



Politechnika  
Wroclawska

# Systemy operacyjne 2

Laboratorium nr 7  
Wyrażenia regularne

Szymon Datko  
[szymon.datko@pwr.edu.pl](mailto:szymon.datko@pwr.edu.pl)

Wydział Informatyki i Telekomunikacji,  
Politechnika Wroclawska

semestr letni 2022/2023



# W skrócie

- ▶ Odnajdywanie ciągów znaków o określonej strukturze w dużym tekście.



## Uwaga!

- ▶ Aby wykonać te zajęcia, należy upewnić się, że w systemie zainstalowana jest odpowiednia implementacja programu `awk` – o nazwie `gawk` (GNU `awk`).
- ▶ Domyślnie może być zainstalowana implementacja, która nie obsługuje poprawnie wszystkich wyrażeń regularnych – na przykład `mawk` (unikać!).

Część I

# Omówienie zagadnień

# Wyrażenia regularne

- Narzędzie pozwalające odnaleźć łańcuchy znaków, pasujące do wzorca.
- Wzorzec do znalezienia może stanowić pojedyncza litera lub zwykły napis.
  - ▶ Oprócz tego dostępny jest szereg rozbudowanych operatorów i symboli.
    - Definiowanie alternatywnych fragmentów (tj. A lub B).
    - Ograniczenie zakresu znaków w danym łańcuchu (np. małe litery).
    - Określenie krotności wystąpień jakiegoś wyrażenia (np. **trzy** cyfry).
    - Sprecyzowanie położenia fragmentu (względem początku/końca...).
  - ▶ Bardzo przydatny jest też mechanizm grupowania i wstecznych referencji.
    - Pozwala użyć wcześniej dopasowanego łańcucha jako części wzorca.
    - Praktyczny przykład: odnalezienie znacznika zamykającego w XML.
- Nie ma jednej jedynej rodziny wyrażeń regularnych.
  - ▶ Składnia większości zapisów działa mimo to zazwyczaj podobnie.
  - ▶ Różne narzędzia mogą implementować specjalne wzorce w inny sposób.
  - ▶ Powszechnie stosowane jest **PCRE** – Perl Compatible Regular Expressions.
  - ▶ <https://unix.stackexchange.com/questions/119905/why-does-my-regular-expression-work-in-x-but-not-in-y>
- Abstrahujemy tutaj od informatyki teoretycznej i gramatyki formalnej.

# Kiedy stosować? – dygresja

„lol, it works :DD” – zerni said on 06/12/05 15:40:06

```

''=-('      '(?{'      .(''      |'%)      .('['      ^-'')
.('''      |'!')      .('''      |',')      .'''.      '\\$'
.'==''      .('['      ^'+')      .('''      |'/')      .('['
^'+')      .'|'|'      .(';')      &'='')      .(';')      &'='')
.';-''      .'-'.      '\\$'      .';='')      .('['      ^(')
.('['      ^'.')      .('''      |''')      .('!'      ^'+')
.'_\\{'      .'\\$'      .';=(')      '\\$=|'      .'\\'|.      '^~^.'
).((('')|      ('/').).      '\\'''+(      '{'~[').      (''|''')      .(''|'/)
).('['~'/')      .('['~'/).      (''|',').      (''|('%')).      '\\'.\\'|.      (''~(')).
'\\'|.      (''~
'#').      '!!--'      '\\$=.\\'|      .{'~[').      (''|'/).      (''|'&').      (
{'~"\\[").      (''|"\\").      (''|"%").      (''|"%").      (''~(')).      '\\').\\'|.
({'~[').      (''|"√").      (''|"\\.").      {'~"\\[").      (''~"√").      (''|"(").      (
''|"%").      {'~"\\[").      (''~"\\,")      (''|"\\").      (''|"\\,")      (''|(',').
'\\'\\'|.+      (''~"+")      .['~"\\").      (''|"\\").      (''|"\\.").      (''~(/)).
'+_,'\\','.      {'~('[']).      ('\\$;!').      (''~"+")      .{'~"√").      (''|"!").      (
''|"+")      (''|"%").      .{'~"\\[").      (''|"√").      (''|"\\.").      (''|"%").      (
{'~"\\[").      (''|"%$").      (''|"√/").      .['~"\\,")      (''|(')).      ','.      ({'')^
[').      ("\\|^
'+).      ("\\|^
'|').      ("\\|^
'(').      ("\\|^
'(').      ("\\|^
[').      ("\\|^
'|').      ("\\|^
'/').      ("\\|^
[').      ("\\|^
'|').      ("\\|^
'|').      ("\\|^
'/').      ("\\|^
'|').      ("\\|^
'|').      ("\\|^
'$').      "\\,")      .(''~('+)).      '\\',_,'\\'|      .!''.      ("!|^
'+).      ("!|^
'+).      '\\'|.      (''~',).      (''|"\\(").      (''|"\\").      (''|"\\,")      .(
''|('%')).      '++\\$="}')      );$=('.')^      '-;$-!@|      (';$^=')^      [';$/='';

