# JĘZYKI SKRYPTOWE

UNIWERSYTET IM. ADAMA MICKIEWICZA W POZNANIU

**Lech S. Borkowski**

# JĘZYKI SKRYPTOWE


POZNAŃ 2006

Recenzent: prof. dr hab. inż. Zbyszko Królikowski



Projekt okładki: Lech S. Borkowski
Redakcja: Aleksandra Jędrzejczak
Redakcja techniczna: Dorota Borowiak

ISBN 83-232-1582-0

WYDAWNICTWO NAUKOWE UNIWERSYTETU IM. ADAMA MICKIEWICZA W POZNANIU
61-734 Poznań, ul. F. Nowowiejskiego 55, tel. +48 61 829 39 85, fax +48 61 829 39 80
e-mail: press@amu.edu.pl http://amu.edu.pl/∼press

Wydanie I. Nakład 300 egz. Ark. wyd. 19,75. Ark. druk. 20,00

Druk i oprawa: ESUS DRUK CYFROWY, POZNAŃ, UL. WIERZBIĘCICE 35

# Spis treści

# Wstęp

Książka przedstawia program wykładu *Języki skryptowe*, prowadzonego na trzecim roku kierunku *Informatyka stosowana* na Wydziale Fizyki Uniwersytetu im. Adama Mickiewicza w Poznaniu. Adresatem tej treści jest osoba zaznajomiona przynajmniej w podstawowym stopniu z językiem C, systemem Unix i podstawami informatyki.

Celem książki jest zachęcenie studenta do programowania w językach skryptowych, stąd obecność w niej tak wielu przykładów. Autor uważa, że student najwięcej zyska, jeśli zastosuje się do zasady *learning by doing*.

Rozdziały od 1. do 5. zawierają podstawowy zasób dokumentacji, wiedzy i przykładów dotyczących najważniejszych narzędzi skryptowych systemu Unix. Obecnie są one dostępne także w wielu innych systemach operacyjnych.

Rozdziały dotyczące języków Tcl i Python mają na celu przedstawienie podstaw składni obu języków.

W rozdziale o języku Python przedstawiono także bardziej rozbudowane skrypty z wykorzystaniem interfejsu graficznego (moduł Tkinter).

W rozdziale 8. umieszczono zbiór około 120 zadań, które są rozwiązane za pomocą narzędzi z rozdziałów 1-5.

# 1. Wyrażenia regularne

Nazwa „wyrażenia regularne” (ang. *regular expressions*) oznacza notację stosowaną w celu wyszukiwania i dopasowywania ciągów znaków. Są dwie grupy wyrażeń regularnych: podstawowa i rozszerzona. Ich składnia w konkretnych narzędziach nieco się różni, ale nie na tyle, by utrudnić omówienie podstawowych cech.

W skład wyrażeń regularnych wchodzą zwykłe znaki (litery, cyfry) i znaki specjalne (metaznaki). Oto metaznaki używane w wyrażeniach regularnych:

```
\   ^   .   [   ]   |   (   )   *   +   ?
```

Są trzy rodzaje wyrażeń regularnych:

- Metaznaki pozycyjne
- Zbiory znaków
- Modyfikatory oznaczające liczbę powtórzeń ostatniego wyrażenia

Do grupy podstawowych wyrażeń regularnych należą:

- Zwykły znak (niebędący metaznakiem), np. `a`, który pasuje do samego siebie
- Sekwencja oznaczająca znak specjalny, np. `\t` oznacza znak tabulacji
- Cytowany metaznak, np. `\*`, pozbawiony w ten sposób specjalnego znaczenia
- Znak `^` oznaczający początek łańcucha znaków
- Znak `$` oznaczający koniec łańcucha znaków
- Kropka `.` oznaczająca dowolny pojedynczy znak
- Klasa znaków, np. `[ABC]` odpowiada dowolnemu spośród znaków `A`, `B` lub `C`. Klasy znaków mogą zawierać skróty, np. `[A-Za-z]` odpowiada dowolnej pojedynczej literze
- Zaprzeczenie klasy znaków, np. `[^0-9]` odpowiada dowolnemu znakowi, oprócz cyfry

Nawias kwadratowy określa klasę znaków. Oznacza on dowolny pojedynczy znak spośród znaków zawartych w nawiasach. Na przykład, `[abcd]` odpowiada `a`, `b`, `c` lub `d`.

W przypadku kilku następujących po sobie znaków, można użyć notacji skróconej. Na przykład, zapis `[a-d]` oznacza małą literę a, b, c lub d. Zapis `[A-Ca-c][0-9]` oznacza jedną literę, wielką lub małą, od a do c, po której następuje dowolna cyfra.

