```
// Q1
// To facilitate reading a large integer, please write a C program to read in
// an integer and then print it out with a comma inserted every third digits.
// Your program should contain LOOPs such that it can be more concise and
// easier to read. Example program executions are
//
// $ ./a.out
// Input an integer N, -214000000 <= N <= 214000000: 123456789
// 123,456,789
// $ ./a.out
// Input an integer N, -214000000 <= N <= 214000000: -98,765
// -98,765
//
// The first example above, user inputs 123456789.
// And the second example with input -98765.

#include <stdio.h>

int main(void)
{
    long int number; // input number
    int sign = 0, i, array[3], first, s;
        // sign: check whether number is negative
        // i: counter for loops
        // array[3]: store number separated into three parts
        // first: find the greatest non-zero index
        // s: temporary number during calculating

    printf("Input an integer N, -214000000 <= N <= 214000000: ");
    scanf("%ld", &number); // get number
    if (number < 0) { // check the sign of number
        sign = 1;
        number *= -1;
    }
    for (i = 0; i < 3; i++) { // separate number into three parts
        s = number % 1000;
        array[i] = s;
        number = number / 1000;
```

```
    }
    for (i = 0; i < 3; i++) { // get the greatest non-zero index
        if (array[i] != 0) {
            first = i;
        }
    }
    if (sign == 1) { // if number is negative, add a minus sign
        array[first] *= -1;
    }
    printf("%d", array[first]); // print the first part
    for (i = first - 1; i >=0; i--) { // print the remaining parts
        printf(",%.3d", array[i]);
    }
    printf("\n");

    return 0;
}

// Q2
// Given an integer array A[N], with N distinct elements, you program performs
//
// 1. Find A[j], such that A[j] is the smallest element in array A,
// 2. Find A[k], such that A[k] is the largest element in array A,
// 3. Reverse all elements in between A[j] and A[k], including A[j] and A[k].
//    And then print out the resulting array A.
//
// For example, N = 10, and A[N] = {6, 11, 2, 12, 17, 4, 28, 25, 16, 14}
// Your program execute to get
// $ ./a.out
// Original: 6 11 2 12 17 4 28 25 16 14
// min: A[2] 2
// max: A[6] 28
// After processing: 6 11 28 4 17 12 2 25 16 14

#include <stdio.h>
#define N 10

int main(void)
```

```
{
    int A[N] = {6, 11, 2, 12, 17, 4, 28, 25, 16, 14};
    int min = 0, max = 0, i, temp;
        // min: index of the smallest element
        // max: index of the largest element
        // i: counter for loops
        // temp: temporary number for swaping

    printf("Original:");
    for (i = 0; i < N; i++) { // print the original array
        printf(" %d", A[i]);
    }
    printf("\n");
    for (i = 1; i < N; i++) { // find max and min
        if (A[i] > A[max]) {
            max = i;
        }
        if (A[i] < A[min]) {
            min = i;
        }
    }
    printf("min: A[%d] %d\n", min, A[min]); // print min
    printf("max: A[%d] %d\n", max, A[max]); // print max
    while (min < max) { // reverse element between min and max
        temp = A[min];
        A[min] = A[max];
        A[max] = temp;
        ++min;
        --max;
    }
    printf("After processing:");
    for (i = 0; i < N; i++) { // print the after processing array
        printf(" %d", A[i]);
    }
    printf("\n");

    return 0;
}
```

```
// Q3
// Please write a C program to find all solutions for the following
// Diophantine equation
//
//  a^3 + b^3 = c^2
//
// for 1 <= a <= b <= c <= 500.
// Your program should execute as efficiently as possible.
// Example program output is
//
// $ ./a.out
// Set #1: 1 2 3
// Set #2: a2 b2 c2
// ...
//   Number of solutions found: n
//

#include <stdio.h>
#include <math.h>
#define N 500

int main(void)
{
    int a, b, c, n = 0, end = 0;
        // a, b, c for the equation
        // n for index
        // end for checking whether c > N
    double c0;
        // c0 for checking whether fraction part is 0

    // find a, b, c
    for (b = 1; b <= N && end == 0; b++) {
        // b range from 1 to N
        // loop will terminate if c > N
        for (a = 1; a <= b; a++) { // a range from 1 to b
            c = c0 = sqrt(a * a * a + b * b * b);
                // store double value into c0 and integer value into c
```

```
            if (c0 > N) { // check whether c is out of range
                end = 1;
            }
            if (c == c0 && end == 0) { // check the fraction part of c0
                ++n;
                printf("Set #%d: %d %d %d\n", n, a, b, c);
            }
        }
    }
    // print total number of sets
    printf("  Number of solutions found: %d\n", n);

    return 0;
}

// Q4
// In a poker game, each player gets 5 cards and the compare the ranking
// of the cards he/she gets. One of the ranking is "full-house", in which,
// two of the cards have the same point and the other three have the same
// point. For example, if the five cards are {2, 2, 6, 6, 6}, then it is
// a full-house. Please write a C program to perform 1,000,000 experiements
// to find the probability of getting a hand of full-house.
// Each experiment should draw five cards randomly using rand() function,
// and determine if it is a full-house. Program execution example is
// shown below.
//
// $ ./a.out
// The probability of getting a full-house is x.xx%
//

#include <stdio.h>
#include <stdlib.h>
#define N 1000000

int main(void)
{
    int card[5], k, i, exp = 0, valid = 0, n1 = 0, n2 = 0, tc = 0;
        // card[5]: get 5 cards in the beginning
```

```
    // k: random type of number ranging from 1 to 13
    // i: counter for loops
    // exp: experiment
    // valid: successful experiment
    // n1, n2: the number of card
    // tc: the number of type
double p; // probability

while (exp < N) { // test N times
    for (i = 0; i < 5; i++) { // get card[5]
        k = rand() % 13 + 1;
        card[i] = k;
    }
    for (i = 0; i < 5; i++) { // get the number of type
        if (card[i] != card[0]) {
            ++tc;
        }
    }
    if (tc != 2) { // type number != 2 isn't valid
        ++exp;
    } else {
        for (i = 0; i < 5; i++) { // calculate n1 and n2
            if (card[i] == card[0]) {
                ++n1;
            } else {
                ++n2;
            }
        }
        // check whether the combination is valid
        if ((n1 == 3 && n2 == 2) || (n1 == 2 && n2 == 3)) {
            ++exp;
            ++valid;
        } else {
            ++exp;
        }
    }
}
p = (double)valid / N * 100; // get p
```

```
    // print p
    printf("The probability of getting a full-house is %.2lf%%\n", p);

    return 0;
}
```