Zadanie 1 (10 pkt. na pracowni, później 5 pkt.). Napisz program, który będzie drukował na standardowym wyjściu szachownicę (złożoną ze znaczków # i spacji) o zadanych wymiarach.

Program powinien wczytywać ze standardowego wejścia pięć liczb całkowitych. Dwie pierwsze mają być dodatnie i oznaczać liczbę pól (odpowiednio w poziomie i w pionie), z których składa się szachownica. Dwie kolejne, również dodatnie, to liczba znaków (również odpowiednio w poziomie i w pionie), z których składa się pojedyncze pole. Ostatnia liczba, 0 lub 1, mówi, czy lewe górne pole ma być odpowiednio puste (składać się ze spacji) czy pełne (składać się z #). Efektem wywołania echo 5 4 3 2 1 | ./a.out powinno być

```
###   ###   ###
###   ###   ###
   ###   ###
   ###   ###
###   ###   ###
###   ###   ###
   ###   ###
   ###   ###
```

W pierwszym odruchu może wydawać się naturalne napisać tu trzy czy cztery zagnieżdżone pętle, ale to rzadko kiedy (żeby nie powiedzieć "nigdy") jest dobry pomysł. Równie naturalnie jest ograniczyć się do dwóch pętli (z licznikami, których wartości mówią, w którym wierszu i kolumnie właśnie zamierzamy wypisać kolejny znak) i za pomocą instrukcji warunkowej decydować, który to znak ma być. Za rozwiązanie bez tego ograniczenia, tj. **z trzema lub więcej zagnieżdżonymi pętlami można dostać najwyżej 6 pkt. na pracowni (później 3 pkt.)**.

Uwagi i wskazówki: Rozwiązując to zadanie na pracowni możesz nie przejmować się poprawnością wczytania liczb przez funkcję scanf (tj. sprawdzać wartości przez nią zwracanej), ale wciąż należy sprawdzić, czy liczba jest zgodna z powyższym opisem, tj. dodatnia lub, w przypadku ostatniej, równa 0 lub 1.

Poza już znanymi działaniami i operatorami (m.in. dzielenie całkowitoliczbowe, reszta z dzielenia, równość) być może przydadzą się operatory logiczne takie jak && (koniunkcja, czyli "i"), || (alternatywa, czyli "lub") oraz ! (negacja, czyli "nie", umieszczany podobnie jak na logice przed warunkiem, któremu chcemy zaprzeczyć). Pisząc złożone wyrażenia z tymi operatorami, warto używać nawiasów, by było jednoznaczne, które podwyrażenia są nimi powiązane.

Zadanie 2 (10 pkt.). Odwróceniem liczby naturalnej dodatniej jest liczba, której zapis dziesiętny jest "czytanym od tyłu" zapisem dziesiętnym liczby oryginalnej. Zakładamy, że zapis dziesiętny nie zawiera wiodących zer, co oznacza, że nie każda liczba ma swoje odwrócenie. Interesują nas liczby, które zsumowane ze swoim odwróceniem dają liczbę, której zapis dziesiętny składa się wyłącznie z cyfr nieparzystych, np. każdy ze składników sum 25+52=77 czy 609+906=1515.

Napisz program, który policzy takie liczby nie przekraczające wartości zadanej w kodzie programu (żeby było łatwiej mierzyć czas wykonania poleceniem time ./a.out). Dla wartości 1000 wynikiem powinno być 120. Przetestuj swój program dla wartości 10^9 – powinien działać najwyżej mniej więcej minutę. Typ int na popularnych architekturach mieści wartości rzędu 2×10^9, więc nie powinno być potrzeby używać dłuższych typów (o których jeszcze nie było mowy na wykładzie).

Zadanie 3 (10 pkt.).

Dana jest liczba q ($1 \le q \le 100\,000$) oraz q zapytań. Każde zapytanie składa się z dwóch liczb: a i b ($1 \le a \le b \le 1\,000\,000$).

Dla każdego zapytania należy w osobnej linii wypisać liczbę liczb pierwszych z przedziału $[a, b]$.

Wejście

W pierwszym wierszu wejścia znajduje się liczba q. W każdym z kolejnych q wierszy znajduje się para liczb: a oraz b.

Wyjście

Na wyjściu powinno znaleźć się q wierszy zawierających pojedyncze liczby; liczba w i-tym wierszu jest odpowiedzią na i-te zapytanie.

Przykłady

Przykład A

Wejście

```
1
2 8
```

Wyjście

```
4
```

Przykład B

Wejście

```
2
1 1
90 100
```

Wyjście

```
0
1
```

Przykład C

Wejście

```
3
100 102
15 17
1 1000
```

Wyjście

```
1
```

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. Program nie musi też kontrolować poprawności wejścia, bo będzie uruchamiany tylko na poprawnych danych.