Wyrażenie, w którym pierwszym znakiem po nawiasie kwadratowym `[` jest znak `^`, oznacza domknięcie klasy znaków. Do domknięcia należą wszystkie znaki, które nie są zawarte w grupie następującej po znaku `^`. Oto kilka przykładów wyrażeń regularnych:

| Wyrażenie | Znaczenie |
|---|---|
| `^[ABC]` | Znak `A`, `B` lub `C` znajdujący się na początku ciągu znaków |
| `^[^ABC]` | Dowolny znak na początku ciągu znaków, różny od `A`, `B` i `C` |
| `[^ABC]` | Dowolny znak różny od `A`, `B` i `C` |
| `A$` | Znak `A` znajdujący się na końcu ciągu znaków |
| `^A$` | Ciąg znaków, którego jedynym elementem jest `A` |
| `^[^a-z]$` | Ciąg znaków składający się dokładnie z jednego znaku innego niż mała litera |
| `^..$` | Dowolny ciąg składający się dokładnie z dwóch znaków |
| `\.$` | Ciąg, w którym kropka jest ostatnim elementem |

Wszystkie znaki występujące wewnątrz klasy znaków reprezentują same siebie, oprócz znaku `\`, znaku `^` na początku (tj. w zestawieniu `[\`) i znaku `-` znajdującego się pomiędzy dwoma znakami. Na przykład, `[.]` odpowiada kropce, a notacja `^[^^]` oznacza dowolny znak różny od `^`, znajdujący się na początku łańcucha znaków.

W celu łączenia wyrażeń regularnych wykorzystywane są następujące operatory:

| Operatory | Znaczenie |
|---|---|
| `A\|B` | Alternatywa, oznacza `A` lub `B` |
| `AB` | Łączność, oznacza `A`, po którym następuje `B` |
| `A*` | Dopełnienie; odpowiada zerowej lub większej od zera liczbie znaków `A` |
| `A+` | Dopełnienie dodatnie; `A+` odpowiada jednemu znakowi `A` lub większej ich liczbie |
| `A?` | Zero lub jeden znak; odpowiada pustemu ciągowi znaków lub jednemu znakowi `A` |
| nawiasy | Zapis `(w)` odpowiada wyrażeniu regularnemu `w` |

Alternatywa `w1|w2` odpowiada dowolnemu łańcuchowi określonemu wyrażeniem `w1` lub `w2`.

Nawiasy `()` służą grupowaniu części składowych. Jeżeli `w1` i `w2` są wyrażeniami regularnymi, to `(w1)(w2)` (uwaga: między `(w1)` i `(w2)` nie ma spacji) odpowiada ciągowi postaci `xy`, gdzie `w1` odpowiada `x`, a `w2` odpowiada `y`. Nawiasy wokół `w1` i `w2` można pominąć, jeśli w wyrażeniach regularnych, które się w nich znajdują, nie ma operatora alternatywy.

Symbole `*`, `+` i `?` są operatorami stosowanymi do zapisu powtórzeń w wyrażeniach regularnych.

Kolejność operatorów według zasady pierwszeństwa (od najniższego priorytetu) jest następująca: alternatywa `|`, łączenie, operatory powtórzeń `*`, `+` i `?`. Podobnie jak w wyrażeniach arytmetycznych, najpierw uwzględniane są operatory o wyższym priorytecie. Wynika stąd, że często można pominąć nawiasy. Wyrażenie `ab|cd` jest równoważne wyrażeniu `(ab)|(cd)`, a `^ab|cd*e$` znaczy to samo, co `(^ab)|(c(d*))e$`.

**Przykłady**

| Wyrażenie | Znaczenie |
|---|---|
| `AB*C` | `AC`, `ABC` lub `ABBC` itd. (zero, jedno lub wiele powtórzeń znaku `B`) |
| `AB+C` | `ABC`, `ABBC` lub `ABBBC` itd. (jedno lub wiele powtórzeń znaku `B`) |
| `AB?C` | `AC` lub `ABC` |
| `[A-Z]+` | Łańcuch wielkich liter, składający się z co najmniej jednego znaku |
| `[A-Z][A-Z]*` | jw. |
| `[A-Z]*` | Brak wielkich liter lub dowolna ich liczba (czyli dowolny ciąg znaków) |
| `(AB)+C` | `ABC`, `ABABC` lub `ABABABC` itd. |

Więcej informacji na temat wyrażeń regularnych i klas znaków zdefiniowanych w standardzie POSIX, znajduje się w podrozdziale 4.9. i 7.11.

