Systemy operacyjne – studia stacjonarne 2023/24
Lab 5. Skrypty powłoki bash cd., język programowania awk.

1. Język programowania awk - wstęp.

AWK jest narzędziem służącym do przetwarzania tekstu i generowania raportów według zdefiniowanych reguł.
Jak wiele unix'owych poleceń przetwarza strumień wejściowy lub pliki z danymi tworząc strumień wynikowy.
Nazwa AWK pochodzi od pierwszych liter nazwisk jego twórców: Aho, Weinberg, Kernighan
AWK jest bardzo wygodnym narzędziem, które umożliwia:

- wykonywanie operacji na ciągach znaków
- wykonywanie operacji arytmetycznych
- korzystanie z konstrukcji programowania strukturalnego

Jest interpreterem, dzięki czemu programy łatwo można uruchamiać pod innym systemem operacyjnym.
AWK można uruchomić w następujący sposób:

```
awk –f program.awk plik1 plik2 plik3 …
```

W tym przypadku awk przetwarza kolejno linie z plików występujących w linii poleceń: plik1, plik2 i kolejne, wykonując na nich program zapisany w pliku program.awk.
Można również uruchomić skrypt jak program dopisując w pierwszej linii komentarz:

```
#!/usr/bin/awk -f
```

Jeśli kod programu nie jest długi można umieścić go bezpośrednio w poleceniu pomiędzy znakami pojedynczego cudzysłowu i wywołać bezpośrednio z powłoki:

```
awk 'kod programu' plik1
```

Program języka awk składa się z par:

```
Wzorzec1 {akcja1}
Wzorzec2 {akcja2}
...
```

- Wzorzec występujący w programie awk może mieć postać wyrażenia regularnego (regular expression, krótko: regex), analogicznego do wyrażeń występujących w poleceniu grep
- Wzorzec jest wyrażeniem logicznym, które jest dopasowywane kolejno do każdej linii każdego z plików wejściowych;
- Jeśli wyrażenie jest prawdziwe, wykonywana jest akcja;
- Jeśli wzorzec nie występuje, akcja jest wykonywana dla każdej linii;
- Nawiasy klamrowe są wykorzystywane do wiązania ze sobą bloków kodu - podobnie jak w C.

Najprostszy wzorzec to wyrażenie regularne występujące między znakami /…./np.:

```
awk '/Anna/ {print $0}' /etc/passwd
```

- wyświetla linie z pliku /etc/passwd, w których występuje słowo Anna;
- Akcja print $0 oznacza wyświetlenie całej linii wejściowego pliku;
- Jeśli akcja jest pominięta wykonywana jest akcja domyślna, którą jest właśnie print $0;

Polecenie równoważne poprzedniemu:

```
awk '/Anna/' /etc/passwd
```

Jeśli w programie awk występuje więcej par: *wzorzec {akcja},* są one oddzielone średnikami lub znakiem końca linii.

Przykład. Wypisanie wierszy zawierających ciąg znaków anna lub piotr w pliku /etc/passwd:

```
awk -F: '/anna/ { printf "%-20s %s\n",$3,$1} ;
        /piotr/ { printf "%-20s %s\n",$3,$1}' /etc/passwd
```

- Opcja –F pozwala zdefiniować separator pól – w powyższym przykładzie jest to znak dwukropka;
- Funkcja printf pozwala sformatować wyjście podobnie jak w języku C, natomiast $1, $2,.. oznaczają kolejne pola linii tekstu wejściowego, przy zadanym separatorze;
- Akcja może również składać się z wielu poleceń, wówczas oddzielone są one średnikami lub znakiem nowej linii

Blok BEGIN i END.

W pisaniu programów awk bardzo przydatne jest stosowanie bloku BEGIN, w którym definiuje się akcje wykonywane przed przetwarzaniem tekstu wejściowego, oraz bloku END, który jest wykonywany na końcu, po przetworzeniu wszystkich wierszy.

