問題文は 5 ページある。

問題 1. 三角形 ABC の内心を I とし、A, B, C, I の座標を

$$A = (x_A, y_A), \qquad B = (x_B, y_B), \qquad C = (x_C, y_C), \qquad I = (x_I, y_I)$$

とおくと、

$$(x_I, y_I) = \left(\frac{ax_A + bx_B + cx_C}{a+b+c}, \frac{ay_A + by_B + cy_C}{a+b+c} \right)$$

である。ただし、a, b, c は辺の長さ、すなわち、$a = |BC|$, $b = |CA|$, $c = |AB|$ とする。

三角形の各頂点の座標から内心の座標を求める計算を C の関数 incenter として実装せよ。関数は計算のみを行い、入出力などそれ以外の動作はいっさい行ってはならない。

(1) 図 1 のインターフェースで実装せよ。

(2) 図 2 のインターフェースで実装せよ。

```
/* 平面上の点 */
struct point {
        double  x;      /* x 座標 */
        double  y;      /* y 座標 */
};
/* 平面上の三角形 */
struct triangle {
        struct point    A;    /* 頂点 A */
        struct point    B;    /* 頂点 B */
        struct point    C;    /* 頂点 C */
};
/* 内心の計算 */
struct point    incenter(struct triangle tri);
    /* tri に三角形のデータが格納されている。 */
    /* 返値が内心のデータである。 */
```

図 1　内心を計算する関数のインターフェース (1)

```
/* 平面上の点 */
struct point {
        double  coordinates[2]; /* 順に、x 座標、y 座標 */
};
/* 平面上の三角形 */
struct triangle {
        struct point    vertices[3];    /* 順に、頂点 A、頂点 B、頂点 C */
};
/* 内心の計算 */
void    incenter(const struct triangle *tri, struct point *center);
    /* tri の指す先に三角形のデータが格納されている。 */
    /* center の指す先に内心のデータを格納する。 */
```

図 2　内心を計算する関数のインターフェース (2)

問題 2. コード 1, 2, 3, 4 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。

コード 1 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 = (あ); i < (い); (う)) {
                s += (え);
        }
        return s;
}
```

コード 2 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 = (お); p < (か); (き)) {
                s += (く);
        }
        return s;
}
```

コード 3 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 = (け); p != NULL; (こ)) {
                s += (さ);
        }
        return s;
}
```

コード 4 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 = (し);
                s += bintree_sum((す));
                s += bintree_sum((せ));
        }
        return s;
}
```

問題 3. コード 5 は、バブルソートを C の関数 ryugu_bubblesort() として実装したものである。コード 6 は、ヒープソートを C の関数 ryugu_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 ryugu 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 ryugu はファイル ryugu.h 内で定義されている。
- 関数 ryugu_swap() と ryugu_less() は、形式が

  ```
  void    ryugu_swap(struct ryugu *, struct ryugu *);
  int     ryugu_less(const struct ryugu *, const struct ryugu *);
  ```

  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 ryugu.h 内でなされている。
  - 関数 ryugu_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 ryugu_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) コード 5 のバブルソートは添字の大きな要素から確定していくが、これを、添字の小さな要素から確定するように書き換えよ。
(3) 関数 ryugu_bubblesort() も関数 ryugu_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。
(4) 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。

コード 5 バブルソートの実装（添字の大きな要素から確定）

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

void
ryugu_bubblesort(struct ryugu *a, size_t n)
{
        size_t  i, j;
        for (i = n - 1; i > 0; i --) {
                for (j = 0; j < i; j ++) {
                        if (ryugu_less( (そ) , (た) )) {
                                ryugu_swap( (ち) , (つ) );
                        }
                }
        }
}
```

コード 6 ヒープソートの実装

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

static inline void
sift(struct ryugu *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 && ryugu_less( (て) , (と) )) {
                        j ++;
                }
                if (ryugu_less( (な) , (に) )) {
                        ryugu_swap( (ぬ) , (ね) );
                } else
                        break;
        }
}

void
ryugu_heapsort(struct ryugu *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 --) {
                ryugu_swap( (の) , (は) );
                sift(a, 0, i);
        }
}
```