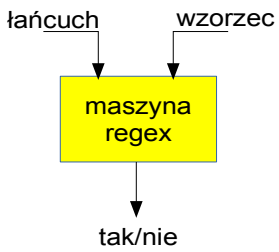
Temat zajęć	Wyszukiwanie informacji
Zakres materiału	Wyrażenia regularne. Polecenia: `grep, find`

Wyrażenia regularne

Wyrażeniem regularnym (ang. *regular expression*, często skracane do *regex* lub *regexp*) nazywa się ciąg znaków (napis), tworzący wzorzec wyszukiwania (często również wzorzec zamiany), nakładany na inny ciąg w celu stwierdzenia istnienia lub braku zgodności między nimi. Pomysł wyrażeń regularnych pochodzi od amerykańskiego matematyka Stephena Kleena, który w roku 1950 pracował nad przetwarzaniem i rozpoznawaniem języka naturalnego.

Każdy znak wyrażenia regularnego jest albo tzw. *metaznakiem* (któremu przypisuje się specjalne znaczenie) albo *literałem* (tzn. znakiem, który oznacza sam siebie).

Semantykę wyrażeń regularnych definiuje się najczęściej przy pomocy tzw. *maszyny wyrażeń regularnych*, będącej deterministycznym automatem skończonym. Maszyna próbuje odszukać w przekazanym jej łańcuchu taką sekwencję znaków, która dopasowuje się do podanego jej wzorca. Wynik próby dopasowania zwracany jest jako rezultat działania maszyny.



Współcześnie wyrażenia regularne są powszechnie używane do wyszukiwania napisów i manipulowania nimi, przy czym podejście do ich stosowania bywa bardzo różnorodne. W niektórych językach programowania wyrażenia regularne są dostępne wprost i stanowią część definicji języka (np. Perl, AWK), w innych dostępne są za pośrednictwem funkcji bibliotecznych (np. „C” i biblioteka *regex*) bądź specjalizowanych klas (np. Java i klasy *Pattern* i *Matcher*).

Poniżej podajemy podzbiór metaznaków języka wyrażeń regularnych oraz reguły ich dopasowywania w postaci akceptowanej przez większość narzędzi w systemach Unix/Linux. W większości przypadków podajemy także przykłady wzorców i napisów wraz z wynikiem dopasowania

Reguły dopasowania wzorca

1. Dopasowanie wzorca do łańcucha odbywa się zawsze od lewej do prawej
2. Jeżeli pewien znak wzorca nie jest metaznakiem, to dopasowuje się do napisu literalnie, tzn. dopasowanie wystąpi tylko wtedy, gdy na odpowiednim miejscu napisu znajdować się będzie dokładnie taki sam znak (ewentualnie z dokładnością do wielkości litery); w szczególności metaznakami są wszystkie cyfry i litery:

napis	*wzorzec*	*dopasowanie*
`Ala ma kota`	`kota`	tak
`Ala ma kota`	`koty`	nie

3. Metaznakami są znaki z następującego zbioru:
   - `.` (kropka)
   - `^` (caret)
   - `$` (dolar)
   - * (gwiazdka)
   - `[]` (nawiasy kwadratowe)
   - `<>` (nawiasy ostre)
   - `()` (nawiasy okrągłe)
   - `\` (backslash)
4. Jeżeli pewien metaznak ma zostać użyty jako literał, należy go poprzedzić znakiem `\` – jest to tzw. *cytowanie*.
5. Jeżeli za znakiem `\` stoi jeden ze znaków wymienionych poniżej, to element taki dopasowuje się do jednego ze znaków sterujących:
   - `\n` znak końca linii (`LF`)
   - `\r` znak powrotu karetki (`CR`)
   - `\t` znak tabulatora
   - `\e` tzw. znak ucieczki (ang. *escape char*)
   - `\f` znak przejścia do nowej strony
6. Jeżeli za znakiem `\` znajduje się jeden ze znaków wymienionych poniżej, to element taki dopasowuje się do klasy znaków:
   - `\s` tzw. biały znak (ang. *whitespace*) – inaczej znak niewidoczny

