2018 年度 情報領域演習第三
— 第 5 回 —

注意事項

- 課題に補足や訂正がある場合には演習の Web ページに掲載される．その他の連絡事項も掲載されることがあるので，演習の Web ページは適宜確認しておくこと．
- 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．
- 締切りは 1 週間後に設定しているが，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．
- 問題の文章中の下線は，注意点や制約などを示しているので気をつけて読むこと．

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/18jr3/05/checkerN
提出開始: 11 月 xx 日 14 時 40 分
提出締切: 11 月 yy 日 14 時 39 分
ユーザ: p1610999, 出席状況: 2018-11-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/18jr3/05/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/18jr3/05/checkerN
提出開始: 11 月 xx 日 14 時 40 分
提出締切: 11 月 yy 日 14 時 39 分
```

```
ユーザ: p1610999, 出席状況: 2018-11-xx-14-47
 問題:    結果 | 提出日時            | ハッシュ値                          |
    1:    成功 | 2018-10-05-15-33   | 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 連結リスト

C 言語では同じ要素を複数まとめて保持する方法として配列を用いることができるが，配列には主に 2 つの欠点がある．

1 つめの欠点はプログラムの実行中に長さを変更できない点である．例えば，`int a[100];` は長さ 100 の `int` 型の配列として変数 `a` を宣言する文であるが，一度宣言した後，プログラムの途中で 100 では足りないとわかっても拡張することはできない．余裕を持って長い配列を用意すればよいかもしれないが，それほど必要ないのであればメモリの無駄になってしまうし，後から部分的にメモリを解放することはできない．

2 つめの欠点は挿入や削除が頻繁に起こるプログラムには適していないという点である．例えば，"新宿", "明大前", "千歳烏山", "調布", "府中", ... と並んでいる配列があったとき，"新宿" と "明大前" の間に "笹塚" を挿入するためには，"明大前"から後ろの要素を全て後ろにずらさなければならない．また，"千歳烏山" を削除するために，"調布" から後ろの要素を全て前にずらさなければならない．

これらの問題を解決するのが**連結リスト** (linked list, 単にリストということもある) である．連結リストの基本的なアイデアは，配列のように連続したメモリ空間に要素を配置せずに，要素ごとに別々にメモリを確保する，というものである．もちろん，それぞれの要素がバラバラにメモリに配置されているだけでは，並んでいる順序がわからないので，それぞれの要素に「次の要素に対応するアドレス」を情報として含める必要がある．これを実現するために，要素と次のアドレスを組にした**節点** (node) を使って連結リストを実現する．まず，節点を表す構造体 `node` を次のように用意しよう:

```
1 struct node {
2   elementtype element;
3   struct node *next;
4 };
```

構造体 `node` には 2 つのメンバがあり，`element` は要素，`next` は次の節点のアドレスを表す．`elementtype` は要素の型で，格納したい要素に応じて適切に `typedef` を行えばよい (要素が `int` 型なら `typedef int elementtype`, `char` 型なら `typedef char elementtype` とする．前回の `stack` や `queue` と同様の理由である．忘れてしまった人は前回の課題のスタックの項をもう一度読んでおくこと)．この構造体 `node` を使えば，どの節点からも次の要素を含む節点がわかるので，"新宿", "明大前", "千歳烏山", "調布", "府中", ... のように並んだデータを実現することができる (この場合の `elementtype` は `char*`型)．最後の節点には次がないので，メンバ `next` には定数 `NULL` (ナルポインタ) を入れる．定数 `NULL` は絶対に使われないアドレス (0 であることが多い) として定義されているので，`next` が `NULL` ならば,「次の節点のアドレスが `NULL`」という意味ではなく,「次の要素がない」という意味になる．

前回の資料にもあるように，構造体をそのまま関数に渡しても中身が書き換わらなかったり効率が悪かったりするので，連結リストも上のような構造体 `node` へのポインタとして表現することが多い．このため，構造体 `node` の宣言に続いて次のように定義する:

```
5 typedef struct node* list;
```