# 2. Filtry grep

Filtry grep (grep, egrep i fgrep) wywodzą się z edytora ed, w którym istnieje polecenie `g/re/p`, gdzie `re` oznacza wyrażenie regularne. Nazwa jest skrótem słów *Global Regular Expression Print*. Oto ogólna postać poleceń grep:

```
$ grep [opcje] wzorzec [pliki]
$ grep [opcje] [-e wzorzec | -f plik] [pliki]
```

grep wyszukuje w plikach (lub w tekście przekazanym na standardowym wejściu) wiersze zawierające `wzorzec`. Domyślną czynnością jest drukowanie znalezionych wierszy. Polecenie `egrep` jest równoważne `grep -E`, a `fgrep` odpowiada `grep -F`. Oto opcje tego programu i ich znaczenie.

`-A` *n* lub `-after-context=`*n*
Drukowanie *n* kolejnych wierszy po znalezieniu wiersza pasującego do wzorca; grupy wierszy oddzielone są znakami `--`

`-a` lub `--text`
Przetwarzanie pliku binarnego tak, jak pliku tekstowego; opcja równoważna `--binary-files=typ`

`-B n lub --before-context=`*n*
Drukowanie *n* wierszy poprzedzających dopasowane wiersze; grupy wierszy oddzielone są znakami `--`

`-C n` lub `--context=`*n*
Drukowanie *n* wierszy przed dopasowanym wierszem i po nim; grupy wierszy oddzielone są znakami `--`

`-b` lub `--byte-offset`
Drukowanie liczby bajtów poprzedzających wiersz pasujący do wzorca, licząc od początku pliku

`--binary-files=`*typ*
Jeżeli początkowe bajty pliku wskazują na to, że plik jest plikiem binarnym, przetwarzany jest jako plik typu *typ*. Domyślnie *typ* ma wartość `binary`. W przypadku wykonywania polecenia `grep` na pliku binarnym drukowany jest krótki komunikat `binary file matches`, jeśli wzorzec pasuje do jakiegoś ciągu znaków w pliku. W przeciwnym razie nie ma żadnego komunikatu. Jeżeli typ ma wartość `without-match`, grep przyjmuje, że w pliku binarnym nie ma żadnych ciągów pasujących do wzorca, co jest równoważne opcji `-I`. Jeżeli typ ma wartość `text`, plik binarny przetwarzany jest tak, jak plik tekstowy, co jest równoważne opcji `-a`. Uwaga: użycie opcji `-binary-files=text` może w niektórych przypadkach dawać nieprzewidziane konsekwencje, jeśli wyjście polecenia jest kierowane na terminal, ponieważ program zarządzający pracą terminala może interpretować niektóre z otrzymanych ciągów jako polecenia

`--colour[=kiedy]` lub `-color[=kiedy]`
Wyróżnienie dopasowanego ciągu w sposób określony wartością zmiennej środowiskowej `GREP_COLOR`; kiedy przyjmuje wartości `never`, `always` lub `auto`

`-c` lub `--count`
Drukowanie liczby dopasowanych wierszy w każdym z plików wejściowych. Dopasowane wiersze nie są drukowane

`-D` *czynność*
Jeżeli plikiem wejściowym jest urządzenie, kolejka FIFO lub gniazdo, strumień danych jest przetwarzany zgodnie z dyrektywą *c*zynność. Domyślnie *c*zynność ma wartość `read`, co oznacza, że dane otrzymywane z urządzeń są przetwarzane tak, jakby pochodziły z pliku. Jeżeli czynność ma wartość skip, dane pochodzące z urządzenia nie są przetwarzane

`-d` *czynność* lub `--directories=`*czynność*
Jeżeli plikiem wejściowym jest katalog, przetwarzanie odbywa się zgodnie z dyrektywą *czynność*. Domyślnie *czynność* ma wartość `read`, co oznacza, że katalogi są odczytywane tak, jak zwykłe pliki. Jeżeli wartością jest `skip`, katalogi są pomijane bez wyświetlania żadnego komunikatu. Jeżeli czynność ma wartość `recurse`, rekursywnie odczytywane są wszystkie pliki ze wszystkich podkatalogów danego katalogu; jest to równoważne opcji `-r`

`-E` lub `--extended-regexp`
Interpretowanie wzorca jako rozszerzonego wyrażenia regularnego

`-e wzorzec` lub `--regexp=`*wzorzec*
Ten sposób deklaracji wzorca umożliwia przetwarzanie wzorców rozpoczynających się znakiem `-`