Przykład. Zliczanie wierszy zawierających ciąg znaków anna lub piotr w pliku /etc/passwd:

```
awk -F: 'BEGIN {printf "\n\n";i=0;j=0}
        /anna/ { printf "%-20s %s\n",$3,$1;i++} ;
        /piotr/ { printf "%-20s %s\n",$3,$1;j++}
        END {printf "linii-%s, Ann-%s, Piotrów-%s\n", NR, i, j}'
/etc/passwd
```

W programie awk zmienne nie wymagają deklaracji i inicjowane są wartością 0, więc zwykle nie jest konieczne inicjowanie ich w bloku BEGIN

Wbudowana zmienna NR, wykorzystana w powyższym programie wskazuje numer aktualnie przetwarzanego wiersza – jej wartość na końcu programu jest równa liczbie przetworzonych wierszy

Awk może być wywołany w skrypcie powłoki

Jeśli w programie awk chcemy się odwołać do zmiennych skryptu, w tym również parametrów skryptu, możemy je przekazać za pomocą opcji –v

Poniższy program (skrypt) wyświetla informacje o wybranych użytkownikach systemu, lecz tym razem imiona są parametrami skryptu

Plik skrypt (zliczanie użytkowników o imionach jak parametry wywołania skryptu):

```
awk -v N1="$1" -v N2="$2" -F: 'BEGIN {printf "\n%s %s\n",N1,N2;i=0;j=0}
     {if (index($0,N1) != 0) { printf "%-20s %s\n",$3,$1;i++}}
     {if (index($0,N2) != 0) { printf "%-20s %s\n",$3,$1;j++}}
     END {printf "linii-%s, Ann-%s, Piotrów-%s\n", NR, i, j}'
/etc/passwd
```

Wywołanie skryptu:

```
bash skrypt piotr anna
```

- W programie wykorzystane zostały elementy zaczerpnięte z języka C: instrukcje arytmetyczne, instrukcja sterująca if oraz funkcja index, która wyszukuje w bieżącej linii (pierwszy parametr funkcji index ma wartość $0) ciąg znaków odpowiadający zmiennej N1 (drugi parametr) i zwraca jego położenie. Kolejne wywołanie funkcji index dotyczy zmiennej N2
- Innym sposobem odwołania się do parametrów skryptu bash w programie awk jest ujęcie ich w pojedyncze cudzysłowy: ‘$1’

```
awk -F: 'BEGIN {printf "\n\n";i=0;j=0}
     /'$1'/ { printf "%-20s %s\n",$3,$1;i++}
     /'$2'/ { printf "%-20s %s\n",$3,$1;j++}
     END {printf "linii-%s, Ann-%s, Piotrów-%s\n", NR, i, j}'
/etc/passwd
```

Przykład 1.1.

Program awk przetwarza wyjście polecenia ls –l, wybiera linie opisujące pliki zwykłe, do których członkowie grupy mają prawo czytania i sumuje rozmiary takich plików. Na standardowym strumieniu wyjścia wyświetla policzoną sumę.

```
ls -l | awk '/^-...r/ {x=x+$5};END {printf "%s\n",x}'
```

Przykład 1.2.

Skrypt badający zajętość file-systemów, dopisujący aktualne wyniki wraz z aktualną data i czasem do pliku wynik.

```
DAT=`date +"%Y-%m-%d %T"`
echo $DAT
df|awk -v DT="$DAT" '{if (NR != 1) printf "%s %-20s %-20s\n", DT,$6,$5}'
>> wynik
```

Przykład 1.3.

