問題文は 1 ページにある。問題で参照するコードは 2〜5 ページにある。

問題 1. コード 1 は、二次元での極座標から直交座標への変換と直交座標から極座標への変換

$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases} \qquad \begin{cases} r = \sqrt{x^2 + y^2} \\ \theta = \arctan\dfrac{y}{x} \end{cases}$$

を、それぞれ、C の関数 `ptoo()` と `otop()` として実装したものである。

(1) 空欄を埋めよ。

ヒント C の標準ライブラリ関数 `hypot(`x`,` y`)` と `atan2(`y`,` x`)` は、それぞれ、$\sqrt{x^2 + y^2}$ と $\arctan(y/x)$ を計算する。`atan2()` の引数の順序に注意。

問題 2. コード 2, 3, 4, 5 は、long 型のデータの集まりに対してその要素の総和を C の関数として実装したものである。それぞれのコードでデータの集まりを実装するデータ構造は、キャプションに書かれたとおりである。

(1) 空欄を埋めよ。

問題 3. コード 6 は、バブルソートを C の関数 `snowflake_bubblesort()` として実装したものである。コード 7 は、ヒープソートを C の関数 `snowflake_heapsort()` として実装したものである。いずれも、第一引数は構造体 `struct snowflake` の配列の先頭を指すポインタ、第二引数はその配列の大きさである。二つの引数で定められた配列を昇順にソートする。

以下を前提とする。

- 構造体 `struct snowflake` はファイル `snowflake.h` 内で定義されている。
- 関数 `snowflake_swap()` と `snowflake_less()` は、形式が

```
void    snowflake_swap(struct snowflake *, struct snowflake *);
int     snowflake_less(const struct snowflake *, const struct snowflake *);
```

  であり、外部で定義され、そのプロトタイプ宣言がファイル `snowflake.h` 内でなされている。
  - 関数 `snowflake_swap()` は、第一引数の指す先と第二引数の指す先の値を交換する。
  - 関数 `snowflake_less()` は、第一引数の指す先と第二引数の指す先で値を比較し、前者が小さければ 0 でない整数を返し、さもなくば 0 を返す。

(1) 空欄を埋めよ。
(2) 関数 `snowflake_bubblesort()` も `snowflake_heapsort()` も第二引数は `size_t` 型であるが、これを `unsigned int` 型にするとどのような不都合があり得るか説明せよ。
(3) 最悪時間計算量の観点からは、クイックソートよりもヒープソートが優れている。にもかかわらず、クイックソートがヒープソートよりもよく使われる理由を論ぜよ。

おまけ問題 1. 2007 年に学校教育法が改正されて、日本の大学から助教授がいなくなり准教授がいるようになった。助教授と准教授の法律上の違いを説明せよ。

コード 1 極座標から直交座標への変換（二次元）

```
#include <math.h>

struct polar { /* 極座標 */
        double  r;      /* r 座標 */
        double  theta;  /* θ 座標 */
};
struct ortho { /* 直交座標 */
        double  x;      /* x 座標 */
        double  y;      /* y 座標 */
};

void
ptoo(const struct polar *p, struct ortho *o)
{
        (あ) = (い) * cos((う));
        (え) = (お) * sin((か));
}

void
otop(const struct ortho *o, struct polar *p)
{
        (き) = hypot((く), (け));
        (こ) = atan2((さ), (し));
}
```

コード 2 配列の要素の総和（添字を小さいほうから大きいほうへ動かす）

```
#include <stdlib.h>

long
array_sum(const long *array, size_t size)
{
        long    s = 0;
        size_t  i;
        for (i = (す); i < (せ); (そ)) {
                s += (た);
        }
        return s;
}
```

コード 3 配列の要素の総和（ポインタを小さいほうから大きいほうへ動かす）

```
#include <stdlib.h>

void
array_sum(const long *array, size_t size)
{
        long    s = 0;
        const long      *p;
        for (p = (ち); p < (つ); (て)) {
                s += (と);
        }
        return s;
}
```

コード 4 単方向線形リストの要素の総和（ダミーセルなし。終端は NULL ポインタ）

```
#include <stdlib.h>

struct cell {
        struct cell     *next; /* 次のセルを指すポインタ */
        long            value; /* セルに格納された値 */
};

long
slist_sum(const struct cell *head)
{
        long    s = 0;
        const struct cell       *p;
        for (p = (な); p != NULL; (に)) {
                s += (ぬ);
        }
        return s;
}
```

コード 5 二分木の節のラベルの総和（再帰的に計算）

```
#include <stdlib.h>

struct node {
        struct node     *left;  /* 左子節を指すポインタ */
        struct node     *right; /* 右子節を指すポインタ */
        long            value;  /* 節にラベルづけられた値 */
};

long
bintree_sum(const struct node *root)
{
        long    s = 0;
        if (root != NULL) {
                s = (ね);
                s += bintree_sum((の));
                s += bintree_sum((は));
        }
        return s;
}
```

コード 6 バブルソートの実装

```
#include <stdlib.h>
#include "snowflake.h"

void
snowflake_bubblesort(struct snowflake *a, size_t n)
{
        size_t  i, j;
        for (i = n - 1; i > 0; i --) {
                for (j = 0; j < i; j ++) {
                        if (snowflake_less((ひ), (ふ))) {
                                snowflake_swap((へ), (ほ));
                        }
                }
        }
}
```

コード 7 ヒープソートの実装

```
#include <stdlib.h>
#include "snowflake.h"

static inline void
sift(struct snowflake *a, size_t m, size_t n)
{
        size_t  i, j;
        for (i = m; (j = i * 2 + 1) < n; i = j) {
                if (j + 1 < n && snowflake_less( (ま) , (み) )) {
                        j ++;
                }
                if (snowflake_less( (む) , (め) )) {
                        snowflake_swap( (も) , (や) );
                } else
                        break;
        }
}

void
snowflake_heapsort(struct snowflake *a, size_t n)
{
        size_t  i;
        if (n <= 1)
                return;
        for (i = n / 2; i > 0;) {
                i --;
                sift(a, i, n);
        }
        for (i = n - 1; i > 0; i --) {
                snowflake_swap( (ゆ) , (よ) );
                sift(a, 0, i);
        }
}
```