この宣言によって，list という型を struct node*型の別名として使うことができるようになる．構造体 node のメンバ next は struct node*型であることから，これも list 型として考えることができる．つまり，list 型の変数 x があるとき，x->next も list 型であり，「次の要素以降のリスト」を表すことになる．同様に x->next->next は「次の次の要素以降のリスト」を表す．

まず，「頭のないリスト」を考えて最初の問題を解いてみよう．

問題 1

先述の節点を表す構造体 node を用意して，連結リストを操作する以下の関数を定義せよ：

- list cons(elementtype e, list l);
  リスト l の先頭に e を追加したリストを返す関数．
- int length(list l);
  リスト l の長さ (要素の数) を返す関数．
- void print_int_list(list l);
  リスト l に含まれる要素を先頭から順に [と] で挟んで標準出力に出力する関数．
  要素の型は int 型であるものに限る．

この問題では要素の型は int 型とするので，typedef を用いて elementtype を適切に設定すること．標準入力として与えられるのは複数行に渡る文字列で，各行につき 1 つの整数値 (int 型で表される範囲) が含まれている．作成すべきプログラムは，空のリストから始めて，入力された整数値を関数 cons を用いて順にリストの先頭に追加していき，入力が終わったら，そのリストの長さ L を関数 length を用いて計算し，標準出力の 1 行めに length=L を表示して，2 行めにリストの内容を関数 print_int_list を用いて出力する．提出するプログラムの main 関数部分を後に示すので，この部分は変更せずに構造体や関数などの定義を追加して作成せよ．なお，関数 cons の定義では malloc を用いて構造体 node のメモリを確保するものとする．

入力例

```
1
2
3
```

出力例

```
length=3
[3][2][1]
```

入力例

```
-11
7
1
0
2
```

出力例

```
length=5
[2][0][1][7][-11]
```

作成するプログラムの main 関数は以下のものとする (変更してはならない)．

```
1  int main(){
2    int i;
3    char buf[128];
4    list l = NULL;
5    while(fgets(buf,sizeof(buf),stdin) != NULL) {
6      sscanf(buf,"%d",&i);
7      l = cons(i, l);
8    }
9    printf("length=%d\n", length(l));
10   print_int_list(l);
11   return 0;
12 }
```

このプログラムでは，入力されるたびにその要素を先頭に追加していくので，入力された順とは逆順にリストに並ぶことを確認しよう．なお，この問題においてリストの長さは関数 cons を呼び出す回数でも求めることができるが，関数 length を用いて計算するものとする．

リストの「頭」

要素を含む節点だけでなく，先頭に「頭」とよばれるダミーの節点をつけて扱う場合がある．まず，その理由を理解するために次の例を考えてみよう．

前問の main 関数では，cons によって先頭に次々と追加することで入力とは逆順に要素が並んだリストを作っていったが，入力と同じ順序で並んだリストを作るにはどうしたらよいだろうか．7 行めの代わりに，常に最後の節点を憶えておいてその next として新たな要素を追加する，とすれば，順序通りに並べることが実現できそうだ．

```
1 last->next = cons(i, NULL); /* 最後の節点の次に，要素 i が先頭で次の要素がない節点を追加する */
2 last = last->next;          /* 元の最後の節点の次の節点を,新たな最後の節点とする */
```

ここで，last は list 型の変数として最初に宣言されているものとする．ただし，これが可能なのは「リストが空でない場合」に限る．リストが空の場合は節点が 1 つもないので「最後の節点」が存在せず，ナルポインタ NULL に対してメンバ next を参照しようとしてエラーになってしまう．

そこで，ダミーの節点を先頭に追加した「頭」のあるリストを考えてみよう．空リストは，頭となる節点からなるリストで，その節点の next が NULL となっている．この場合，この頭の節点を last としておけば，上の操作によって要素が 1 つ末尾に追加されることになる.「頭」を使うと便利な理由はこれだけではないが,「空リストを特別扱いしなくてよい」というのが主な理由である．なお，頭の節点 (node) のメンバ element はリストの要素ではなく，通常は値を代入せずに使われるため，間違って参照してしまうとセグメンテーション違反となることがあるので注意せよ．

