プログラミング演習 第 12 回

問 A [12A]: 複数のキーに対する整列

端末 (標準入力) から、8 桁の文字列のアカウントと試験の点数 (1000 点満点の整数) を空白で区切って並べた組が 1 行に 1 つずつ与えられる。この入力をすべて読み込んでから 2 つのキーにもとづいて、1) アカウントと点数の組を**点数の降順**に、2) 同点の場合は**アカウントの辞書順**に、整列し出力するプログラムを作成せよ。入力は 10,000 件以下とする。

この問題では最初に配列に読み込んでから整列するのではなく、挿入ソートの要領で 1 行読み込むたびにしかるべき位置に挿入することで整列された配列を作ること。また、読み込んだレコードの個数を n とすると、整列対象のレコードはプログラミング通論で示されたように配列の 1 番目 (添字の値が 1) から n 番目 (添字の値が n) の要素である。これは、C の配列は 0 番目 (添字の値が 0) から始まるが、挿入ソートにおいて整列対象を収めた配列の 0 番目の要素は番兵として用いるためであり、意図的に整列対象を 1 番目から格納していることに注意せよ。

図 1 に `main` 関数、ソートを行なう関数 `insert` のプロトタイプ宣言、その他必要な定義を示す (必要なヘッダファイルのインクルード、あるいは不足しているコードは各自で補うこと)。

入出力例 (503 点が二人おり、cm は yb より前である)

入力

```
fw015001 0
yb015002 503
qm015003 305
kf015004 812
se015005 990
cm015006 503
```

出力

```
se015005 990
kf015004 812
cm015006 503
yb015002 503
qm015003 305
fw015001 0
```

問 B [12B]: 0 番目から始まる配列要素に対するヒープソート

端末 (標準入力) から与えられる高々 10,000 個の整数を配列に読み込み、それをヒープソートで昇順に整列した結果を出力するプログラムを作成せよ。整列対象の整数は、配列の大きさ (10,000) に達するか、入力の終わりに達するまで読み続けるものとする。

プログラミング通論で示されたヒープソートのコードを図 2 に、ヒープソート関数 `heapsort` の呼び出しを含む `main` 関数およびレコードの型定義を図 3 に示す (必要なヘッダファイルのインクルードは各自で補うこと)。この `main` 関数からわかるように、端末 (標準入力) から与えられた整数の個数を n とすると、整列対象の整数は、プログラミング通論で扱ったように配列の 1 番目 (添字の値が 1) から n 番目の要素ではなく、0 番目から $n-1$ 番目の要素である。

入出力例

入力

```
5564
3689
8796
4760
5360
2687
```

出力

```
2687
3689
4760
5360
5564
8796
```

問 C [12C]: リストのソート

端末 (標準入力) から、8 桁の文字列のアカウントと 10 種の試験の点数 (各 100 点満点) を空白で区切って並べた組が 1 行に 1 つずつ与えられる。入力の終わりに達するまで全ての行を読み込んでから、それらを試験の**合計点の降順**に、同点の場合は**アカウントの辞書順**に出力せよ。

図 4 に `main` 関数を含むプログラムのひな形をに示す (必要なヘッダファイルのインクルードは各自で補うこと)。この図からわかるように、1 行分のデータは構造体 `student` に保持され、全データは `student` 構造体を要素とする単方向リストに格納される。試験の合計点の降順にリストを整列する関数を `listsort` とし、これを定義することでプログラムを完成させよ。

入出力例

入力
```
rw012001 16 76 78 53 42 69 98 84 42 64
yn012002 70  5 24 21  0 71 61 70 68 78
lg012003 84 52 58 69 98 29 69 41 76 94
nx012004 13 51 47  3 56 13 73 98 73 80
ue012005 98  1 56  9 16  2 35 50 14 31
qv012006  4 76 64 80 46 15 68  1 70 59
```

出力
```
lg012003 84 52 58 69 98 29 69 41 76 94
rw012001 16 76 78 53 42 69 98 84 42 64
nx012004 13 51 47  3 56 13 73 98 73 80
qv012006  4 76 64 80 46 15 68  1 70 59
yn012002 70  5 24 21  0 71 61 70 68 78
ue012005 98  1 56  9 16  2 35 50 14 31
```

条件とヒント

- **リスト要素の交換は節点を指すポインタのつなぎ変えで行うこと。要素自体の交換で実現してはいけない。**
- リストなので配列のようにランダムアクセス性 (どの要素にも一定時間でアクセスできること) はないので実現に適した整列の手順は限られるが、たとえば選択ソートを使えばよい。
- 本問は繰り返しのための構文 (`for` や `while` など) あるいは再帰を用いて実現できるが、これについても各自で実現方針を検討すること。