- `\S` znak, który nie jest biały
- `\w` znak w słowie (litera, cyfra, podkreślenie)
- `\W` znak, który nie jest znakiem w słowie
- `\b` miejsce w napisie na granicy słowa
- `\B` miejsce w napisie pomiędzy znakami słowa

7. Metaznak `.` (kropka) dopasowuje się do dowolnego (jednego!) znaku w napisie:

napis	*wzorzec*	*dopasowanie*
`Ala ma kota`	`kot.`	tak
`Ala ma koty`	`kot.`	tak

8. Ujęcie listy znaków w nawiasy kwadratowe `[]` definiuje tzw. zbiór znaków; zbiór znaków dopasowuje się w pewnego znaku w napisie, gdy znak ten jest elementem zbioru:

napis	*wzorzec*	*dopasowanie*
`Ala ma trzy koty`	`kot[uy]`	tak
`Ala ma kota`	`kot[uy]`	nie
`Ala mówi kotu`	`kot[uy]`	tak

9. Jeżeli pierwszym znakiem zbioru jest `^`, to zbiór rozumiany jest jako swoje **dopełnienie**, tzn. traktowany jest tak, jakby składał się ze wszystkich tych znaków, których nie wymieniono:

napis	*wzorzec*	*dopasowanie*
`Ala ma trzy koty`	`kot[^uy]`	nie
`Ala ma kota`	`kot[^uy]`	tak
`Ala mówi kotu`	`kot[^uy]`	nie

10. Zbiór może być definiowany tzw. zakresem podawanym jako krańce dolny i górny, rozdzielone znakiem łącznika (`-`); zakres rozumiany jest jako wszystkie znaki, które w danym alfabecie leżą

pomiędzy wskazanymi krańcami wraz z tymi krańcami; jeżeli system operacyjny został prawidłowo zlokalizowany, zakres powinien obejmować sobą również ewentualnie istniejące pomiędzy krańcami znaki narodowe.

napis	*wzorzec*	*dopasowanie*
`Ala ma 12 kotów`	`1[1-5]`	tak
`Ala ma 16 kotów`	`1[1-5]`	nie
`Ala ma 10 kotów`	`1[1-5]`	nie

11. Jeden zbiór może zawierać dowolnie wiele zakresów, a zakresy mogą być rozdzielane pojedynczymi elementami:

napis	*wzorzec*	*dopasowanie*
`x=0xf`	`0x[0-9a-f]`	tak
`x=0xf`	`0x[0-9A-F]`	nie
`x=0xf`	`[0-9abc-f]`	tak

12. Jeżeli wewnątrz zbioru umieszczony jest element postaci `[:klasa:]`, to element taki dopasowuje się do odpowiedniego znaku w napisie wtedy, gdy znak ten należy do wskazanej klasy; definiuje się następujące klasy i ich nazwy:

nazwa	klasa	zbiór równoważny
`[:alnum:]`	litera (mała lub wielka) lub cyfra dziesiętna	`[a-zA-Z0-9]`
`[:alpha:]`	litera (mała lub wielka)	`[a-zA-Z]`
`[:ascii:]`	znak kodu ASCII	`[\x00-\x7F]`
`[:blank:]`	spacja lub tabulator	[\t]
`[:cntrl:]`	znak sterujący	`[\x00-\x1F\x7F]`
`[:digit:]`	cyfra dziesiętna	`[0-9]`
`[:lower:]`	mała litera	`[a-z]`
`[:graph:]`	znak widoczny (każdy oprócz białych i sterujących)	`[\x21-\x7E]`
`[:print:]`	znak drukowalny (każdy oprócz sterujących)	`[\x20-\x7E]`