問題 2

「頭」のある連結リストを操作する次のプログラムを完成させよ．

```
1  int main(){
2    int i;
3    char buf[128];
4    list l, last;
5    /* ここで l を「頭」のある空リストとして適切に初期化する */
6    last = l;
7    while(fgets(buf,sizeof(buf),stdin) != NULL) {
8      sscanf(buf,"%d",&i);
9      last->next = cons(i,NULL);
10     last = last->next;
11   }
12   printf("length=%d\n", length(l));
13   print_int_list(l);
14   return 0;
15 }
```

作成すべきプログラムは，上記の main 関数 (5 行めを適切に埋める．1 行でなくてもよい) を含むプログラムで，構造体や length と print_int_list 以外の関数の定義は問題 1 と同じ定義である．関数 length と print_int_list は適切に定義し直すこと．入出力についても問題 1 とほぼ同じで，異なる点は作成するリストの要素の順序が入力の順序と同じになる点だけである．

入力例

```
1
2
3
```

出力例

```
length=3
[1][2][3]
```

入力例

```
-11
7
1
0
2
```

出力例

```
length=5
[-11][7][1][0][2]
```

問題 3

頭のある連結リストに対し，先頭に節点を挿入する関数

```
1 void insert(list l, elementtype e);
```

を定義せよ．この関数は先頭に挿入する関数であるが，リスト l の先頭の次に挿入したい場合は insert(l->next,e)，さらにその次に挿入したい場合は insert(l->next->next,e) のようにすれば，先頭以外の位置にも挿入することができる．

この問題では要素の型は char 型とする．作成すべきプログラムは，標準入力の 1 行めに与えられる文字列から char 型のリストを作成し，以降の入力の各行に与えられる「位置」と「アルファベット 1 文字」に対し，その位置にアルファベットを挿入するプログラムを作成せよ．「位置」は整数値で与えられ，0 番めが先頭を表すものとする．たとえば，標準入力の 1 行めが Big のとき，'B'，'i'，'g' の順に並んだリストを作成し，2 行めに 2 n と入力されていれば，2 番めに 'n' が挿入されて，'B'，'i'，'n'，'g' の順に並んだリストになり，続く 3 行めに 4 o と入力されていれば，'B'，'i'，'n'，'g'，'o' の順に並んだリストになる．また，標準出力には，標準入力から 1 行入力されるごとに，その時点でのリストの内容を先頭から順に文字列として並べたものと改行文字を出力する．「位置」がその時点でのリストの長さより大きい場合には，末尾に挿入するものとせよ．入力は 2 行以上あるものと仮定してよい．なお，作成するプログラムの形式は問わないが，1 行めの入力は必ずリスト (先述の list 型，頭の有無は問わない) に格納すること．ヒントとして，main 関数の例を示したので参考にするとよい．

入力例

```
Big
2 n
4 o
```

出力例

```
Big
Bing
Bingo
```

入力例

```
ink
0 L
42 d
4 e
```

出力例

```
ink
Link
Linkd
Linked
```

ヒント 指定された入力を処理する main 関数は以下のように定義できる．作成すべきプログラムでは，構造体や補助関数を適切に定義し，main 関数は※印のあるコメントの部分を適切なプログラム片で置き換えればよい．

```
1 int main() {
2   char c, buf[128];
3   int i;
4   list l;
5   /* ※ここでリスト l を適切に初期化 */
6   fgets(buf,sizeof(buf),stdin); /* ← 1 行めを buf に読み込み */
7   for(i=0; (c = buf[i])!='\n'; ++i) { /* ← 1 文字ずつ処理する for 文 */
8     /* ※ここで c をリスト l の末尾に挿入 */
9   }
```

```
10    /* ※ここでリスト l の中身を出力 */
11    while(fgets(buf,sizeof(buf),stdin) != NULL) { /* ← 2 行め以降を順に buf に読み込み */
12      sscanf(buf,"%d␣%c", &i, &c);
13      /* ※ここでリスト l の i 番めに c を挿入 */
14      /* ※ここでリスト l の中身を出力 */
15    }
16    return 0;
17  }
```

