Zadanie 1 (15 pkt).

Zaimplementuj w osobnym module (nagłówek fgetline.h, implementacja fgetline.c) funkcję fgetline o działaniu takim jak fgets, ale w bezpieczny sposób obsługującą wiersze nieograniczonej długości. Dokładniej:

- Jej pierwszym argumentem ma być strumień typu FILE *, z którego będzie czytana treść. Na razie nie przejmuj się tym, co to za typ (będzie o tym mowa na najbliższym wykładzie) – do testowania funkcji podawaj jako argument stdin (tak jak w dotychczasowych użyciach fgets), a do czytania znak po znaku używaj fgetc, która różni się od getchar tylko tym, że przyjmuje jako (jedyny) argument właśnie taki strumień. (Jeśli masz inny pomysł na implementację, który wykorzystywałby coś innego niż odpowiednik getchar, znajdź potrzebną standardową funkcję w dokumentacji bądź skonsultuj się z osobą prowadzącą pracownię.)
- Czytane bajty powinny być wpisywane do dynamicznie przydzielonego na stercie bufora, którego rozmiar należy zmieniać (realloc) w razie potrzeby – zwiększać (na przykład za każdym razem o czynnik 2), póki czytane są nowe znaki, oraz ew. zmniejszyć w momencie, kiedy już kończymy czytać, a "tymczasowo" mamy zarezerwowany zbyt duży bufor.
- Zwracana wartość (typu char *) to wskaźnik na ten bufor.
- Czytanie należy zakończyć po napotkaniu znaku złamania wiersza ('\n'), który wtedy powinien być umieszczony w buforze, lub końca strumienia (EOF). W obu przypadkach w buforze należy umieścić bajt zerowy (a bufor musi mieć pozwalający na to rozmiar). Wyjątkiem jest sytuacja, kiedy nic nie przeczytamy przed EOF; wtedy nic nie piszemy w buforze (tylko co z nim robimy?) i zwracamy NULL.
- Nie chcemy dopuścić do utraty danych, więc jeśli coś już przeczytaliśmy, a nie powiedzie się realokacja bufora, to i tak zwracamy wskaźnik na niego, z tym, że:
  - musi on kończyć się bajtem zerowym, więc najpierw go tam wpisujemy; jeśli nie mamy na niego miejsca (zwłaszcza że właśnie nie powiodła się realokacja), to ostatnio wpisany znak... "zwracamy" do strumienia używając ungetc;
  - żeby było wiadomo, ile mamy zarezerwowanej pamięci (a może być to inna liczba niż strlen(zwroconybufor)+1), wpisujemy tę liczbę pod adres przekazany jako **drugi argument** typu long * (chyba że podano NULL, co ma oznaczać, że funkcji wywołującej naszą "nie interesuje" ta informacja).
- W innych problematycznych sytuacjach (np. niepowodzenie już pierwszej alokacji bufora) zwracaj NULL i/lub zapytaj osobę prowadzącą pracownię, co jest sensowną decyzją.
- Nie przejmuj się innymi opisanymi w dokumentacjami zachowaniami fgets (ustawianie indykatora błędu/EOF), zresztą można uznać, że dobrze użyte fgetc załatwi to, jak trzeba.

Powyższa część zadania może będzie przydatna w późniejszych tygodniach, więc napisz ją porządnie i zgodnie ze specyfikacją (być może jej wykorzystanie w późniejszych zadaniach będzie testowane z konsolidacją z innymi poprawnymi implementacjami). Za rozwiązanie tylko tej części dostaniesz **do 5 pkt.**

Napisz program, który wykorzysta powyższą funkcję, by wczytać wiersz po wierszu całą treść ze standardowego wejścia (tj. aż do EOF) i wypisze ją w sposób zgodny z wybranymi przez użytkownika flagami wywołania. Flagi mają być obsługiwane zarówno w postaci długiej, jak i krótkiej:

- --reverse, -r skutkuje wypisaniem wierszy w odwróconej kolejności;
- --sorted, -s skutkuje uporządkowaniem wierszy od najkrótszego do najdłuższego;
- --contains PATTERN, -cPATTERN skutkuje wypisaniem wierszy zawierających tylko podsłowo PATTERN – nie przejmuj się efektywnością wyszukiwania podsłowa w tekście i oczywiście nie traktuj go jako wyrażenia regularnego itp.;
- --head N, -hN i --tail N, -tN skutkują wypisaniem tylko odpowiednio pierwszych i ostatnich N wierszy.

Krótkie opcje można sklejać (choć nie bez ograniczeń, por. koniec podsłowa za -c), np. poprawne byłoby wywołanie -sh15ckot -r. Wywołania z niepoprawnymi flagami (np. --head z czymś co nie jest liczbą), flagami wykluczającymi się (jednocześnie -h i -t) albo nieistniejącymi flagami (np. -x) powinny kończyć się od razu z niezerowym kodem wyjścia (return). (Jeśli chcesz wypisywać komunikat o błędzie, nie rób tego na standardowe wyjście, tylko użyj fprintf(stderr, ...).) **Uzupełnienie 24.12**: wielokrotne wystąpienie (dowolnej postaci) którejkolwiek z flag należy traktować jako błąd użytkownika. Co za tym idzie, kolejność występowania flag jest nieistotna.