`[:punct:]`	znak przestankowy	`[!"#$%&'()*+,`
		`\-./:;<=>?@`
		`[\\\]^_`{|}~]`

`[:space:]`	biały znak	`[ \t\r\n\f]`
`[:upper:]`	wielka litera	`[A-Z]`
`[:xdigit:]`	cyfra szesnastkowa	`[A-Fa-f0-9]`

Na przykład:

napis	*wzorzec*	*dopasowanie*
`Ola`	`[[:upper:]]la`	tak
`bla`	`[[:upper:]]la`	nie
`dla`	`[[:lower:]]la`	tak

13. Metaznak `^` dopasowuje się nie do znaku, a do miejsca, które leży przed pierwszym znakiem napisu:

napis	*wzorzec*	*dopasowanie*
`Ala ma kota`	`^Ala`	tak
`Ala ma kota`	`^kota`	nie

14. Metaznak `$` dopasowuje się nie do znaku, a do miejsca, które leży za ostatnim znakiem napisu:

napis	*wzorzec*	*dopasowanie*
`Ala ma kota`	`Ala$`	nie
`Ala ma kota`	`kota$`	tak

15. Metaznaki `\<` i `\>` dopasowują się do miejsc, które leżą odpowiednio przed pierwszym i za ostatnim znakiem słowa (tzn. na granicach słowa):

napis	*wzorzec*	*dopasowanie*
`To jest rzeka`	`\<rzeka\>`	tak
`On narzekał.`	`\<rzeka\>`	nie

16. Metaznak `?` oznacza, że poprzedni element wzorca występuje w napisie opcjonalnie (dokładnie zero razy lub raz):

napis	*wzorzec*	*dopasowanie*
`To jest kot`	`koty?`	tak
`To są koty`	`koty?`	tak

17. Metaznak * oznacza, że poprzedni element wzorca nie występuje w napisie lub występuje dowolną liczbę razy:

napis	*wzorzec*	*dopasowanie*
`Oto__kot`	`Oto_*kot`	tak
`Otokot`	`Oto_*kot`	tak

18. Metaznak `+` oznacza, że poprzedni element wzorca występuje w napisie co najmniej raz:

napis	*wzorzec*	*dopasowanie*
`Oto__kot`	`Oto_+kot`	tak
`Otokot`	`Oto_+kot`	nie

19. Element postaci `{n}` gdzie *n* jest liczbą naturalną, oznacza, że poprzedni element wzorca występuje w napisie dokładnie *n* razy:

napis	*wzorzec*	*dopasowanie*
`12345`	`[[:digit:]]{5}`	tak
`1234`	`[[:digit:]]{5}`	nie

20. Element postaci `{n,m}` gdzie *n* i *m* są liczbami naturalnymi oraz $n \leq m$ oznacza, że poprzedni element wzorca występuje w napisie co najmniej *n* razy i nie więcej niż *m* razy

napis	*wzorzec*	*dopasowanie*
`1`	`[[:digit:]]{2,3}`	nie
`123`	`[[:digit:]]{2,3}`	tak

21. Element postaci `{n,}` gdzie *n* jest liczbą naturalną, oznacza, że poprzedni element wzorca występuje w napisie co najmniej *n* razy

napis	*wzorzec*	*dopasowanie*
`1`	`[[:digit:]]{2,}`	nie
`123`	`[[:digit:]]{2,}`	tak

22. Metaznak `|` jest operatorem alternatywy i oznacza, że w napisie ma wystąpić jeden z elementów stojących na lewo i prawo od tego operatora:

napis	*wzorzec*	*dopasowanie*
`Ala`	`O|Ala`	tak
`Ula`	`O|Ala`	nie

Polecenie grep

Niektóre cechy polecenia `grep` poznałeś już w czasie poprzednich zajęć, kiedy to używałeś go w charakterze filtra. Dzisiaj zapoznasz się z jego kolejnymi możliwościami. Należy tu zaznaczyć, że historycznie `grep` występował w kilku odmianach różniących się sposobem działania – były to:

- `grep` wersja podstawowa
- `egrep` wersja akceptująca wyrażenia regularne
- `fgrep` wersja akceptująca wzorce wyszukiwania podawane jako kolejne wiersze

Obecnie wszystkie te zadania wykonuje jedno narzędzie (`grep`) uruchamiane z odpowiednimi przełącznikami (np. `-E` włącza tryb `egrep`), jednak dla zachowania zgodności ze starymi konwencjami w systemie istnieją zwykle również pozostałe warianty.