問 D [12D]: ヒープの別種の作成法

問 B ではヒープになっていない木の葉 (ここでは子を持たない配列要素) から根 (ここでは親を持たない配列要素) に向かって順に要素を調べ、しかるべき位置より根に近い方にある要素を葉の方に降ろす (pushdown 操作) ことでヒープ構造を作り上げた。逆に、完成したヒープの葉に新しい要素を追加し、その要素をしかるべき位置まで根の方に向かって持ち上げていくことで、ヒープに要素を追加することもできる。1 要素からなるヒープから始めて、この操作によって要素を追加することでヒープを作成するように、問 B の `heapsort` を変更せよ。

葉に追加した要素をしかるべき位置まで持ち上げる関数を `f` とすると、図 2 における関数 `heapsort` 内の、

```
for(i=n/2;i>=1;i--) pushdown(a,i,n);
```

は、

```
for(i = 1; i <= n; i++) f(a, i);
```

と書き換えられる。

```
struct student {
  char account[9];       /* アカウント名 */
  int score;             /* 試験の点数 (1000 点満点) */
};

void insert(struct student*, int, struct student); /* この関数を定義 */

int main(void) {
#define BUFSIZE 80
  char buf[BUFSIZE];
  struct student *students, v;
  int nstudents, i;

#define LIMIT 10000      /* 最大レコード数 */
  students = (struct student*)malloc(sizeof(struct student)*(LIMIT+1));

  for (nstudents = 0; nstudents <= LIMIT; nstudents++) {
    if (fgets(buf, sizeof(buf), stdin) == NULL) break;
    buf[8] = '\0';
    strcpy(v.account, buf);
    v.score = atoi(buf+9);
    insert(students, nstudents, v);
  }
  for (i = 1; i <= nstudents; i++) {
    printf("%s %d\n", students[i].account, students[i].score);
  }

  return 0;
}
```

図 1: 問 A の main 関数、その他

```
void pushdown(recordtype a[], int first, int last){
  int r = first; int k = r+r;
  while(k<=last){
    if(k<last && a[k].key<a[k+1].key) k++;
    if(a[r].key >= a[k].key) break;
    swap(&a[r],&a[k]);
    r=k; k=r+r;
  }
}
/* a[1]〜a[n] のソート */
void heapsort(recordtype a[], int n){
  int i;
  for(i=n/2;i>=1;i--) pushdown(a,i,n);
  for(i=n; i>=2; i--) {
    swap(&a[1], &a[i]);
    pushdown(a,1,i-1);
  }
}
```

図 2: 講義で示された heapsort のコード

```
typedef struct {
  int n;
} recordtype;

void heapsort(recordtype*, int); /* この関数を定義 */

#define LIMIT 10000
recordtype a[LIMIT];

int main(void) {
#define BUFLEN 16
  char buf[BUFLEN];
  int i, n;

  for (n = 0; n < LIMIT; n++) {
    if (fgets(buf, sizeof(buf), stdin) == NULL) break;
    a[n].n = atoi(buf);
  }
  heapsort(a, n);
  for (i = 0; i < n; i++) printf("%d\n", a[i]);

  return 0;
}
```

図 3: 問 B の main 関数、その他

```
typedef struct {
  char account[9];           /* アカウント名 */
  int score[10];             /* 10 種の試験の点数 (各 100 点満点) */
} student;

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

void listsort(struct node*); /* この関数を定義 */

int main(void) {
#define BUFLEN 1024
  char buf[BUFLEN], *s;
  struct node *list, *p;
  int i, n;

  p = list = (struct node*)malloc(sizeof(struct node));
  while (fgets(buf, sizeof(buf), stdin) != NULL) {
    p = p->next = (struct node*)malloc(sizeof(struct node));
    buf[8] = '\0';
    strcpy(p->data.account, buf);
    s = buf+9;
    for (i = 0; i < 10; i++) {
      while (isspace(*s)) ++s;
      for (n = 0; isdigit(*s); ++s) n = n*10 + (*s-'0');
      p->data.score[i] = n;
    }
  }
  p->next = NULL;

  listsort(list);
  for (p = list->next; p; p = p->next) {
    printf("%s", p->data.account);
    for (i = 0; i < 10; i++) printf("%3d", p->data.score[i]);
    printf("\n");
  }

  return 0;
}
```

図 4: 問 C の main 関数、その他