`-F` lub `--fixed-strings`
Traktowanie wzorca jako listy ciągów o stałej długości znaków oddzielonych od siebie znakiem nowego wiersza. Dopasowywany jest każdy z tych ciągów. Użycie dodatkowej opcji `-P` lub `--perl-regexp` powoduje, że wzorzec jest traktowany jako wyrażenie regularne języka Perl

`-f` *plik* lub `--file=`*plik*
Wzorce pobierane są z pliku, po jednym z każdego wiersza. Plik pusty zawiera zero wzorców i w związku z tym nie pasuje do żadnego tekstu

`-G` lub `--basic-regexp`
Wzorzec jest interpretowany jako podstawowe wyrażenie regularne. W podstawowych wyrażeniach regularnych metaznaki `? + { | ( )` tracą specjalne znaczenie. Aby je przywrócić, należy użyć sekwencji `\? \+ \{ \| \(` i `\)`

`-H` lub `--with-filename`
Drukuje nazwę pliku dla każdego dopasowanego wiersza

`-h` lub `--no-filename`
W przypadku przeszukiwania wielu plików nie drukuje nazwy pliku po znalezieniu pasującego wiersza

`--help`
Drukowanie krótkiego objaśnienia

`-I`
Przetwarzanie pliku binarnego tak, jak gdyby nie zawierał ciągów pasujących do wzorca; równoważne opcji `--binary-files=without-match`

`-i` lub `--ignore-case`
Ignorowanie wielkości liter we wzorcu i w przetwarzanych plikach

`-L` lub `--files-without-match`
Dopasowane wiersze nie są drukowane. Drukowane są nazwy plików, w których nie znaleziono żadnych pasujących wierszy. Przetwarzanie pliku jest zakończone po znalezieniu pierwszego dopasowania

`-l` lub `--files-with-matches`
Dopasowane wiersze nie są drukowane. Drukowane są nazwy plików, w których znaleziono pasujący wiersz. Przetwarzanie pliku jest zakończone po znalezieniu pierwszego dopasowania

`-m` *n* lub `--max-count=`*n*
Zakończenie wczytywania pliku po znalezieniu *n* pasujących wierszy. W przypadku użycia opcji `-v` lub `--invert-match` przetwarzanie jest zakończone po wydrukowaniu *n* wierszy niezawierających wzorca

`--mmap`
Jeśli to możliwe, wczytywanie danych odbywa się za pomocą funkcji systemowej `mmap(2)` zamiast domyślnej funkcji `read(2)`. W niektórych sytuacjach daje to większą wydajność. Związane jest z tym jednak pewne ryzyko (mogą być wykonane zrzuty stanu pamięci), gdy plik wejściowy się zmniejszy w czasie działania programu grep lub pojawi się błąd operacji I/O

`-n` lub `--line-number`
Na początku każdego wiersza drukowany jest jego numer w pliku wejściowym

`-o` lub `--only-matching`
Drukowanie tylko ciągów znaków pasujących do wzorca, a nie całych wierszy

`--label=`*nazwa*
Wyświetlanie danych pochodzących ze standardowego wejścia w taki sposób, jakby pochodziły z pliku *nazwa*. Jest to przydatne w przypadku korzystania z narzędzi w rodzaju zgrep, na przykład `gzip -cd nazwa.gz | grep --label=nazwa wzorzec`

`--line-buffering`
Buforowanie wierszy. Może skutkować obniżeniem wydajności

`-q` lub `--quiet` lub `--silent`
Na wyjściu nie są drukowane żadne wyniki. Po znalezieniu pasujących wierszy przetwarzanie jest zakończone z kodem zero, niezależnie od pojawienia się błędów. Zobacz opcja `-s` lub `--no-messages`

`-R` lub `-r` lub `--recursive`
Rekursywne przetwarzanie wszystkich plików z każdego katalogu i podkatalogów; równoważne opcji `-d`

`--include=`*wzorzec*
Rekursywne przetwarzanie katalogów z uwzględnieniem nazw plików pasujących do wzorca

`--exclude=`*wzorzec*
Rekursywne przetwarzanie katalogów z pominięciem nazw plików pasujących do wzorca

`-s` lub `--no-messages`
Pominięcie komunikatów o nieistniejących plikach lub problemach z odczytaniem plików

`-U` lub `-binary`
Pliki są traktowane jako pliki binarne

`-u` lub `--unix-byte-offsets`
Drukowanie liczby bajtów poprzedzających wiersz pasujący do wzorca, licząc od początku pliku; zakłada się format plików tekstowych Uniksa – znaki `CR` są pominięte

