**Зад. 50 2022-SE-01** Съвременните компютри могат да влизат в различни режими за енергоспестяване (suspend) и излизат от този режим (wake-up) при настъпването на определени събития. Linux kernel предоставя специален виртуален файл `/proc/acpi/wakeup`, чрез който може да се проверява или променя настройката за "събуждане" при настъпване на различни събития, в общия случай - при активност на някое устройство. Тъй като този файл е интерфейс към ядрото, "четенето" от файла и "писането" във файла изглеждат различно.

За улеснение на задачата ще ползваме опростено описание на този файл.

Под *whitespace* разбираме произволна комбинация от табове и интервали. При четене от файла изходът представлява четири колони, разделени с whitespace, в полетата не може да има whitespace; първият ред е header на съответната колона. Примерно съдържание на файла:

```
Device    S-state      Status       Sysfs node
GLAN      S4           *enabled     pci:0000:00:1f.6
XHC       S3           *enabled     pci:0000:00:14.0
XDCI      S4           *disabled
LID       S4           *enabled     platform:PNP0C0D:00
HDAS      S4           *disabled    pci:0000:00:1f.3
RP01      S4           *enabled     pci:0000:00:1c.0
```

където:

- първата колона дава уникално име на устройстрво, което може да събуди машината, като името е ограничено до четири знака главни латински букви и цифри;
- третата колона описва дали това устройство може да събуди машината. Възможните стойности са `enabled`/`disabled`, предхождани от звездичка;
- втората и четвъртата колона ги игнорираме в рамките на задачата.

При записване на име на устройство във файла `/proc/acpi/wakeup`, неговото състояние се променя от `disabled` на `enabled` или обратно. Например, ако файлът изглежда както примера по-горе, при запис на `XHC` в него, третият ред ще се промени на:

```
XHC       S3           *disabled    pci:0000:00:14.0
```

При запис на `HDAS`, шестият ред ще се промени на:

```
HDAS      S4           *enabled     pci:0000:00:1f.3
```

Дефиниран е формат на конфигурационен файл, който описва желан комплект от настройки на wakeup събития. Примерен файл:

```
# comment bar
GLAN    disabled
LID     enabled    # comment foo
```

където:

- знакът диез (#) е знак за коментар до края на реда;
- редовете *би трябвало* да са комбинация от име на устройство и желаното състояние на настройката за събуждане при събития от това устройство, разделени с whitespace.

Напишете скрипт, който при подаден първи параметър име на конфигурационен файл в горния формат осигурява исканите настройки за събуждане. Ако във файла има ред за устройство, което не съществува, скриптът да извежда предупреждение. Обърнете внимание на обработката за грешки и съобщенията към потребителя – искаме скриптът да бъде удобен и валиден инструмент.

**Зад. 51 2022-SE-02** Дефинирана е система за бекъпване на сървъри, която държи направените архиви в главна директория (която ще наричаме *fubar*), в която има четири под-директории за различни *класове* бекъпи:

- `0` – съдържа годишни бекъпи
- `1` – съдържа месечни бекъпи
- `2` – съдържа седмични бекъпи
- `3` – съдържа дневни бекъпи

Всяка директория съдържа архивни файлове с имена във формат `hostname-area-YYYYMMDD.tar.xz`, където:

- `hostname` е името на някаква машина, която е бекъпвана;
- `area` е типът бекъп за съответната машина;
- `YYYYMMDD` е датата, на която е направен бекъпа;
- никое от горните полета не може да съдържа тире или точка;
- някои от файловете могат да са symlink-ове.

Примерни имена на файлове:

```
astero-etc-20220323.tar.xz   stratios-etc-20220428.tar.xz   nestor-db-20220404.tar.xz
gila-srv-20220405.tar.xz     catalyst-var-20220406.tar.xz   drake-home-20220404.tar.xz
dominix-var-20220404.tar.xz
```

Комбинацията от `hostname` и `area` дефинира уникален *обект* за бекъпване. Всички архивни файлове са *пълноценни* бекъпи на даден *обект* и са достатъчни за неговото възстановяване при нужда (*заб. извън обхвата на задачата*).

Ако даден файл е symlink, то той може да е валиден или счупен. Symlink-овете са създадени за удобство и не ги считаме за *пълноценни* бекъпи.

Политиката ни за бекъп казва, че за да имаме *валиден* бекъп на даден *обект*, за него трябва да имаме минимум 1 годишен, 2 месечни, 3 седмични и 4 дневни *пълноценни* бекъпа.