問題 4

頭のある連結リストに対し，先頭の節点を削除する関数

```
1  void delete(list l);
```

を定義せよ．この関数は，空リストに対しては何も削除しないものとする．この関数は先頭を削除する関数であるが，リスト l の先頭の次を削除したい場合は delete(l->next)，さらにその次を削除したい場合は delete(l->next->next) のようにすれば，先頭以外の位置のものも削除することができる．なお，削除の際にどこからもたどることができないメモリが残ってしまうこと (メモリリーク) が起こらないように，不要になった節点のメモリは free 関数によって必ず解放する必要がある．

この問題では要素の型は char型とする．作成すべきプログラムは，標準入力の 1 行めに与えられる文字列から char 型のリストを作成し，以降の入力の各行に与えられる「位置」に対し，その位置のアルファベットを削除するプログラムを作成せよ.「位置」は整数値で与えられ，0 番めが先頭を表すものとする．たとえば，標準入力の 1 行めが Bingo のとき，'B'，'i'，'n'，'g'，'o' の順に並んだリストを作成し，2 行めに 2 と入力されていれば，2 番めの'n' が削除されて，'B'，'i'，'g'，'o' の順に並んだリストになり，続く 3 行めに 3 と入力されていれば，'B'，'i'，'g'，の順に並んだリストになる．また，標準出力には，標準入力から 1 行入力されるごとに，その時点でのリストの内容を先頭から順に文字列として並べたものと改行文字を出力する.「位置」がその時点でのリストの長さ以上の場合には，何も削除しないものとする．なお，作成するプログラムの形式は問わないが，1 行めの入力は必ずリスト (先述の list 型) に格納すること．問題 3 のヒントの main 関数を参考にするとよい．

入力例

```
Bingo
2
3
```

出力例

```
Bingo
Bigo
Big
```

入力例

```
Linked
4
4
4
4
0
4
```

出力例

```
Linked
Linkd
Link
Link
Link
ink
ink
```

問題 5

標準入力から与えられる文字列を連結リストに格納し，'(' と')' で挟まれた部分を先頭に移動し，'[' と']' で挟まれた部分を末尾に移動するプログラムを作成せよ．たとえば，a[b](cd)e であれば，'a'，'['，'b'，']'，'('，'c'，'d'，')'，'e' の 9 つを要素とする連結リストが生成され，このプログラムにより，'c'，'d'，'a'，'e'，'b' の 5 つを要素とする連結リストに変形される．

問題文から明らかかもしれないが，この問題では要素の型は char型である．作成すべきプログラムは，標準入力として改行までの文字列を読み込み，各文字を char 型のリストに格納し，上で示した節点の移動を行い，そのリストの要素を順に標準出力に出力する．入力される文字列の開き括弧と閉じ括弧は正しく対応しており，同じ種類の括弧は入れ子になっていないものと仮定してよい．また，同じ種類の括弧が複数含まれる場合は，先に現れる方を先に置くものとする．なお，入力にはどちらかあるいは両方の括弧が現れない場合もあるので注意せよ．作成するプログラムの形式は問わないが，1 行めの入力は必ずリスト (先述の list型，頭の有無は問わない) に格納すること．また，節点の移動は next の繋ぎ換えで実現するものとし，入力の文字列をリストに格納した後で，節点を新たに導入したり節点の element を書き換えたりしてはならない．入力として与えられる文字列は最後に改行文字を含む文字列で，改行文字を除き 126 文字以下とする．入力の読み込みは問題 3 の main 関数の例を参考にするとよい．

