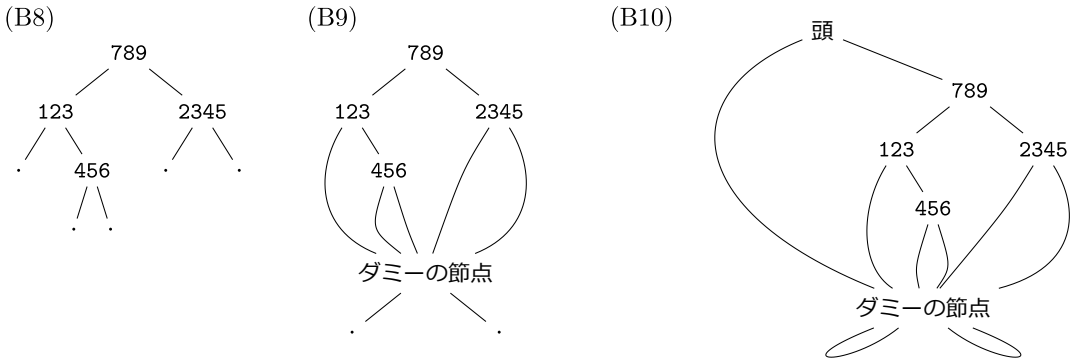
作成すべきプログラムは以下の形式とする．※印を含むコメント部分を適切に書き換えること．

```c
#include<stdio.h>
#include<stdlib.h>
char buf[128]; /* 関数 get_tree で用いるグローバル変数 */

struct student { int id; char name[32]; int score; };
typedef struct student datatype; /* ← 格納するデータは構造体 student */
struct node { datatype data; struct node *left, *right; };

struct node* get_tree() {
  /* ※ここは問題 5 と同じ */
}

void print_bst(struct node *t) {
  /* ※ここも問題 5 と同じ */
}

struct node* bst_delete(struct node *t, int id) {
  /* ※ここを適切なプログラムで埋める */
}

int main() {
  struct node *t = get_tree();
  int id;
  scanf("%d␣", &id);
  t = bst_delete(t, id);
  print_bst(t);
  return 0;
}
```

1.4 ダミーの節点を利用した二分探索木

前回，効率のよい線形探索を実装するために番兵を用いたことを思い出そう．添字の小さい方から探索する前に，探索する範囲の 1 つ後ろに目的のデータをあらかじめ入れておくという方法で，探索が必ず成功するため毎回配列の範囲を超えていないか確かめる必要がない．探索が終わった時点の添字が探索する範囲に入っていれば目的のデータが見つかったことになり，範囲を超えていればその範囲の配列には見つからなかったことになる．

これと同様のアイデアで，二分探索木においても毎回葉 (NULL) であるかどうかを確認せずに探索することを可能にする方法がある．二分探索木 (B8) を用いる代わりに，まず，ダミーの節点を用意し，(B9) のように二分探索木のすべての葉を NULL ではなくダミーの節点を指すようにしておく．このダミーの節点に目的のデータを格納しておけば，左右の子をたどりながら探索していく際に，毎回 NULL であるか確認する必要はない．ただし，このような二分探索木をあらかじめ用意したとしても，ダミーの節点にデータを格納するためにダミーの節点まで子をたどる必要がある．そこで，(B10) のように，連結リストの頭に相当する節点を用意し，その節点の左の子をダミーの節点とし，右の子として実際の二分探索木の根を置く．これにより，頭のアドレスさえ知って入れば，二分探索木のダミーの節点に目的の値を入れるという操作を簡単に行うことができる．さらにダミーの節点の左右の子として自分自身を指すことで，その節点を見ただけでダミーであるかわかるようにしておくと便利である．



問題 7

与えられたすべてのデータ (構造体 student) を格納するような頭とダミーを用いた二分探索木にデータを追加する関数 bst_insert を定義せよ．関数 bst_insert の型は以下の通りである：

```
void bst_insert(struct node *t, struct student d);
```

この関数は,「構造体 node のアドレス t の指す節点を根とする二分探索木に，構造体 student の値 d をメンバ data とする節点を追加する関数」である．葉は NULL ではなくダミーの節点を用いるため，問題 5 と同じ関数ではない点に注意せよ．また，頭付きの二分探索木を扱うことで，葉だけからなる節点のない二分木を特別扱いする必要がないため，返り値の型も void 型となっている．

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 bst_insert の定義を適切に埋めることにより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複数行に渡る文字列で，各行は学籍番号と名前と得点をカンマで区切った文字列である．標準出力には，与えられた学生の情報を上記のとおりに作成した探索二分木を，問題 5 の出力と同じ形式 (葉を含む行きがけ順) で出力する．入力には同じ学籍番号の学生は複数いないと仮定してよい．