Важност:

- *обектите* са равни по важност помежду си;
- важността на *класовете* бекъпи е във възходящ ред по горния списък, т.е. при равни други условия дневните бекъпи са по-ценни от седмичните и т.н.;
- в рамките на един *клас* бекъпи по-новите бекъпи са по-важни от по-старите.

Напишете shell скрипт, който приема два два задължителни позиционни аргумента – име на директория и число в интервала $[1, 99]$. Примерно извикване: `./foo.sh ./bar/ 30` където:

- директорията представлява главна директория (*fubar*) на описаната система за бекпъване;
- числото дефинира колко процента е максималното допустимо използвано място на файловата система, в която се намира подадената директория.

За удобство приемаме, че директорията *fubar* и всички обекти в нея се намират в една и съща файлова система.

*Упътване*: Командата `df` извикана с аргумент име на файл/директория връща информация за файловата система, в която той се намира. Пример:

```
$ df ./README.md
Filesystem        1K-blocks     Used Available Use% Mounted on
/dev/mapper/o7-hm 100663296 61735732  37288764  63% /home
```

Скриптът трябва да изтрива *минимален* брой *пълноценни* архивни файлове така, че:

- всеки *обект* да има *валиден* бекъп;
- обръща внимание на описаните по-горе важности;
- процентите използвано място върху файловата система да е *не повече* от подаденият параметър (а ако това е невъзможно, скриптът да освободи колкото може повече място, без да нарушава *валидностите* на бекъпите на *обектите*);
- не е допустимо след работата на скрипта да останат счупени symlink–ове.

**Зад. 71 2022-SE-01** Напишете програма на C, която приема два позиционни аргумента – имена на двоични файлове. Примерно извикване: `./main data.bin comparator.bin`

Файлът `data.bin` се състои от две секции – 8 байтов хедър и данни. Структурата на хедъра е:

- `uint32_t, magic` – магическа стойност `0x21796F4A`, която дефинира, че файлът следва тази спецификация;
- `uint32_t, count` – описва броя на елементите в секцията с данни.

Секцията за данни се състои от елементи – `uint64_t` числа.

Файлът `comparator.bin` се състои от две секции – 16 байтов хедър и данни. Структурата на хедъра е:

- `uint32_t, magic1` – магическа стойност `0xAFBC7A37`;
- `uint16_t, magic2` – магическа стойност `0x1C27`;
- комбинацията от горните две magic числа дефинира, че файлът следва тази спецификация;
- `uint16_t, reserved` – не се използва;
- `uint64_t, count` – описва броя на елементите в секциата с данни.

Секцията за данни се състои от елементи – наредени 6-торки:

- `uint16_t, type` – възможни стойности: `0` или `1`;
- 3 бр. `uint16_t, reserved` – възможни стойности за всяко: `0`;
- `uint32_t, offset1`;
- `uint32_t, offset2`.

Двете числа `offset` дефинират отместване (спрямо $\mathbb{N}_0$) в брой елементи за `data.bin`; `type` дефинира операция за сравнение:

- `0`: "по-малко";
- `1`: "по-голямо".

Елементите в `comparator.bin` дефинират правила от вида:

- "елементът на `offset1`" трябва да е "по-малък" от "елементът на `offset2`";
- "елементът на `offset1`" трябва да е "по-голям" от "елементът на `offset2`".

Програмата трябва да интепретира по ред дефинираните правила в `comparator.bin` и ако правилото не е изпълнено, да разменя in-place елементите на съответните отмествания. Елементи, които са равни, няма нужда да се разменят.

**Зад. 82 2022-IN-01** Ваши колеги - асистенти по ОС имат нужда от демострационна програма на C, която да служи като пример за конкурентност и синхронизация на процеси. Напишете такава програма, приемаща два задължителни позиционни параметъра – едноцифрени числа. Примерно извикване: `./main N D`

Общ алгоритъм на програмата:

- началният (родителски) процес създава процес-наследник
- $N$ на брой пъти се изпълнява:
  - родителският процес извежда на stdout низа “`DING `”
  - процесът-наследник извежда на stdout низа “`DONG`”
  - родителският процес изчаква $D$ секунди

Гарантирайте, че:

- процесът-наследник винаги извежда “`DONG`” след като родителския процес е извел “`DING `”
- родителският процес винаги започва изчакването след като процеса-наследник е извел “`DONG`”

*Забележка*: За изчакването погледнете sleep(3).