Trzy różne skrypty zapisujące do pliku wynik linie z pliku, będącego pierwszym parametrem skryptu, z zakresu określonego poprzez dwa pozostałe parametry wywołania.

```
awk -v z1=$2 -v z2=$3 'NR>z1 && NR<=z2' $1 >wynik
awk -v z1=$2 -v z2=$3 'NR>z1 && NR<=z2 {print $0 > "wynik"}' $1
cat $1|awk 'NR>'$2' && NR <='$3' {print $0 > "wynik"}'
```

2. Definiowanie zmiennych, zmienne wbudowane.

Zmienne nie mają typu, nie są deklarowane, traktowane są jako teksty lub liczby. Jeżeli wartość zmiennej jest ciągiem liczb na takiej zmiennej możemy wykonywać operacje matematyczne. Przypisanie do zmiennej zm wartości 15:

```
{zm = 15}
```

Następuje automatyczna konwersja z napisów na liczby (i odwrotnie). W wyrażeniach arytmetycznych wartość zmiennej tekstowej jest równa 0. Przykłady:

```
1 + "234" = 235
1 + "tekst" = 1
((2+2)(2+2))/4 = ((4)(4))/4 = (44)/4 = 11
zm = zm + 1   lub   zm++
```

Zmienne wbudowane (wewnętrzne):

NR	Numer obecnego rekordu, na końcu liczba wszystkich przeczytanych rekordów
FILENAME	Nazwa bieżącego pliku
FNR	Liczba wszystkich rekordów w obecnie przetwarzanym pliku
FS	Separator pola – domyślnie spacja
RS	Wejściowy separator rekordu – domyślnie znak nowej linii
OFS	Wyjściowy separator pola – domyślnie spacja
ORS	Wyjściowy separator wiersza – domyślnie znak nowej linii

NF	Liczba pól w bieżącym rekordzie
ARGC	Liczba argumentów przekazanych do skryptu
ARGV	Tablica wartości argumentów przekazanych do skryptu
RSTART	Początkowa pozycja łańcucha zwróconego przez dopasowanie (funkcja match)
RLENGTH	Długość łańcucha zwróconego w wyniku dopasowania wzorca (funkcja match)

3. Rekordy wielowierszowe.

Odpowiednio ustawiając zmienną FS, można przy pomocy awk odczytywać różnego rodzaju strukturalne dane pod warunkiem ,że każdy rekord zapisany jest w jednej linii. W niektórych sytuacjach konieczne jest ustawienie zmiennej separatora rekordów RS, która określa koniec jednego i początek kolejnego rekordu.

Przykładowy plik z danymi studentów.

```
Karol Górski
Ul. Krakowska 15
31-855, Kraków

Maria Jezierska
Aleja 29 listopada 13
00-920, Warszawa
```

Ustawiając zmienne FS i RS jak poniżej awk każdy z 3-wierszowych adresów rozpozna jako oddzielny rekord:

```
BEGIN {
    FS="\n"
    RS=""
}
{ print $1 ", " $2 ", " $3 }
```

Lub inaczej

```
BEGIN {
    FS="\n"
    RS=""
    OFS=", "
}
{ print $1, $2, $3 }
```

4. Tworzenie wzorców, wyrażenia regularne, warunki logiczne.

Wyrażenia regularne – składnia:

^	Dopasowanie do początku wiersza/pola
$	Dopasowanie do końca wiersza/pola
.	Zastępuje jeden dowolny znak
*	Zero lub więcej wystąpień tego (znaku) co przed gwiazdką
+	Jedno lub więcej wystąpień tego (znaku) co przed plusem
?	Zero lub jedno powtórzenie tego co przed pytajnikiem
[abcd]	Jeden znak z listy wewnątrz nawiasów
[^abcd]	Jeden znak spoza listy (negacja listy)
.*	Dowolny ciąg znaków

|	Alternatywa wzorców
()	Grupowanie podwyrażeń

Przykłady.

Program wypisujący linie zawierające tylko liczby dodatnie:

```
awk '/^[0-9]+(\.[0-9]+)?$/ {print}' pl
```

Program wypisuje linie zawierające ciąg znaków jan lub piotr:

```
awk '/jan|piotr/ {print}' /etc/passwd
```

Zamiast wzorca mogą pojawiać się warunki arytmetyczne, relacyjne, logiczne oraz funkcje matematyczne. W zapisie stosujemy odpowiednie operatory, tak jak w C.