入力例

```
a[b](cd)e
```

出力例

```
cdaeb
```

入力例

```
a[b(cd)ef]g(hi)[j]k
```

出力例

```
cdhiagkbefj
```

入力例

```
(U[niversity]) of Electro-Commu(nications)
```

出力例

```
Unications of Electro-Communiversity
```

問題 6

連結リストにおいて，末尾の節点の next として NULL を置く代わりに先頭の節点のアドレスを入れたものを**循環リスト**という．たとえば，"春"の次が"夏"，"夏"の次が"秋"，"秋"の次が"冬"，"冬"の次が"春"に戻るようなリスト構造を作ることができる．循環リストは，どの節点からも (同じリストに含まれる) 他の全ての節点にたどり着くことができるため，OS のプロセスの管理などにも用いられている．通常の連結リストと循環リストの違いは，末尾の節点の next が NULL であるか先頭の節点を指すかの違いだけであるため，これまでと同じ構造体 node を用いて循環リストを実現することが可能である．ただし，この問題では頭のないリストを考え，要素は必ず 1 個以上あるものと仮定してよい．

この問題では要素の型は int型とする．標準入力から与えられるのは複数行に渡る (1 行以上の) 文字列で，各行につき 1 つの整数値 (-10000 以上で int 型で表される範囲) が含まれている．作成すべきプログラムは，入力の文字列を順に格納した循環リストを作成し，格納した整数の最大値を要素とする節点のうち最も後に現れた節点から順に，再び同じ節点に戻るまで，整数値を [と] で挟んで標準出力に出力するプログラムである．プログラムの形式は問わないが，構造体 node を用いて循環リストを作成するものとし，循環リストの内容を順に出力する際には，回数を数えて終了を判定するのではなく同じ節点に戻ったかどうかで判定すること．

入力例

```
1
3
5
2
4
```

出力例

```
[5][2][4][1][3]
```

入力例

```
182
-8585
182
-17
182
-9999
```

出力例

```
[182][-9999][182][-8585][182][-17]
```

問題 7

これまでの連結リストでは，各節点に次の節点のアドレスの情報をもたせていたが，前の節点のアドレスを知ることはできない．たとえば，ある特定の節点 n を削除したい場合には，n の前の節点の next を，n の次の節点のアドレスで置き換える，という操作が必要となるが，前の節点を知らないとこの操作を行うことができない．そこで，全ての節点について次の節点も前の節点もわかるように考案されたのが**双方向リスト** (doubly-linked list) である．双方向リストにおける節点は，以下のような構造体 dlnode で表現できる:

```
1 struct dlnode {
2   elementtype element;
3   struct dlnode *prev, *next;
4 };
```

メンバ element と next は，連結リストの節点を表す構造体 node と同じものだが，さらに prev という前の節点を参照するためのメンバが追加されている．この問題では，各双方向リストには，連結リストの「頭」に相当するダミーの節点があり，ダミーの節点の next が先頭の節点で，先頭の節点の prev がダミーの節点，そして，ダミーの節点の prev が末尾の節点で，末尾の節点の next がダミーの節点となっているものとする．空リストであれば，ダミーの節点の next も prev もダミーの節点自身のアドレスになっている．prev と next がそれぞれ前の節点と次の節点を表すことから，構造体 dlnode のポインタ変数 d について，d->next->prev と d->prev->next は常に d は同じアドレスでなければならない．この性質は挿入や削除があっても常に成り立つようにする必要がある．

この問題では，要素の型を int型とし，まず構造体 dlnode を定義した上で，以下の関数を実装せよ:

- `void insert(struct dlnode *p, elementtype e);`
  p の指している節点の prev に，element として e を含む節点を挿入する関数.
- `void delete(struct dlnode *p);`
  p の指している節点のみを削除する関数 (p の前後のポインタを繋ぎ変えて，free 関数により削除した節点のメモリを解放する).
- `void print_dllist(struct dlnode *d);`
  d の指しているダミーの節点から，next を順にたどって自分自身に戻るまで element を [と] で挟んで標準出力に出力し続けたのち，prev を順にたどって自分自身に戻るまで element を{と}で挟んで標準出力に出力し続ける関数 (最後に改行).
- `void append_dllist(struct dlnode *d1, struct dlnode *d2);`
  d1，d2 の指している節点をダミーの節点とする 2 つの双方向リストを連結する関数 (d1 から next をたどっていくと，1 つめの双方向リストの要素に続いて 2 つめの双方向リストの要素が現れ，d1 に戻ってくるように定義し，2 つめのダミーの節点 d2 のメモリは free 関数によって解放する).