`-V` lub `--version`
Drukowanie numeru wersji programu grep na standardowym wyjściu błędów

`-v` lub `--invert-match`
Negacja dopasowania; drukowane są jedynie wiersze niepasujące do wzorca

`-w` lub `--word-regexp`
Drukowane są jedynie wiersze, w których dopasowane wyrażenia tworzą pełne słowa. Pasujący ciąg znaków musi znajdować się na początku wiersza lub być poprzedzony znakiem nietworzącym słów. Podobnie za słowem musi znajdować się koniec wiersza lub znak nietworzący słów. Do znaków tworzących słowa zaliczają się litery, cyfry i podkreślenia `_`

`-x` lub `--line-regexp`
Drukowanie jedynie wierszy w całości zgodnych ze wzorcem (oprócz ciągu znaków zgodnego ze wzorcem nie zawierają żadnych innych znaków)

`-y`
Synonim opcji `-i`

`-Z` lub `--null`
Drukowanie znaku `NUL` (kod ASCII 000) zamiast znaku, który zwykle pojawia się po nazwie pliku. Na przykład, `grep -lZ` drukuje `NUL` po każdej nazwie pliku zamiast znaku nowego wiersza. Ułatwia to przetwarzanie nazw plików zawierających nietypowe znaki, na przykład znak nowego wiersza.

# Przykłady

Oto przykłady stosowania filtrów grep:

1. `fgrep abc plik1`
   Drukowanie wierszy pliku plik1 zawierających ciąg `abc`

2. `fgrep -f plik1 plik2`
   Drukowanie wierszy z pliku `plik1` zawierających wyrażenia zapisane w `plik2`

3. `grep abc plik` Drukowanie wierszy zawierających ciąg `abc`

4. `grep '^abc' plik`
   Drukowanie wierszy rozpoczynających się ciągiem `abc`

5. `grep '^a' plik`
   Drukowanie wierszy rozpoczynających się literą `a`

6. `grep '^[a-z]' plik`
   Drukowanie wierszy rozpoczynających się małą literą

7. `grep '^.[ab]' plik`
   Drukowanie wierszy, w których drugą literą jest `a` lub `b`

8. `grep '\.$' plik`
   Drukowanie wierszy, w których ostatnim znakiem jest `.` (kropka)

9. `grep ';$' plik`
   Drukowanie wierszy kończących się znakiem `;` (średnik)

10. `grep -c ';'` plik
    Liczenie wierszy kodu programu w języku C

11. `grep '\.\.*' plik`
    Drukowanie wierszy kończących się co najmniej jedną kropką

12. `egrep ';+' plik`
    Drukowanie wierszy zawierających co najmniej jeden średnik

13. `egrep 'a(_)?1' plik`
    Drukowanie wierszy zawierających ciąg `a1` lub `a_1`

14. `egrep '(a1)|(a_1)' plik`
    Drukowanie wierszy zawierających `a1` lub `a_1`

15. `egrep -f plik1 plik2`
    Drukowanie tych wierszy pliku `plik1`, które zawierają wyrażenia zapisane w `plik2`

16. `egrep '([0-9][ -]?){7}' plik`
    Drukowanie wierszy zawierających ciąg 7 cyfr, bezpośrednio następujących po sobie lub rozdzielonych pojedynczą spacją lub znakiem `-` (np. do wyszukiwania lokalnych numerów telefonicznych)

Na zakończenie rozdziału przykład z literatury. Sprawdzimy częstotliwość wzmiankowania poszczególnych postaci w *Panu Tadeuszu* Adama Mickiewicza. W tym celu wystarczy na pliku tekstowym zawierającym utwór wykonać polecenie `grep -c 'postac' tekst_utworu`. Na przykład,

```
$ grep -c 'Tad' pantadeusz.txt
183

$ grep -c 'Hrabi' pantadeusz.txt
205
```

# 3. Strumieniowy edytor tekstu sed

Nazwa sed jest skrótem słów Stream EDitor. Narzędzie to powstało w 1973 lub 1974 roku. Jego autorem jest Lee E. McMahon. Edytor sed działa w trybie nieinterakcyjnym (wsadowym). Przetwarza kolejne wiersze wejściowego strumienia tekstu, dokonuje w nich zmian zapisanych w instrukcjach i przekazuje wynik do standardowego wyjścia. Wykorzystywane są dwa bufory danych: przestrzeń wzorca (ang. *pattern space*) i przestrzeń tymczasowa (ang. *hold space*). Pojedynczy wiersz po wczytaniu przez sed trafia do przestrzeni wzorca, w której wykonywane są manipulacje tekstu. Przestrzeń tymczasowa jest początkowo pusta, ale za pomocą niektórych poleceń można przekazywać do niej dane z przestrzeni wzorca i w odwrotnym kierunku.