Polecenie `grep`, oprócz pracy w potoku, kiedy to analizuje przepływający przez nie strumień znaków, potrafi przeglądać wskazane pliki i dokonywać w nich przeszukań z wykorzystaniem wyrażeń regularnych. W trybie domyślnym `grep` wyprowadza na swoje standardowe wyjście wszystkie linie plików wejściowych, które albo zawierają poszukiwanych ciąg znaków albo dają się dopasować do wskazanego wyrażenia regularnego. W takim wariancie `grep` uruchamiany jest w następujący sposób:

```
grep [przełącznik...] wzorzec [plik...]
```

gdy wzorzec jest prostym ciągiem znaków

```
grep [przełącznik...] -E wzorzec [plik...]
```

gdy wzorzec ma postać wyrażenia regularnego

Najwygodniejszym sposobem zapisu wzorca jest ujęcie go w apostrofy bądź cudzysłowy. W przeciwnym przypadku należy się liczyć z tym, że niektóre z metaznaków wyrażeń regularnych mogą być fałszywie interpretowane przez powłokę i wtedy konieczne będzie ich cytowanie za pomocą znaku `\` (niezależnie od cytowania wymuszanego w wyrażeniach regularnych).

Oprócz przełączników poznanych na poprzednich zajęciach akceptowane są również poniższe:

- `-f plik`

pobierz wzorce nie z linii poleceń, a z pliku o wskazanej nazwie

- `-x`
  wybieraj tylko te linie, które w całości dopasowują się do wzorca
- `-L`
  zamiast domyślnej postaci wyjścia wygeneruj listę nazw plików, z których nie wyprowadzono by żadnej pasującej linii
- `-l`
  zamiast domyślnej postaci wyjścia wygeneruj listę nazw plików, z których wyprowadzono by pasujące linie
- `-m` *num*
  przerwij czytanie pliku wejściowego po znalezieniu *num* dopasowań
- -o
  wypisuj tylko te części linii, dla których znaleziono dopasowanie (normalnie wypisywane są całe linie)
- `-s`
  wyłącz komunikaty o błędach w dostępie do przeglądanych plików
- `-b`
  przed każdą prezentowaną linią wyprowadź licznik bajtów
- `-H`
  przed każdą linią wyprowadź nazwę zawierającego ją pliku
- `-h`
  nie wyprowadzaj nazw plików (zachowanie domyślne, gdy przeszukiwany jest tylko jeden plik)
- `-n`
  przed każdą prezentowaną linią wyprowadź licznik linii
- `-a`
  przetwarzaj pliki binarne tak, jakby zawierały tekst
- `--exclude=wzorzec_nazwy`
  pomijaj pliki, których nazwy pasują do wzorca (uwaga – to nie jest wyrażenie regularne, a prosta maska nazwy pliku ze znakami `?` i * w tradycyjnym znaczeniu)
- `--exclude-dir=wzorzec_nazwy`
  pomijaj katalogi, których nazwy pasują do wzorca (uwaga jak wyżej)
- `-r`
  przetwarzaj katalogi rekursywnie; honoruj linki symboliczne tylko wtedy, gdy zostały jawnie wymienione w linii poleceń

- `-R`

  przetwarzaj katalogi rekursywnie; honoruj wszystkie linki symboliczne
- `-w`

  szukaj tylko linii, w których wzorzec dopasowuje się do całych słów
- `-v`

  wyprowadzaj tylko te linie, które **nie pasują** do wzorca

Przykłady:

W zaprezentowanych poniżej przykładach zakładamy istnienie plików o nazwach `test1` i `test2` oraz o następującej zawartości:

```
SAME WIELKIE LITERY
same małe litery
Linia Wielbłądzia Czyli Kapitaliki


powyżej są 2 puste linie
ostatnia linia
```

Poniżej podajemy kilka przykładów użycia polecenia `grep`, prezentując kompletne polecenia i dane wyprowadzane na standardowe wyjście:

- ```
  grep "litery" test1
  same małe litery
  ```

- ```
  grep "li" test1
  same małe litery
  Linia Wielbłądzia Czyli Kapitaliki
  powyżej są 2 puste linie
  ostatnia linia
  ```

- ```
  grep "litery" test*
  test1:same małe litery
  test2:same małe litery
  ```

- ```
  grep -i "litery" test1
  SAME WIELKIE LITERY
  same małe litery
  ```

- ```
  grep -E "lini." test1
  powyżej są 2 puste linie
  ostatnia linia
  ```

- ```
  grep -E "[[:digit:]]" test1
  powyżej są 2 puste linie
  ```

- ```
  grep -E "^$" test1
  --pusta linia--
  --pusta linia--
  ```
  ```

- `grep -v -E "^$" test1`
```
SAME WIELKIE LITERY
same małe litery
Linia Wielbłądzia Czyli Kapitaliki
powyżej są 2 puste linie
ostatnia linia
```

- `grep -c -v -E "^$" test1`
```
5
```

- `grep -l -i "li" test*`
```
test1
test2
```

- `grep -b -i "li" test*`
```
test1:0:SAME WIELKIE LITERY
test1:20:same małe litery
test1:38:Linia Wielbłądzia Czyli Kapitaliki
test1:77:powyżej są 2 puste linie
test1:104:ostatnia linia
test2:0:SAME WIELKIE LITERY
test2:20:same małe litery
test2:38:Linia Wielbłądzia Czyli Kapitaliki
test2:77:powyżej są 2 puste linie
test2:104:ostatnia linia
```

Polecenie find

Polecenie `find` służy do wyszukiwania plików, których metadane spełniają określone kryteria.