Warto napisać pętlę, która przejdzie przez wszystkie argumenty wywołania i odczyta z nich powyższe flagi. Z niektórymi kombinacjami flag może nie być potrzebne wczytywanie całego wejścia do pamięci, i wystarczy przepisywanie go na wyjście "na bieżąco" – postaraj się to zrobić właśnie tak. Z flagą -s postaraj się uniknąć wielokrotnego liczenia długości napisów – spamiętaj je. Do sortowania warto użyć qsort – żeby połączyć to z poprzednią uwagą, może być konieczne umieszczenie w tablicy jakichś struct-ów parujących napisy z ich spamiętanymi długościami.

Jeśli chcesz wydzielić część kodu do osobnego modułu, możesz to zrobić (możesz przesłać do pięciu plików), ale w fgetline.{c,h} powinna znajdować się tylko funkcja fgetline (i ew. jakieś pomocnicze rzeczy).

Zadanie 2 (15 pkt).

Paweł, wielki miłośnik lasów i leśnictwa, postanowił posadzić swój własny las, w którym będzie hodował graby. Jednakże nie będzie to zwykły las.

W pierwszym roku swojej hodowli, Paweł kupił małe, kwadratowe pole podzielone na 4 kwadraty jednostkowe. Następnie, w kwadracie o współrzędnych (0, 0) posadził grab odmiany 0 (gdyż Pawłowi łatwiej posługiwać się numerami niż skomplikowanymi nazwami ich odmian), w kwadracie (1, 1) grab odmiany 1, w (1, 0) taki odmiany 2, a w polu (0, 1) grab odmiany 3.

W każdym kolejnym roku Paweł postanawiał, że powiększy swoje pole aż czterokrotnie, dokupując 3 dodatkowe pola wielkości tego, które aktualnie posiada -- jedno po prawej stronie od aktualnie posiadanego, oraz dwa pola pod nimi, dzięki czemu jego pole wciąż było kwadratowe. Następnie Paweł rozsadzał graby o kolejnych, rosnących numerach odmian używając tego samego schematu, a mianowicie na początku rozsadził nowe odmiany grabów w prawym-dolnym kwadracie, następnie w lewym-dolnym kwadracie, a na końcu w prawym-górnym. Rozsadzanie każdego kwadratu wyglądało oczywiście tak samo jak tego pierwotnego. Dla ułatwienia, poniżej można zobaczyć jak wygląda las Pawła po upływie 2 lat.

```
0  3  12 15
2  1  14 13
8  11 4  7
10 9  6  5
```

Las Pawła jest już rozsadzany w ten sposób przez *n* lat. Z racji, że jest on już bardzo duży, potrzebuje on teraz systemu do zarządzania odmianami grabów, a mianowicie sprawdzania jaki grab rośnie na danym polu oraz sprawdzania, gdzie rośnie grab o danej odmianie. Czy jesteś w stanie mu pomóc?

Wejście

W pierwszym wierszu wejścia znajdują się dwie liczby *n* oraz *q*. Pierwsza z nich oznacza ile lat Paweł hoduje już swój las, a druga oznacza liczbę zapytań.

W kolejnych *q* wierszach znajdują się opisy zapytań w następującym formacie:

- x *w k* – wypisz numer odmiany grabu z pola o współrzędnych (*w*, *k*),
- o *d* – podaj współrzędne pola w którym rośnie grab odmiany *d*.

Wyjście

Na wyjściu, dla każdego zapytania wypisz jedną linię, zawierającą odpowiedź.

Ograniczenia

- $1 \le n \le 30$,
- $1 \le q \le 100'000$,
- $0 \le w, k \le 2^n$,
- $0 \le d \le 2^{(2*n)}$.

Dodatkowo:

- w pierwszych 6 testach zachodzi $n \le 10$,
- w pierwszych 12 testach wszystkie zapytania są typu x.

Limit pamięci to 16 MB.

Przykłady

Przykład A

Wejście

```
2 5
x 1 1
x 1 0
x 0 0
x 3 2
x 2 3
```

Wyjście

```
1
2
0
6
7
```

Przykład B

Wejście

```
4 8
x 13 10
x 0 11
x 2 1
x 12 14
x 8 1
x 9 9
x 10 15
x 7 4
```

Wyjście

```
110
207
11
92
131
65
119
26
```

Przykład C

Wejście

```
3 10
o 23
o 14
o 35
x 7 1
x 7 4
o 30
o 60
```

x 2 3
x 1 6
x 7 2

Wyjście

6 7
1 2
4 1
41
26
5 6
0 6
7
62
38