Oto ogólna postać polecenia sed:

```
sed [opcje] polecenie_edycji pliki
```

Polecenie edycji ma następującą postać:

```
[adres1[,adres2]][funkcja][argumenty]
```

Adresy nie są obowiązkowe i mogą być oddzielone od funkcji spacją lub znakiem tabulacji.

Rolę adresów wskazujących konkretne wiersze tekstu mogą pełnić liczby dziesiętne. Pierwszy przetwarzany wiersz jest oznaczony numerem 1. W przypadku przetwarzania wielu plików tekst wszystkich plików jest traktowany jak jedna całość i adresy wierszy są stale zwiększane. Ostatni przetwarzany wiersz jest oznaczany znakiem `$`. Oprócz adresów liczbowych używane są też adresy kontekstowe, którymi są wzorce z wyrażeniami regularnymi ograniczone ukośnikami /.

Polecenia edycji mogą zawierać 0, 1 lub 2 adresy oddzielone przecinkami. Zakres działania poleceń edycji jest wówczas następujący:

| Liczba adresów | Zakres działania |
|---|---|
| 0 | Wszystkie wiersze tekstu |
| 1 | Wiersz o wskazanym adresie |
| 2 | Wszystkie wiersze w zakresie od wiersza wskazanego przez pierwszy adres do wiersza odpowiadającego drugiemu adresowi. Czynności edycji są następnie powtarzane w następnym zakresie wierszy, do którego pasują podane adresy. |

Polecenia powinny być umieszczane w pojedynczym cudzysłowie fl, jeśli określone są dodatkowe opcje lub funkcje. Wzorce pełnią tę samą rolę, co adresy kontekstowe. Różnią się od adresów kontekstowych tym, że ich ogranicznikami mogą być nie tylko spacja i znak otwarcia nowego wiersza, ale także wszystkie zwykłe znaki tekstowe. Wstawiany łańcuch nie jest wyrażeniem regularnym. Metaznak poprzedzony ukośnikiem  jest pozbawiony specjalnego znaczenia.

W następującej tabeli są umieszczone najczęściej używane opcje edytora sed.

| Opcja | Znaczenie |
|---|---|
| `-e` | Traktuj następny argument jako instrukcję edytora |
| `-n` | Nie drukuj domyślnego wyjścia, a jedynie wiersze wybrane instrukcjami `p` lub `s///p` |
| `-f skrypt` | Pobierz skrypty edycji z pliku `skrypt` |

Do najczęściej używanych instrukcji sed należy instrukcja podstawiania:

```
s/wzorzec/nowy_lancuch/znacznik
```

W miejsce wszystkich ciągów pasujących do wzorca podstawiany jest `nowy_lancuch`. Wzorzec jest wyrażeniem regularnym. Przetworzony tekst pojawia się na standardowym wyjściu, czyli na ekranie. Wyjście można skierować do innego pliku:

```
$ sed [opcje] 'polecenia' plik > plik1
```

Polecenia edycji mogą być zapisane w pliku. Wywołuje się je za pomocą opcji `-f`:

```
$ sed -f plik_polecen plik > plik1
```

Komentarze w skryptach sed są oznaczane znakiem `#`. Wiersz jest traktowany jako komentarz, jeżeli `#` jest pierwszym znakiem wiersza.

W łańcuchu, który jest wstawiany w miejsce wyrażenia regularnego (wzorca) jedynie następujące znaki mają znaczenie specjalne:

| Znak | Znaczenie |
|---|---|
| `&` | W to miejsce wstawiany jest łańcuch pasujący do wyrażenia regularnego |
| `\n` | Pasuje do $n$-tego podciągu zdefiniowanego we wzorcu za pomocą znaków `( i )` |
| `\` | Używany do zmiany specjalnego znaczenia znaków `&`, `\` i ogranicznika w poleceniu podstawienia tekstu |

Oto polecenia edytora sed:

`#`
Początek komentarza; musi być pierwszym znakiem wiersza

`:etykieta`
Etykieta dla poleceń `b` i `t`

`=`
Drukowanie numeru bieżącego wiersza

`a\tekst`
Dołączenie ciągu `tekst`. Jeżeli dołączany tekst składa się z wielu wierszy, znaki nowego wiersza muszą być poprzedzone znakiem `\`