```
find [-H] [-L] [-P] [katalog...] [wyrażenie]
```

`find` będzie szukał plików o cechach określonych przez wyrażenie, a proces szukania będzie kolejno rozpoczynany w każdym z wymienionych katalogów

Pierwsze trzy opcje, z którym może wystąpić co najwyżej jedna (jeśli została podana, musi poprzedzać jakiekolwiek inne elementy linii poleceń) determinują sposób, w jaki `find` traktuje dowiązania symboliczne:

- `-H`

  ignoruj dowiązania symboliczne (zachowanie domyślne)

- `-L`

  honoruj dowiązania symboliczne traktując je tak, jakby były obiektami, na które wskazują

- `-P`

  honoruj dowiązanie symboliczne tylko wtedy, gdy zostały jawnie wymienione w linii poleceń

Wyrażenie jest kombinacją warunków domyślnie połączonych spójnikiem „i". `Find` umożliwia określenie innego sposobu wiązania warunków za pomocą nawiasów okrągłych i jawnych operatorów logicznych. Po znalezieniu pliku spełniającego podane warunki `find` wypisuje jego nazwę na swoje standardowe wyjście. Dodatkowo, w takiej sytuacji `find` może wykonać pewną akcję (np. uruchomić wskazane polecenie).

Argumenty numeryczne będące składnikami warunków zapisuje się w sposób następujący

- `+n`

  większe od *n*
- `-n`

  mniejsze od *n*
- `n`

  równe *n*

  Warunki, odwołujące się do metadanych pliku określających czas, mają przedrostek:
  - `a` gdy chodzi o *atime* (czas dostępu)
  - `c` gdy chodzi o *ctime* (czas utworzenia)
  - `m` gdy chodzi o *mtime* (czas modyfikacji)

  Przedrostek taki będziemy umownie oznaczać jako **T**.

  Wybrane warunki sprawdzane przez polecenie `find`:
- `-`*min* `n` (gdzie *min* to jedno z: `amin`, `cmin`, `mmin`)

  czas *min* jest odległy o *n* minut od bieżącego
- `-`*newer* `nazwa_pliku` (gdzie *newer* to jedno z: `anewer`, `cnewer`, `mnewer`)

  odnaleziony plik jest nowszy niż plik o nazwie *nazwa_pliku*
- `-`*time* `n` (gdzie *time* to jedno z: `atime`, `ctime`, `mtime`)

  czas *time* jest odległy od bieżącego o *n*24* godziny (stosuje się zaokrąglenie w górę, które oznacza, że np. `-atime +1` wskazuje plik, który był użyty przed dwoma dniami albo wczesniej)
- `-empty`

  plik/katalog jest pusty
- `-executable`

  plik jest wykonywalny lub katalog jest „*przeglądalny*"
- `-gid n`

plik należy do grupy o numerze n

- `-group g`

  plik należy do grupy o nazwie g

- `-name nazwa`

  nazwa odnalezionego pliku pasuje do wzorca nazwa (wielkość liter ma znaczenie)

- `-iname nazwa`

  nazwa odnalezionego pliku pasuje do wzorca nazwa (wielkość liter nie ma znaczenia)

- `-perm prawa_pliku`

  odnaleziony plik ma dokładnie taka prawa jak określone w *prawa_pliku*

- `-readable`

  odnaleziony plik jest odczytywalny

- `-size n[cwbkMG]`

  odnaleziony plik ma rozmiar *n* jednostek, gdzie:

  - `b` blok 512-bajtowy
  - `c` bajt
  - `w` słowo dwubajtowe
  - `k` kilobajt (dokładniej: *kibibajt*)
  - `M` megabajt (dokładniej: *mebibajt*)
  - `G` gigabajt (dokładniej: *gibibajt*)

- `-type c`

  odnaleziony plik jest (wymieniono podstawowe):

  - `d` katalogiem
  - `f` zwykły plikiem
  - `l` dowiązaniem symbolicznym

- `-uid n`

  plik należy do użytkownika o numerze *n*

- `-user u`

  plik należy do użytkownika o nazwie *u*

- `-writable`

  plik jest zapisywalny

Dodatkowo definiuje się pewną liczbę pseudo-warunków, których zadaniem nie jest badanie cech pliku, a wykonanie specyficznej akcji. Podajemy dwa z nich:

- `-exec` *`polecenie`* `\;`

wykonuje dowolne polecenie w miejscu odnalezienia pliku; traktowany jest jako spełniony gdy polecenie kończy się poprawnie (zwraca kod powrotu 0); w poleceniu można używać symbolu `{}` oznaczającego nazwę odnalezionego pliku/katalogu

- `-print`

  traktowany tak, jakby zawsze był spełniony; powoduje wypisanie nazwy odnalezionego pliku do standardowego wyjścia; używany, gdy na skutek użycia innych warunków (np. `-exec`) domyślne wypisywanie nazw jest wyłączone

Powyższe warunki można dowolnie wiązać poniższymi operatorami (wymienionymi w kolejności malejących priorytetów):

- `( wyrażenie )`

  nawiasy – wymuszają zmianę wiązania operatorów; ponieważ powłoka może używać ich dla swoich celów, bądź przygotowany na konieczność użycia cytowania `\(` i `\)`
- `! wyrażenie`

  negacja – tu również przygotuj się na użycie cytowania
- `-not wyrażenie`

  jak wyżej, ale wygodniej, bo bez cytowania
- `wyrażenie1 wyrażenie2`

  złączenie wyrażeń operatorem „i”
- `wyrażenie1 -a wyrażenie2`

  jak wyżej, ale jawnie (`-a` jak *and*)
- `wyrażenie1 -o wyrażenie2`

  złączenie wyrażeń operatorem „lub” (`-o` jak *or*)

Przykłady:

- `find ~ -name abc.txt`

  wyszukuje wszystkie pliki/katalogi o nazwie `abc.txt`, które mogą znajdować się w katalogu domowym użytkownika oraz w jego podkatalogach

- `find ~/temp -name ”*.txt”`

  wyszukuje wszystkie pliki/katalogi o nazwie z przyrostkiem `.txt`, które znajdują się w katalogu `temp` w katalogu domowym użytkownika oraz jego ewentualnych podkatalogach

- `find ~ -iname ”*.txt” -type f -size +10M`
  wyszukuje wszystkie pliki zwykłe w katalogu domowym użytkownika (i jego podkatalogach), których nazwy mają przyrostek `.txt` (wielkość liter bez znaczenia) oraz rozmiar większy niż 10 mebibajtów

- `find /tmp -type f -atime +2 -exec rm {} \; -print`
  wyszukuje wszystkie pliki zwykłe w katalogu `/tmp` oraz jego podkatalogach, których nie używano w żaden sposób w ciągu ostatnich 48 godzin; usunie wszystkie odnalezione pliki zostaną usunięte; dodatkowy przełącznik -print powoduje, że zostaną wyświetlone nazwy odnalezionych plików, pomimo wykonania na nich dodatkowej operacji (tutaj `rm`).

Zadania do wykonania w czasie zajęć

Wskazówka: do testowania swoich wyrażeń regularnych możesz wykorzystać taki oto prosty potok:

```
echo napis | grep -E 'wzorzec'
```

1. W jaki sposób umieścić znak `[` wewnątrz zbioru?
2. Czy nasz system operacyjny jest zlokalizowany w sposób, który sprawia, że litera 'ą' jest elementem zbioru `[a-c]`?
3. Jakie znaki w napisie mogą dopasować się do klasy `[:cntrl:]`?
4. Korzystając z podręcznika systemowego zbadaj sposób użycia narzędzia `fgrep`.
5. Od jakiej wartości startują liczniki prezentowane przez przełączniki `-n` i `-b` polecenie `grep`?
6. Zbadaj działanie przełączników `-A` i `-B` polecenia `grep`
7. Skonstruuj wyrażenia regularne sprawdzające, czy plik zawiera:
   - numery PESEL
   - kody pocztowe
   - adres IP
   - adres e-mail
   - numer konta bankowego ze spacjami rozdzielającymi lub bez nich
   - datę w formacie ISO 8601
   - puste linie
   - linie nie zawierające cyfr

- linie potencjalnie zawierające imiona i nazwiska (dwa słowa obok siebie, rozdzielone białymi znakami, każde zaczynające się wielką literą)
- znaki końca linii w standardzie systemu MS Windows
- linie zaczynające się od liczb
- linie nie kończące się liczbami
- linie zawierające dokładnie jedną liczbę

8. Zbadaj działanie warunków `-maxdepth` i `-mindepth` polecenia `find`
9. Skonstruuj linię poleceń uruchamiające narzędzie `find` i odszukującą pliki o następujących cechach:
   - nieużywane od 2 lat
   - mniejsze od 1kB i większe od 100B
   - większe od 1kB lub mniejsze od 100B
   - takie, których właścicielem jest ktoś inny
   - wykonywalne w systemie MS Windows
   - większe niż 100MB i użyte w ciągu ostatniego miesiąca
   - uruchamiane w ciągu ostatniego tygodnia
10. Powiąż ze sobą polecenia `find` i `grep` używając warunku `-exec` i symbolu `{}` w celu znalezienia plików:
    - których nazwy kończą się przyrostkiem „txt” i które zawierają adresy email
    - które są mniejsze niż 10KB i które sprawiają wrażenie, że zawierają kod źródłowy w języku C/C++ (*wskazówka: co zwykle zawierają pierwsze linie pliku źródłowego w tych językach?*)