Operatory arytmetyczne:

```
+, -, *, /, %,^
```

Operatory relacyjne:

```
==, !=, >, <, >=, <=
```

Operatory logiczne:

```
&&, ||, !
```

Funkcje matematyczne:

```
sin(x), cos(x), log(x), sqrt(x), exp(n), rand(), srand(x), int
```

Dopasowanie w warunkach – operatory **~**, **!~**:

`wzorzec ~ tekst` wzorzec pasuje do tekstu

`wzorzec !~ tekst` wzorzec nie pasuje do tekstu

Zatem program języka awk można zapisać:

`wzorzec {akcja}` lub równoważnie `$0 ~ wzorzec {akcja}`

Przykłady zastosowania operatorów – podstawowe polecenia awk:

'NR>5 && NR <15'	Wypisze wiersze od 6 do 14
'NR < 15 && /Anna/'	Wypisze wiersze, których numer jest <15 i zawierają wzorzec Anna
'NR == 12'	Wypisze 12 wiersz z pliku
'NR == 10, NR == 20'	Wypisuje wiersze od numeru 10 do 20 włącznie
'$3 == 0 {print $1}'	Jeżeli zawartość 3 pola równa się 0 wypisz pole nr 1
'!($4 == 5 && $1 ~ /Anna/)'	Wypisze wiersze nie spełniające warunku podanego w nawiasie
'$3 > 10 {print } '	Jeżeli wartość numeryczna 3-ego pola większa od 10 – wypisz ten wiersz
'$3 > "10" {print } '	Porównanie łańcuchów z 3-ego pola i tekstu 10 – znak po znaku
'$3 ~ /A/ {print $1,$4}'	Jeżeli 3-cie pole zawiera ”A” – wypisz pierwsze i czwarte pole
'$3 !~ /A/ {print $1,$4}'	Jeżeli 3-cie pole nie zawiera ”A” – wypisz pierwsze i czwarte pole
'$1=$1'	Usuwa dodatkowe spacje pomiędzy polami i puste wiersze
'{$1=$1;print}'	Usuwa dodatkowe spacje pomiędzy polami i pozostawia puste wiersze
'/^$/'	Wypisuje wszystkie puste wiersze
'NF'	Usuwa wszystkie puste wiersze
'{print $NF}'	Wypisuje ostatnie pole dla każdego wiersza

5. Instrukcje sterujące.

Instrukcje pętli oraz instrukcję warunkową zapisujemy identycznie jak w C.

Pętla for

```
for ( i = 0; i <= 4; i++ ) {
    print "iteracja_",i
}
```

Pętla while

```
i=0
while (i <= 4) {
    print "iteracja_",i
    i++
}
```

Pętla do-while

```
{
    count=1
    do {
        print "wydruk co najmniej raz"
    } while ( count != 1 )
}
```

Instrukcja warunkowa **if-else**

```
{
    if ( $1 > $2 ) {
        print $1,”>”,$2
    }
    else if ($1 == $2 ) {
        print $1,”=”,$2
    }
    else {
        print $1,”<”,$2
    }
}
```

Break, continue

```
i=1
while (1) {
    if ( i == 4 ) {
        i++
        continue
    }
    print "iteracja_",i
    if ( i > 20 ) {
        break
    }
    i++
}
```

Next – przejście do następnego rekordu.

```
{print "A";
next;
print "B"}
```

Polecenie print ”B” nigdy się nie wykona.

6. Tablice.

Cechy tablic:

- Tablice tworzone dynamicznie;
- Tablic nie trzeba deklarować;
- Zwyczajowo indeksowanie tablic od 1;
- Można tworzyć tablice wielowymiarowe `(tab[i,j]);`
- Usunięcie elementu tablicy poleceniem delete np.: `delete tab[1];`
- Sprawdzenie czy istnieje element tablicy poleceniem `in;`
- Można tworzyć tablice asocjacyjne – indeksowane słowami;
- Pętla for dla tablic asocjacyjnych: `for (i in tab) { wykonaj coś na tab[i]}`

