Zadanie 1 (10 pkt. na pracowni, później 5 pkt.). Twoim zadaniem będzie napisanie własnych implementacji następujących funkcji:

- zamieniającej każdy znak ciągu znaków na literę podaną w argumencie (bez zmiany długości ciągu),
- kopiującej ciąg znaków,
- sprawdzającej długość ciągu znaków,
- zmieniającej wszystkie wielkie litery napisu na małe, a małe na wielkie (możesz wykorzystać (is|to) (lower|upper) z ctype.h),
- porównującej dwa ciągi znaków.

Funkcje powinny przekazywać dane przy użyciu wskaźników. Poniżej znajdują się sygnatury funkcji, odpowiednio:

- void set(char* sequence, char character);
- void copy(const char* from, char* to);
- int length(const char* sequence);
- void swapcase(char* sequence);
- int compare(const char* seq1, const char* seq2);

W funkcji main przetestuj działanie zaimplementowanych funkcji.

Uwaga: o słowie kluczowym (kwalifikatorze typu) const jeszcze będzie mowa na wykładzie, ale w uproszczeniu oznacza on tutaj, że obszarów pamięci, do których prowadzi dany wskaźnik, funkcja nie może modyfikować. Dla dobrze zaimplementowanych funkcji obecność tego kwalifikatora powinna nic nie psuć (tj. nie powodować pojawienia się błędów kompilacji).

Zadanie 2 (10 pkt.). Sortowanie przez scalanie tablic długości 2^*m* działa w oparciu o następującą zasadę: w tablicy długości 2^*k* najpierw osobno sortujemy rekurencyjnie pierwsze pół i drugie pół tablicy (każde długości 2^(*k*-1)), a następnie scalamy je, przepisując we właściwej kolejności do drugiej tablicy. Tablice długości 1 są posortowane.

Napisz funkcję, która będzie sortować tablice liczb całkowitych długości potęgi dwójki (choć implementacja dla dowolnych liczb nie powinna być kłopotliwa – dla tablic nieparzystej długości wystarczy podzielić je na podproblemy o długościach różniących się o 1). Twoje rozwiązanie powinno wykorzystywać, poza oryginalną tablicą (którą może modyfikować) jeszcze tylko jedną tablicę tej samej długości; funkcja rekurencyjna powinna przyjmować jako argumenty m.in. sortowaną tablicę, a dokładniej wskaźnik na obecnie sortowany fragment, jego długość, i wskaźnik na odpowiadający region w drugiej tablicy.

Poza "właściwą" funkcją rekurencyjną uzasadnione może być napisanie też krótkiej funkcji, która będzie przygotowywać obliczenia (np. przydzielać pomocniczy bufor, ustawiać jakieś początkowe wartości) i wywoływać "właściwą" rekurencję. Taka funkcja powinna mieć sygnaturę void sortuj(int dlugosc, int* dane) i nie mieć innych efektów ubocznych, np. nie powinna drukować otrzymanej zawartości tablicy.

W main przetestuj działanie swojej funkcji – na kilku różnorodnych przykładach, albo umożliwiając ich wprowadzenie przez standardowe wejście lub argumenty wywołania.

Zadanie 3 (10 pkt.).

Dana jest dziwna, ale zarazem prosta funkcja, opisana wzorem rekurencyjnym:

- fun(0, 0) = 1,
- fun(x, y) = fun(x+y-1, 0), dla x ≤ 2, gdy nie zachodzi x = y = 0,
- fun(x, y) = fun(x-3, y+3) + fun(x-1, y), dla parzystych y oraz x > 2,
- fun(x, y) = fun(x+1, y-1) + fun(x, y-1), w przeciwnym przypadku.

Z racji, że funkcja ta może przyjmować bardzo duże wartości, będzie nas interesować **tylko ich ostatnie 9 cyfr** bez zer wiodących (czyli po prostu wypisz wyniki modulo 10^9).

Zadanie

Napisz program, który wczyta podane na wejściu zapytania postaci dwóch liczb x i y, i dla każdego z zapytań policzy i wypisze wartość fun(x, y) modulo 10^9.

Wejście

W pierwszym wierszu podana jest jedna liczba całkowita *q*, oznaczająca liczbę zapytań. W kolejnych *q* wierszach podane są zapytania. Każde z nich składa się z dwóch liczb całkowitych *x_i* oraz *y_i*.

Wyjście

Dla każdego zapytania wypisz w kolejnym, *i*-tym wierszu jedną liczbę całkowitą, oznaczającą wartość fun(x_i, y_i) modulo 10^9.

Ograniczenia

We wszystkich testach zachodzi 1 ≤ *x_i*, *y_i* ≤ 1000 i 0 ≤ *q* ≤ 10^5.

- W pierwszych 6 testach zachodzi *x_i*, *y_i* ≤ 8.
- W pierwszych 3 testach zachodzi q ≤ 10.

Przykłady

Przykład A

Wejście

```
3
5 4
5 1
1 3
```

Wyjście

```
444
36
2
```

Przykład B

Wejście

```
10
```

0 2
0 3
4 3
7 8
5 4
6 6
1 2
0 5
5 4
7 3

Wyjście

1
1
56
7882200
444
49212
1
4
444
4512

Przykład C

Wejście

5
42 100
64 23
16 63
11 23
96 62

Wyjście

539799740
749657020
571841052
636113652
381705732

Porady

Najprostsze rozwiązanie, czyli zapisanie w języku C rekurencyjnej definicji funkcji, dostanie tylko kilka punktów. Znacznie więcej można będzie uzyskać, dodając spamiętywanie wyników obliczeń w tablicy, żeby nie liczyć wiele razy tej samej wartości. Ponieważ funkcja jest nieco bardziej skomplikowana, niż przykłady znane np. z WDI, to może nie być łatwo zaimplementować ją tak, żeby używana była tylko jednowymiarowa tablica (ostatni wiersz / kolumna / przekątna całości) – ale to nie powinno być problemem, zresztą w każdym teście mamy odpowiadać na *q* pytań, potencjalnie więcej niż jedno. Jeśli nawet ze spamiętywaniem nie masz kompletu punktów, to problemem może być za mały rozmiar wykorzystywanej tablicy, a w drugiej kolejności – przepełnienie stosu (któremu możesz zaradzić wyliczając – przed właściwym przetworzeniem wejścia – stan całej tablicy sekwencją takich wywołań funkcji rekurencyjnej, że każde z nich zmieści się na stosie).

Uwagi

Przedstawienie wejścia w osobnych wierszach ma znaczenie głównie wizualne; pisząc program nie trzeba przejmować się tym, czy pomiędzy kolejnymi liczbami jest spacja, czy złamanie wiersza – specyfikator konwersji %d w napisie formatującym scanf i tak wczyta wszystkie potrzebne białe znaki poprzedzające kolejną liczbę.