作成すべきプログラムは，構造体 dlnode のこれらの関数定義を含むものとし，main 関数は以下で与える (※印を含むコメント部分は各自で定義すること).

```
1 int main() {
2   char buf[128];
3   int i;
4   struct dlnode *d1, *d2;
```

```
5   /* ※ここで d1, d2 が空の双方向リストのダミーの節点を指すように初期化 (d1 と d2 は異なる) */
6   while(fgets(buf,sizeof(buf),stdin)!=NULL) {
7     sscanf(buf,"%d",&i);
8     insert(d1, i); /* ← d1 の末尾に i を要素として挿入 */
9     insert(d2, i); /* ← d2 の末尾に i を要素として挿入 */
10  }
11  print_dllist(d1); /* ← d1 の内容を出力 */
12  print_dllist(d2); /* ← d2 の内容を出力 */
13
14  /* ※ここでdelete 関数を繰り返し用いて, d1 に含まれる奇数要素を削除 */
15  /* ※ここでdelete 関数を繰り返し用いて, d2 に含まれる偶数要素を削除 */
16  print_dllist(d1); /* ← d1 の内容を出力 */
17  print_dllist(d2); /* ← d2 の内容を出力 */
18
19  append_dllist(d1,d2); /* ← d1 の末尾に d2 を連結 */
20  print_dllist(d1); /* ← d1 の内容を出力 */
21  return 0;
22 }
```

入力例

```
1
2
3
```

出力例

```
[1][2][3]{3}{2}{1}
[1][2][3]{3}{2}{1}
[2]{2}
[1][3]{3}{1}
[2][1][3]{3}{1}{2}
```

入力例

```
123
-3579
-567
91
```

出力例

```
[123][-3579][-567][91]{91}{-567}{-3579}{123}
[123][-3579][-567][91]{91}{-567}{-3579}{123}

[123][-3579][-567][91]{91}{-567}{-3579}{123}
[123][-3579][-567][91]{91}{-567}{-3579}{123}
```

入力例

```
8888
-88
8
-888
```

出力例

```
[8888][-88][8][-888]{-888}{8}{-88}{8888}
[8888][-88][8][-888]{-888}{8}{-88}{8888}
[8888][-88][8][-888]{-888}{8}{-88}{8888}

[8888][-88][8][-888]{-888}{8}{-88}{8888}
```

問題 8

標準入力として整数の列を読み込み，その絶対値が大きい順に標準出力に出力するプログラムを作成せよ．絶対値が同じ場合には正の数を先に出力するものとし，同じ数値がある場合には 1 つしか出力しないものとする．作成すべきプログラムでは，問題 1 や問題 2 で使った `list` 型による連結リストを用いて，以下のアルゴリズムにより実装すること (頭の有無は問わない)．

1. `list` 型の空リスト `l` を用意する．
2. 入力行がなくなるまで 1 行ずつ以下を繰り返す:

   入力行に書かれた整数値 `i` に対し，`l` の先頭から順に要素と比較していき，適切な位置に `i` を追加する．ただし，リスト `l` に既に `i` があれば追加しない．
   (この時点で `l` は順序通り並ぶ)

3. l の中身を先頭から順に出力する.

入力例

```
3301
-2725
-467
-3301
6779
-500
```

出力例

```
6779
3301
-3301
-2725
-500
-467
```

入力例

```
9
-10
-20
0
5
-20
9
```

出力例

```
-20
-10
9
5
0
```