```

# Podstawowe operatory i wyrażenia

- $a$  – proste dopasowanie pojedynczego znaku (symbolu).
  - ▶ Szukamy czy w podanym łańcuchu znajduje się litera  $a$ .
- $abc$  – proste dopasowanie łańcucha znak po znaku.
  - ▶ Szukamy fragmentu łańcucha, rozpoczynającego się od litery  $a$ .
  - ▶ Następnie musi znaleźć się litera  $b$ , a za nią litera  $c$ .
  - ▶ Można to interpretować, jako logiczną koniunkcję kolejnych wyrażen ( $\text{AND}$ ).
- $ab|cde$  – alternatywa, dopasowanie łańcucha  $ab$  lub  $cde$ .
  - ▶ Koniunkcja ( $\text{AND}$ ) wiąże silniej, niż alternatywa ( $\text{OR}$  – operator  $|$ ).
  - ▶ Jest to co samo co zapisanie  $(ab)|(cde)$ .
- $a(b|c)de$  – nawiasy umożliwiają grupowanie fragmentów wyrażen.
  - ▶ Szukamy łańcucha, rozpoczynającego się od litery  $a$ .
  - ▶ Następnie musi wystąpić litera  $b$  lub litera  $c$ .
  - ▶ Dalej musi znaleźć się litera  $d$ , a za nią litera  $e$ .

# Określenie zakresu znaków

- `[abc]` – ustalenie zakresu dopuszczalnych znaków na danej pozycji.
  - ▶ W tym miejscu może wystąpić litera a, litera b lub litera c.
  - ▶ Poniekąd jest to analogiczne do zapisu `a|b|c`.
  - ▶ Kolejność i powtórzenia podanych znaków nie mają znaczenia.
    - Zapis `[abbbaa]` oznacza dokładnie to samo, co `[ab]`.
  - ▶ Przy pomocy znaku `-` (minus) można określić przedział znaków.
    - `[a-z]` to dopuszcza pojedynczą małą literę.
    - `[0-9]` to pojedyncza cyfra.
    - `[a-zA-Z0-9_-]` obejmuje małe i duże litery, cyfry oraz znaki `_` i `-`.
    - **Uwaga!** Zakres `[a-z]` często nie obejmuje znaków typu `ą`, `ę`, `ó`, itd.
  - ▶ W operatorze `[]` większość znaków specjalnych traci swoje znaczenie.
    - Na przykład `.`, `*`, `(` i `)` to po prostu kropka, gwiazdka i nawiasy.
- `[^abc]` – odwrotność `[abc]`, zakres **niedopuszczalnych** znaków.
  - ▶ W tym miejscu może wystąpić dowolny znak, inny niż litera a, b lub c.
  - ▶ Ustawienie znaku `^` w innym miejscu wyłącza to specjalne zachowanie.
    - Czyli `[a^bc]` dopuszcza jeden z czterech znaków: a, b, c oraz `^`.

