問題 1. 三角形 ABC の類似重心を K とし、A, B, C, K の座標を

$$A = (x_A, y_A), \qquad B = (x_B, y_B), \qquad C = (x_C, y_C), \qquad K = (x_K, y_K)$$

とおくと、

$$(x_K, y_K) = \left(\frac{a^2 x_A + b^2 x_B + c^2 x_C}{a^2 + b^2 + c^2}, \frac{a^2 y_A + b^2 y_B + c^2 y_C}{a^2 + b^2 + c^2} \right)$$

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類似重心の座標を求める計算を C の関数 `symmedian_point` 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 */
};
/* 類似重心の計算 */
struct point    symmedian_point(struct triangle tri);
    /* tri に三角形のデータが格納されている。 */
    /* 返値が類似重心のデータである。 */
```

図 1　類似重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 */
};
/* 類似重心の計算 */
void    symmedian_point(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 の指す先に類似重心のデータを格納する。 */
```

図 2　類似重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 は、配列の先頭を指すポインタと大きさを受け取って要素の最大の値を返す関数 `max_value` を定義する C のコードである。配列内のデータの並び順について何も仮定されない。空配列に対しては `long` で表現される最大の整数値を返す。

1. 空欄を埋めよ。
2. 配列の大きさを n として、この関数の時間計算量を評価せよ。

コード 1 ソートされていない配列の要素の最大値

```
#include <stdlib.h>
#include <limits.h>

long
max_value(const long *array, size_t size)
{
        long    m = LONG_MIN; /* 仮の最大値 */
        const long      *p;
        for (p = (あ); p < (い); p ++) {
                if ((う) < (え)) {
                        m = (お);
                }
        }
        return m;
}
```

問題 3. 単方向リストのセルを表す構造体 `struct cell` がファイル `slist.h` 内でコード 2 の通り定義されている。

コード 3 は、単方向線形リスト（ダミーセルなし）の先頭を指すポインタを受け取ってリストの要素中で最大の値を返す関数 `max_value` を定義する C のコードである。データの並び順について何も仮定されない。空リストに対しては `long` で表現される最小の整数値を返す。

1. 空欄を埋めよ。
2. リストの長さ（セルの個数）を n として、この関数の時間計算量を評価せよ。

コード 2 単方向線形リストに関する型定義

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

コード 3 順序なし単方向リストの要素の最大値

```
#include <stdlib.h>
#include <limits.h>
#include "slist.h"

long
max_value(const struct cell *first)
{
        long    m = LONG_MIN; /* 仮の最大値 */
        const struct cell       *p;
        for (p = [ (か) ]; p != NULL; p = [ (き) ]) {
                if ( [ (く) ] < [ (け) ] ) {
                        m = [ (こ) ];
                }
        }
        return m;
}
```

問題 4. 二分木の節点を表す構造体 `struct node` がファイル `bintree.h` 内でコード 4 の通り定義されている。

コード 5 は、二分木の根を指すポインタを受け取って節にラベルづけられた最大の値を返す関数 `max_value` を定義する C のコードである。データの並び順について何も仮定されない。空木に対しては `long` で表現される最小の整数値を返す。

1. 空欄を埋めよ。
2. 二分木の大きさ（節の個数）を n として、この関数の時間計算量を評価せよ。

コード 4 二分木に関する型定義

```
struct node {
        struct node     *left;  /* 右側の子節を指すポインタ */
        struct node     *right; /* 右側の子節を指すポインタ */
        long            value;  /* この節にラベルづけられた値 */
};
```

コード 5 順序づけられない二分木のラベルの最大値

```
#include <stdlib.h>
#include <limits.h>
#include "bintree.h"

static long
max_value_sub(const struct node *p)
{
        long    m = p->value; /* 仮の最大値 */
        if (p->left != NULL) {
                long    m_left = max_value_sub( (さ) );
                if ( (し) < (す) ) {
                        m = (せ) ;
                }
        }
        if (p->right != NULL) {
                long    m_right = max_value_sub( (そ) );
                if ( (た) < (ち) ) {
                        m = (つ) ;
                }
        }
        return m;
}

long
max_value(const struct node *root)
{
        return root != NULL ? max_value_sub(root) : LONG_MIN;
}
```

ヒント. C の構文　式$_1$? 式$_2$: 式$_3$　は、まず　式$_1$ を計算し、その値が真ならば 式$_2$ を計算してその結果を全体の結果とし、偽ならば 式$_3$ を計算してその結果を全体の結果とする式である。

問題 5. コード 6 は、配列に対するヒープソートを C の関数 `pine_sort()` として実装したものである。第一引数は構造体 `struct pine` 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 pine` はファイル `pine.h` 内で定義されている。
- 関数 `pine_swap()` と `pine_less()` は、形式が

  ```
  void    pine_swap(struct pine *, struct pine *);
  int     pine_less(const struct pine *, const struct pine *);
  ```

  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 `pine.h` 内でなされている。
  - 関数 `pine_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 `pine_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) 実用的な三つのソートアルゴリズムであるクイックソート・ヒープソート・マージソートをどう使い分ければ良いか、時間計算量と空間計算量（計算時間とメモリ消費量）の観点から解説せよ。

コード 6 ヒープソートの実装

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

static inline void
sift(struct pine *a, size_t m, size_t n)
{
        size_t  i, j;
        for (i = m; (j = i * 2 + 1) < (て); i = j) {
                if (j + 1 < (と) && pine_less(&a[j], &a[j + 1])) {
                        j ++;
                }
                if (pine_less(&a[i], &a[j])) {
                        pine_swap(&a[(な)], &a[(に)]);
                } else
                        break;
        }
}

void
pine_sort(struct pine *a, size_t n)
{
        size_t  i;
        if (n <= 1)
                return;
        for (i = n / 2; i > 0;) {
                i --;
                sift(a, (ぬ), (ね));
        }
        for (i = n - 1; i > 0; i --) {
                pine_swap(&a[(の)], &a[(は)]);
                sift(a, (ひ), (ふ));
        }
}
```