`i\tekst`
Wstawianie tekstu. Jeżeli dołączany tekst składa się z wielu wierszy, znaki nowego wiersza muszą być poprzedzone znakiem `\`

`r plik`
Wczytanie i dołączenie tekstu z `pliku`

`b etykieta`
Przetwarzanie bieżącego wiersza ma być kontynuowane w poleceniu znajdującym się po `etykiecie`. Jeżeli brak etykiety, przejdź na koniec skryptu

`c\ tekst`
W miejsce wierszy określonych adresem wstawiany jest `tekst`. Jeżeli dołączany tekst składa się z wielu wierszy, znaki nowego wiersza muszą być poprzedzone znakiem `\`

`d`
Usunięcie wierszy określonych adresem

`D`
Usunięcie pierwszego wiersza z przestrzeni wzorca, od początku do zamaskowanego znaku nowego wiersza. Przetwarzanie jest następnie kontynuowane od początku skryptu. Jeżeli w przestrzeni wzorca nie ma więcej wierszy, wczytywany jest nowy wiersz wejściowy

`g`
Skopiowanie zawartości przestrzeni tymczasowej do przestrzeni wzorca

`G`
Dołączenie zawartości przestrzeni tymczasowej do przestrzeni wzorca; dołączony tekst jest oddzielony znakiem nowego wiersza

`h`
Skopiowanie zawartości przestrzeni wzorca do przestrzeni tymczasowej

`H`
Dołączenie zawartości przestrzeni wzorca do przestrzeni tymczasowej; dołączony tekst jest oddzielony znakiem nowego wiersza (również wtedy, kiedy przestrzeń tymczasowa jest pusta)

`i\tekst`
Wstawienie tekstu przed każdym wierszem zgodnym z adresem

`l`
Wypisanie zawartości przestrzeni wzorca; w miejsce znaków niedrukowalnych wyświetlane są kody ASCII

`n`
Wyświetlenie przestrzeni wzorca, usunięcie bieżącej zawartości i wczytanie następnego wiersza do przestrzeni wzorca; następną wykonywaną instrukcją jest kolejne po n polecenie skryptu

`N`
Dołączenie następnego wiersza wejściowego do przestrzeni wzorca; dołączony tekst jest oddzielony znakiem nowego wiersza

`p`
Wyświetlenie zawartości przestrzeni wzorca

`P`
Wyświetlenie pierwszego wiersza przestrzeni wzorca

`q`
Zakończenie działania po napotkaniu wiersza o określonym adresie

`r plik`
Wczytanie pliku do przestrzeni wzorca

`s`
`s/wzorzec/nowylancuch/[znaczniki]`
Podstawienie w miejsce wzorca nowego tekstu. Można użyć następujących znaczników:

| | |
|---|---|
| `g` | Wykonanie podstawienia dla wszystkich wystąpień wzorca w wierszu |
| `n` | Liczba od 1 do 512; podstawienie jest wykonywane tylko dla *n*-tego wystąpienia wzorca w wierszu |
| `p` | Jeżeli podstawienie zostało wykonane, wiersz jest drukowany na standardowym wyjściu |
| `w plik` | Jeżeli podstawienie zostało wykonane, wiersz jest wpisywany do `pliku` |

`t`
Jeżeli polecenie `s///` zostało wykonane, przejdź do polecenia oznaczonego etykietą. W przypadku braku etykiety przejdź na koniec skryptu

`w plik`
Zapisanie zawartości przestrzeni wzorca w `pliku`

`x`
Zamiana zawartości przestrzeni wzorca i przestrzeni tymczasowej

`y/znaki1/znaki2/`
Translacja znaków z przestrzeni wzorca; na przykład `y/abc/123/` oznacza zmianę wszystkich wystąpień `a` na `1`, `b` na `2`, `c` na `3`

## Przykłady

1. `sed 's/,/,\ /g' plik`
   W miejsce każdego przecinka wstawia przecinek i spację

2. `sed 's/abc/def/' plik`
   Zamienia ciąg `abc` na `def`. Zamiana dotyczy jedynie pierwszego ciągu `abc` w wierszu. Jeżeli w tym samym wierszu `abc` pojawia się więcej razy, kolejne wystąpienia `abc` nie będą zamieniane

3. `sed '1,4s/abc/def/' plik`
   Zamienia `abc` na `def` w wierszach od 1 do 4. Zamiana dotyczy jedynie pierwszego ciągu `abc` w wierszu

4. `sed 's/abc/def/g' plik1 plik2`
   Zamienia `abc` na `def` w plikach `plik1` i `plik2`. Znacznik `g` oznacza zamianę wszystkich wystąpień `abc` w całym wierszu