Przykład 1. Odwracanie kolejności wierszy w pliku.

```
{ wiersze[NR] = $0 }
END {
    licznik=NR;
    while (licznik > 0) {
        print wiersze[licznik];
        licznik--;
    }
}
```

Przykład 2. Zliczanie liczby słów w pliku.

```
{ for (i=1; i<=NF; i++)
    slowa[$i]++ }
END { for (i in slowa)
    {print i, slowa[i] }
}
```

Uwaga: $i to słowo na i-tej pozycji w danym wierszu

Przykład 3. Sprawdzenie czy w tablicy wiersze istnieje element.

```
{if ( 3 in wiersze ) {
    print "Jest."
} else {
    print "Nie ma."
}
}
```

7. Funkcje wykonujące operacje na tekstach.

length(s) Zwraca liczbę znaków w tekście s (bez znaku \n)

index(s,t)	Zwraca indeks początku łańcucha t w tekście s lub 0 jeżeli nie ma łańcucha t w s
match(s,r)	Zwraca indeks początku dopasowania wzorca r w tekście s lub 0 jeżeli dopasowanie nie występuje. Ustawia wartości zmiennych RSTART i RLENGTH
gsub(r,s,t)	Podstawia ciąg tekstowy s w ciągu t za każdym razem, gdy pasuje wyrażenie regularne r i zwraca liczbę dokonanych podstawień.
sub(r,s,t)	Zachowanie podobne do gsub, zastępowane jest tylko pierwsze dopasowanie
substr(s,i,n)	Zwraca fragment tekstu o długości n z tekstu s rozpoczynający się od indeksu i – numeracja od 1. Jeżeli n nie zostanie podane, wtedy zwróci tekst od podanego indeksu do końca wiersza.
split(s,T,fs)	Dzieli tekst s na części, które zapisuje w tablicy T zgodnie ze zdefiniowanym separatorem pola fs. Funkcja zwraca liczbę pól na którą podzielono tekst s. Jeżeli fs pomięto, wtedy wykorzystany jest domyślny separator pola FS
tolower(s)	Zamienia wszystkie znaki w s na małe litery
toupper(s)	Zamienia wszystkie znaki w s na duże litery

Przykład 1. Liczba znaków w pliku (łącznie z \n)

```
{len = len + 1 + length($0)}
END {print "liczba znaków w pliku = ", len}
```

8. Funkcje definiowane przez użytkownika.

Definicja funkcji podobnie jak w C. Definicja funkcji składa się ze słowa kluczowego function, nazwy funkcji, argumentów przekazanych do funkcji i ciała funkcji. Parametry przekazywane przez wartość. Zwracanie wartości poleceniem return.

```
function fun(a, b) {
    a = b + 5
    return a
}
```

9. Zadania.

Zad.1. Napisz skrypt, który zwróci liczbę słów w pliku będącym argumentem wywołania.

Zad.2. Napisz skrypt, który w zależności od wybranej opcji (pierwszy argument wywołania) zwróci liczbę znaków, słów i linii w pliku będącym drugim argumentem wywołania skryptu (jak polecenie wc).

Wejście/wywołanie programu:

```
./zad2.awk -cwl plik1
```

Wyjście/wynik:

```
liczba znaków w pliku =  932
liczba słów w pliku =  25
liczba wierszy w pliku =  18
```

Zad.3. Plik z danymi pojazdów zawiera następujące informacje: nazwa_kategorii, rok_produkcji, cena. Przykładowy plik z danymi:

```
auto    1999       2000
rower   2017       4500
auto    2011       32000
rower   2010       350
motor   2009       1200
auto    2022       85000
motor   2019       16000
```

Napisz skrypt, .który z podanego pliku z danymi dla określonej kategorii i podanego roku produkcji wyświetli maksymalną cenę pojazdu wyprodukowanego po podanym roku.