## Ustalenie liczby powtórzeń symbolu

- Podane tu operatory odnoszą się do poprzedzającego je symbolu/grupy.
- $a\{3\}$  – dokładnie 3 wystąpienia znaku  $a$ .
  - ▶ Odpowiednik zapisu  $aaa$ .
- $[abc]\{2,4\}$  – od 2 do 4 wystąpień znaków z zakresu  $[a-c]$ .
  - ▶ Odpowiednik  $[abc][abc] \mid [abc][abc][abc] \mid [abc][abc][abc][abc]$ .
- $a(bc)?$  – oznaczenie opcjonalności, grupa  $bc$  może wystąpić 0 lub 1 razy.
  - ▶ Czyli pasującym fragmentem będzie napis  $a$  oraz  $abc$ .
- $[A-Z][0-9]^+$  – co najmniej jedna cyfra po dużej literze i znaku minusa.
  - ▶ W ten sposób można dopasować oznaczenia budynków naszej Uczelni.
- $a(bc)^*$  – dowolna liczba wystąpień (0 lub więcej) ciągu  $bc$  po literze  $a$ .
  - ▶ Pasujące napisy:  $a$ ,  $abc$ ,  $abcabc$ ,  $abcabcabc$ ; ale nie:  $ab$ ,  $abcb$ ,  $abcbcc$ .
  - ▶ Standardowo silniki usiłują znaleźć najdłuższe możliwe dopasowanie.
  - ▶ Dygresja – należy przemyśleć i unikać zapisów typu  $(ab^*)(b^*)!$



## Inne przydatne symbole i zapisy

- `.` – dowolny pojedynczy znak.
- `^` – początek przetwarzanego łańcucha.
  - ▶ Zwykle jest to po prostu początek linii (gdy przetwarzamy wiersze).
  - ▶ Na przykład `^[0-9]` dopasuje linie, rozpoczynające się od dowolnej cyfry.
- `$` – koniec przetwarzanego łańcucha.
  - ▶ Zwykle jest to po prostu koniec linii (gdy przetwarzamy wiersze).
    - W tym wypadku ostatnim znakiem **nie** jest znak nowej linii `\n`.
  - ▶ Zapis `a$` może dopasować więc linię, której ostatnim znakiem jest litera `a`.
- Sekwencja ucieczki `\` pozwala wyłączyć specjalnie znaczenie znaków.
  - ▶ Zapis `\*` oznacza wtedy to samo co `[*]`.
  - ▶ Analogicznie `[\[]` pozwala dopasować pojedynczy znak `[` lub `]`.
- Wiele implementacji wspiera też określenie klas znaków w operatorze `[]`.
  - ▶ Lista: [https://www.gnu.org/software/gawk/manual/html\\_node/Bracket-Expressions.html](https://www.gnu.org/software/gawk/manual/html_node/Bracket-Expressions.html)
  - ▶ Na przykład `[:lower:]` to to samo co `a-z`, ale może być czytelniejsze.
  - ▶ **Uwaga!** Użycie wygląda wtedy następująco w zapisie: `[[:lower:]]`.

# Stosowanie w typowych narzędziach

- Narzędzia `grep`, `sed` i `vi` używają wariantu Basic Regular Expressions.
  - ▶ Główna różnica polega na tym, że część operatorów wymaga dodania `\`.
  - ▶ Na przykład `\(...\)`, `\{...\}`, `|`, `\?`, `\+` zamiast `(...)`, `{...}`, `|`, `?`, `+`.
    - Jak na ironię, zapisy `.`, `*`, `^`, `$`, `[...]` pozostają bez zmian!
- Narzędzia `awk`, `grep -E` i `sed -E` używają Extended Regular Expressions.
  - ▶ Obejmują one zapisy, które zostały omówione na poprzednich slajdach.
- Narzędzie `grep -P` i wiele języków programowania obsługuje PCRE.
  - ▶ Obejmuje on szereg zaawansowanych mechanizmów, nieumówionych tu.
- W powłokach ścieżki dopasowanie są za pomocą mechanizmu `glob`.
  - ▶ Znaczenie części symboli w tym mechanizmie różni się od tu omawianych.
  - ▶ Niektóre powłoki oferują także rozszerzenia w tym temacie, np. `extglob`.
  - ▶ W powłoce **Bash** występuje operator `=~`, implementujący wariant ERE.
    - On akurat nie służy do dopasowywania ścieżek...
    - Stosowany jest do porównywania ciągów w konstrukcji `[[ ... ]]`.

