**問題 1.** (1) 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値が偶数である要素の個数を返す関数 `count_even` を定義する C のコードである。空欄を埋めよ。

(2) 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偶数の値を持つ要素の個数を返す関数 `count_even` を定義する C のコードである。空欄を埋めよ。

**コード 1** 配列の要素のうち偶数を数える

```
#include <stdlib.h>

size_t
count_even(const long *array, size_t size)
{
        size_t  counter = 0;
        const long      *p;
        for (p = [ (あ) ]; p < [ (い) ]; p ++) {
                if ( [ (う) ] % 2 == 0) {
                        [ (え) ];
                }
        }
        return counter;
}
```

**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** 単方向リストの要素のうち偶数を数える

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

size_t
count_even(const struct cell *first)
{
        size_t  counter = 0;
        const struct cell       *p;
        for (p = [ (お) ]; p != NULL; p = [ (か) ]) {
                if ( [ (き) ] % 2 == 0) {
                        [ (く) ];
                }
        }
        return counter;
}
```

**問題 2.** コード 4 は、配列に対するヒープソートを C の関数 `suzu_sort()` として実装したものである。第一引数は構造体 `struct suzu` 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 suzu` はファイル `suzu.h` 内で定義されている。
- 関数 `suzu_swap()` と `suzu_less()` は、形式が

```
void    suzu_swap(struct suzu *, struct suzu *);
int     suzu_less(const struct suzu *, const struct suzu *);
```

  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 `suzu.h` 内でなされている。
  - 関数 `suzu_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 `suzu_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。

**問題 3.**

$$\begin{cases} a_0 = 0, \\ a_1 = 1, \\ a_n = a_{n-2} + a_{n-1} \qquad n \geq 2 \text{ のとき} \end{cases}$$

で定義される数列を**フィボナッチ数列**と呼ぶ。

フィボナッチ数列の第 $n$ 項を 3600 で割った余りを計算する C の関数を、有栖川有栖がコード 5 のように書いた。江神二郎がそれを見て欠陥を指摘し、コード 6 のように修正した。火村英生は、間違ってはいないがまだ少し改良できるとして、さらに、コード 7 のように修正した。火村英生は、このプログラムでは改良の効果はほとんどないが、整数の表現に `unsigned` 型ではなく多倍長整数を使ってもっと大きなフィボナッチ数を計算するプログラムならば、改良の効果が見えると解説した。

(1) 有栖川有栖版の欠陥を説明せよ。
(2) 火村英生版が江神二郎版よりも優れている点を説明せよ。

**問題 4.** (1) クイックソートとヒープソートとマージソートはどのように使い分ければ良いか解説せよ。
(2) 単なる順序つき二分木ではうまくいかず平衡木のどれかが必要になるのはどのような状況で、平衡木がどのように解決するかを解説せよ。

**コード 4** ヒープソートの実装

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

static inline void
sift(struct suzu *a, size_t m, size_t n)
{
        size_t  i, j;
        for (i = m; (j = i * 2 + 1) < [ (け) ]; i = j) {
                if (j + 1 < [ (こ) ] && suzu_less(&a[j], &a[j + 1])) {
                        j ++;
                }
                if (suzu_less(&a[i], &a[j])) {
                        suzu_swap(&a[ (さ) ], &a[ (し) ]);
                } else
                        break;
        }
}

void
suzu_sort(struct suzu *a, size_t n)
{
        size_t  i;
        if (n <= 1)
                return;
        for (i = n / 2; i > 0;) {
                i --;
                sift(a, [ (す) ], [ (せ) ]);
        }
        for (i = n - 1; i > 0; i --) {
                suzu_swap(&a[ (そ) ], &a[ (た) ]);
                sift(a, [ (ち) ], [ (つ) ]);
        }
}
```

**コード 5** フィボナッチ数列の第 $n$ 項を 3600 で割った余りの計算（有栖川有栖版）

```
unsigned
fibonacci(unsigned n)
{
        unsigned        a0 = 0;
        unsigned        a1 = 1;
        unsigned        i;
        for (i = 0; i < n; i ++) {
                unsigned        a2 = a0 + a1;
                a0 = a1;
                a1 = a2;
        }
        return a0 % 3600;
}
```

**コード 6** フィボナッチ数列の第 $n$ 項を 3600 で割った余りの計算（江神二郎版）

```
unsigned
fibonacci(unsigned n)
{
        unsigned        a0 = 0;
        unsigned        a1 = 1;
        unsigned        i;
        for (i = 0; i < n; i ++) {
                unsigned        a2 = (a0 + a1) % 3600;
                a0 = a1;
                a1 = a2;
        }
        return a0;
}
```

**コード 7** フィボナッチ数列の第 $n$ 項を 3600 で割った余りの計算（火村英生版）

```
unsigned
fibonacci(unsigned n)
{
        if (n == 0)
                return 0;
        unsigned        a0 = 0;
        unsigned        a1 = 1;
        unsigned        i;
        for (i = 1; i < n; i ++) {
                unsigned        a2 = (a0 + a1) % 3600;
                a0 = a1;
                a1 = a2;
        }
        return a1;
}
```