Wejście/wywołanie programu:

```
./zad3.awk dane_3 auto 2009
```

Wyjście/wynik:

```
Maksymalna cena = 85 000
```

Zad.4. Plik z danymi zawiera następujące informacje o dostępnych towarach:
nazwa_kategorii, nazwa_towaru, cena, liczba_dostępnych_sztuk.
Przykładowy plik z danymi:

```
kosmetyk krem 45 2
zabawka samolot 30 11
zabawka auto 100 10
zabawka robot 200 90
zabawka lego 67  0
kosmetyk perfumy 200 2
```

Poniższy skrypt wybiera z podanej kategorii dostępny towar w podanym przedziale cenowym. Argumentem wywołania skryptu jest nazwa pliku zawierającego spis dostępnych towarów.

```
echo -n "kategoria: "; read kategoria
echo -n "cena minimalna: "; read cenamin
echo -n "cena maksymalna: "; read cenamax
awk '/'$kategoria'/ {if (($4>0) && ($3>='$cenamin') && ($3<='$cenamax'))
    printf( "%-20s %-20s\n", $2, $3)}' $1
```

Wykorzystując powyższe napisz skrypt, .który korzystając z podanego pliku z danymi, wyświetli informację ile towarów z zadanej kategorii można kupić za podaną kwotę (uwzględniając dostępne towary). Skrypt działa w pętli, kończy się gdy użytkownik nie poda kategorii.

Wejście/wywołanie programu:

```
./zad4.awk 250 dane_4
```

Wyjście/wynik:

```
podaj kategorie: zabawka
samolot             : cena jednostkowa: 30      ile sztuk: 8
auto                : cena jednostkowa: 100     ile sztuk: 2
robot               : cena jednostkowa: 200     ile sztuk: 1
```

Zad.5. Napisz skrypt, który wypisze posortowaną listę zalogowanych użytkowników z podaniem numeru IP komputera i liczby sesji uruchomionych z danego IP. Wyniki wyświetli na ekranie i zapisze w pliku o nazwie LOGI_{data} (np. dla raportu poniżej LOGI_{2022-11-07 19:40:21})

Wejście/wywołanie programu:)

```
./zad5.awk
```

Wyjście/wynik:

```
Lista zalogowanych użytkowników dnia 2022-11-07 19:40:21
Lp.     NAZWISKO       IMIĘ                      HOST   LICZBA
  1     AGABALAEV      PAVEL           149.156.51.230    1
  2     BAFIA          WOJCIECH        185.209.178.20    3
  3     GODFRYD        MARCIN          109.197.175.99    2
```

Zad.6. Plik w którym zapisano oceny/punkty poszczególnych studentów zawiera następujące informacje:
imię, nazwisko oraz kolejne kolumny z ocenami/punktami oddzielone spacjami/tabulacjami.
Liczba ocen może być dowolna – np. student nie pisał kolokwium wówczas w danej kolumnie pozostaje wolne miejsce (zakładamy, że wszystkie oceny znajdują się w jednym wierszu). Przykładowy plik z danymi:

```
Karolina Jezierska 5 4 5 5 3 5
Maria Dobrowolska 2 3 4 5 5 4 3 2
Dariusz Nowacki 5     3     5
Sebastian Ziemowit 4 4 5 5 4 5 3 2 5
Piotr Nowak
Janusz Nawrocki 5 4 3   2 2
```

Napisz skrypt, który z podanego pliku z danymi utworzy posortowaną po nazwisku listę studentów zawierającą nazwisko imię, sumę ocen/punktów oraz średnią. Wyniki wyświetli na ekranie i zapisze w pliku, którego nazwa jest drugim argumentem wywołania skryptu.

Wejście/wywołanie programu:

```
./zad6.awk dane_6 wynik_6
```

Wyjście/wynik:

```
Nazwisko        Imię            Suma    Średnia
Dobrowolska     Maria            28     3.50
Jezierska       Karolina         27     4.50
Nawrocki        Janusz           16     3.20
Nowacki         Dariusz          13     4.33
Nowak           Piotr             0     0.00
Ziemowit        Sebastian        37     4.11
```

*Treści oznaczone kursywą pochodzą z różnych źródeł internetowych.