# Użycie w programie awk

- Opcja `--re-interval`, aby działał operator określenia powtórzeń `{}`.
- Definiowanie separatorów.
  - ▶ Zawartość zmiennych `RS` i `FS` stanowi wyrażenie regularne.
  - ▶ Pozwala to elastycznie je definiować, np. domyślne `FS = "[\t\n]+"`.
- Wzorce: `/wyrażenie/` lub `zmienna ~ /wyrażenie/`
  - ▶ Akcje zostaną wykonane, jeśli rekord lub `zmienna` pasuje do wyrażenia.
  - ▶ Na przykład `/^$/ { ILE += 1 }` pozwala zliczyć puste rekordy.
- Funkcje do manipulacji na łańcuchach znaków.
  - ▶ `match(łańcuch, wyrażenie, tablica)`
    - Zwraca pozycję pierwszego pasującego znaku w łańcuchu, lub zero.
    - Opcjonalny argument `tablica` przechowuje dopasowany ciąg/grupy.
  - ▶ `sub(wyrażenie, zamiennik, cel)`
  - ▶ `gsub(wyrażenie, zamiennik, cel)`
    - Zamienia pasujący fragment tekstu w zmiennej `cel` (domyślnie `$0`).
    - Zwraca liczbę zmian – 0 lub 1 w `sub()`; 0 lub więcej w `gsub()`.
  - ▶ Więcej: [https://www.gnu.org/software/gawk/manual/html\\_node/String-Functions.html](https://www.gnu.org/software/gawk/manual/html_node/String-Functions.html)

# Narzędzia przydatne przy nauce i pracy

- <https://regexone.com>
  - ▶ Świetne, praktyczne i interaktywne, wprowadzenie do wyrażeń regularnych.
- <https://regex101.com>
  - ▶ Chyba najlepsze narzędzie do sprawdzania działania wyrażeń regularnych.
  - ▶ Na bieżąco wskazuje co zostało dopasowane i tłumaczy dlaczego.
  - ▶ Wspiera wiele wariantów / silników wyrażeń.
- <https://regexpr.com>
  - ▶ Trochę inne, ale wciąż bardzo dobre narzędzie do testowania wyrażeń.
- Porównanie narzędzi: <https://regexland.com/comparison-regex-testers/>

Część dla dociekliwych

# Zadanie dodatkowe

## Zadanie dodatkowe – dla zainteresowanych

### Monstrum, albo funkcji opisanie.

*„Zaprawdę, nie masz nic wstrętniejszego ponad monstra owe, naturze przeciwne, funkcjami zwane, bo są to płody plugawego skrypciarstwa i diabelstwa. Są to zapisy bez cnoty, właściciela i uprawnień, istne stwory piekielne, do błędów jedynie zdadne. Nie masz dla takich jak oni między skryptami pocziwymi miejsca. A owo `.bashrc`, gdzie ci bezecni się gnieźdzą, gdzie ohydnych swych definicji dokonują, starte być musi z powierzchni dysku, a ślad po nim solą i `shredem` posypyany.”*

Proszę omówić stosowanie funkcji w skryptach powłoki – jak je definiować, wywoływać i przekazywać do nich parametry oraz w jaki sposób można zwracać wyniki z funkcji. Czy można jakoś upewnić się, że jakaś funkcja jest zdefiniowana i ewentualnie wczytać ją z osobnego pliku? Jak usunąć zdefiniowaną funkcję? Czy mogą istnieć dwie funkcje o tej samej nazwie, ale różnej liczbie argumentów?