入力例

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

出力例

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

入力例

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

出力例

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

入力例

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

出力例

```
10123,Ichiro Suzuki,51
.
10567,Cristiano Ronaldo,7
10345,Shohei Ohtani,17
.
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．

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

struct student { int id; char name[32]; int score; };
typedef struct student datatype; /* ← 格納するデータは構造体 student */
struct node { datatype data; struct node *left, *right; };

void bst_insert(struct node *t, struct student d) {
  /* ※ここを適切なプログラムで埋める */
}

/* 葉がダミーになった二分探索木を出力する関数 */
void print_bst_dummy(struct node *t, struct node *dummy) {
  if(t==dummy)
    printf(".\n");
  else {
    printf("%d,%s,%d\n",t->data.id,t->data.name,t->data.score);
    print_bst_dummy(t->left,dummy);
    print_bst_dummy(t->right,dummy);
  }
  return;
}
/* 葉がダミーになった頭付きの二分探索木を出力する関数 */
void print_bst(struct node *t) {
  print_bst_dummy(t->right,t->left);
  return;
}

int main() {
  char buf[128];
  struct node *t = (struct node*)malloc(sizeof(struct node)),
              *dummy = (struct node*)malloc(sizeof(struct node));
```

```
33   struct student st;
34   t->left = t->right = dummy->left = dummy->right = dummy;
35   while(fgets(buf,sizeof(buf),stdin)!=NULL) {
36     /* st の指すメモリに入力された学生のデータを格納 */
37     sscanf(buf,"%d,%[^,],%d",&st.id,st.name,&st.score);
38     /* 二分探索木にそのデータを追加 */
39     bst_insert(t, st);
40   }
41   print_bst(t);
42   return 0;
43 }
```

問題 8

構造体 student の情報を格納した二分探索木から目的のデータを見つけるプログラムを完成させよ．問題 4 とは異なり，関数 find_info の引数として与えられるアドレス t は，二分探索木の根ではなく，上のような頭のついた二分探索木であることに注意せよ．探索のアルゴリズムも葉であるかを判定せずに子をたどっていく方法を用いるものとする．

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 find_info の定義を適切に埋めることにより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複数行に渡る文字列で，最後の行を除き，各行は学籍番号と名前と得点をカンマで区切った文字列である．最後の行は探索すべき学籍番号を表す 5 桁の整数値と . (ドット) が与えられる．標準出力には，まず，問題 7 と同様に作成した二分探索木を出力し，その後に探索結果を出力する．探索については，与えられた学籍番号と一致する学生を上記のアルゴリズムで探索し，見つかった場合にはその名前と得点をカンマで区切った文字列を標準出力に出力し，見つからなかった場合には Not found. を出力する．入力には同じ学籍番号の学生は複数いないと仮定してよい．

入力例

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

出力例

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

入力例

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

出力例

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

入力例

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

出力例

```
10345,Shohei Ohtani,17
10012,Hideki Matsui,55
.
10123,Ichiro Suzuki,51
.
10234,Makoto Hasebe,17
.
.
10789,Cristiano Ronaldo,7
10567,Yuto Nagatomo,55
10456,David Beckham,77
.
.
.
.
Not found.
```

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>

struct student { int id; char name[32]; int score; };
typedef struct student datatype; /* ← 格納するデータは構造体 student */
struct node { datatype data; struct node *left, *right; };

void bst_insert(struct node *t, struct student d) {
  /* ※ここは問題 7 と同じ */
}

void print_bst_dummy(struct node *t, struct node *dummy) {
  /* ※ここも問題 7 と同じ */
}

void print_bst(struct node *t) {
  /* ※ここも問題 7 と同じ */
}
```

```
19
20 void find_info(struct node *t, int id) {
21   /* まず,ダミーの節点のデータに目的の学籍番号を代入 */
22   t->left->data.id = id;
23   /* ※ここを適切なプログラムで埋める */
24   /*   while 文で節点を移動するたびにダミーかどうか判定しないこと.*/
25 }
26
27 int main() {
28   char buf[128], c;
29   int id;
30   struct node *t = (struct node*)malloc(sizeof(struct node)),
31               *dummy = (struct node*)malloc(sizeof(struct node));
32   struct student st;
33   t->left = t->right = dummy->left = dummy->right = dummy;
34   while(fgets(buf,sizeof(buf),stdin)!=NULL) {
35     sscanf(buf, "%d%c", &id, &c);
36     if(c==',') {
37       sscanf(buf,"%d,%[^,],%d",&st.id,st.name,&st.score);
38       /* 二分探索木にそのデータを追加 */
39       bst_insert(t, st);
40     }
41   }
42   print_bst(t);
43   find_info(t, id);
44   return 0;
45 }
```