5. `sed 's/abc/def/gw zmiany' plik`
   Zamienia `abc` na `def`. Zmienione wiersze są dopisywane do pliku `zmiany`

6. `sed 's/abc/def/g' plik1 >> plik2`
   Zamienia `abc` na `def` w całym pliku. Dopisuje zmiany do `plik2`

7. `sed 's/abc/def/3' plik`
   Zamienia trzeci ciąg `abc` w każdym wierszu

8. `sed '10!s/pecet/notebook/' plik`
   Zamienia `pecet` na `notebook` wszędzie, oprócz 10 wiersza

9. `sed 'd' plik`
   Usuwa wszystkie wiersze

10. `sed '1d' plik`
    Usuwa pierwszy wiersz

11. `sed '$d' plik`
    Usuwa ostatni wiersz

12. `sed '/^$/d' plik`
    Usuwa puste wiersze

13. `sed '12,28d' plik1`
    Usuwa z pliku `plik1` wiersze od 12 do 28

14. `sed '10,20d' plik1 >> plik2`
    Usuwa z pliku `plik1` wiersze od 10 do 20, a zmienione wiersze są dopisywane do pliku `plik2`

15. `sed '/abc/,/def/d' plik`
    Usuwa wszystkie wiersze od pierwszego, w którym pojawia się ciąg `abc`, do najbliższego, w którym pojawia się `def`

16. `sed '/\{/,/\}/d' plik`
    Usuwa wiersze od pierwszego, w którym pojawia się znak `{`, do najbliższego, w którym pojawia się `}`

17. `sed '1,/abc/d' plik`
    Usuwa wiersze od pierwszego do najbliższego, zawierającego ciąg `abc`

18. `sed '1,/^$/d' plik`
    Usuwa wiersze od początku pliku do pierwszego pustego wiersza; przydatne do usuwania nagłówków poczty

19. `sed '1,/^$/!d' plik`
    Zaprzeczenie poprzedniego polecenia; pozostawia jedynie wiersze od początku do pierwszego pustego, a pozostałe usuwa

20. `sed '100,$d' plik`
    Usuwa wiersze od 100 do końca pliku

21. `sed '/\./,$d' plik`
    Usuwa wiersze od pierwszego wystąpienia znaku `.` (kropka) do ostatniego wiersza

22. `sed 's/ *$//d' plik`
    Usuwa wszystkie spacje na końcu każdego wiersza

23. `sed 's/^[[:blank:]]*//' plik`
    Usuwa wszystkie spacje i znaki tabulacji na początku każdego wiersza

24. `sed 's/[[:blank:]]*$//' plik`
    Usuwa wszystkie spacje i znaki tabulacji na końcu każdego wiersza

25. `sed 's/^[[:blank:]]*//; s/[[:blank:]]*$//' plik`
    Usuwa wszystkie spacje i znaki tabulacji na początku i na końcu każdego wiersza

26. `sed -n '/^Date:/,/^Name:/p' plik`
    Po napotkaniu ciągu `Date:` na początku wiersza drukuje ten wiersz i wszystkie kolejne, aż do pojawienia się na początku wiersza ciągu `Name:`

27. `sed -n 1,10p plik`
    Drukuje tylko pierwszych 10 wierszy tekstu

28. `sed G plik > plik1`
    Między wierszami tekstu umieszcza dodatkowy pusty wiersz

29. `sed 'G;G' plik > plik1`
    Między wierszami tekstu umieszcza dwa puste wiersze

30. `sed -n '/[0-9]\{3\}/p' plik`
    Drukuje wiersze zawierające ciąg 3 cyfr

31. `sed -e :etykieta -e 'N; s/\n/ /g; tetykieta' plik`
    Umieszcza całą zawartość pliku w jednym wierszu

32. Chcemy umieścić w pliku html odsyłacz do wyszukiwarki Google. Odsyłacz ma być ostatnim elementem tej strony www.

    Tworzymy jednowierszowy skrypt `wstaw.sed`:

    ```
    s/<\/body>/\
    <a href="http:\/\/www.google.com\/">Google<\/a>/
    ```

    Uruchamiamy go poleceniem `sed -f wstaw.sed plik.html`

33. Pewien program rejestrujący połączenia użytkowników z serwerem zapisuje dane w następującej postaci:

    ```
    ...
    9:10:16   uzytkownik1   grupa   komputer1  (192.168.10.7)
    9:11:25   uzytkownik2   grupa   komputer2  (192.168.10.12)
    9:13:07   uzytkownik3   grupa   komputer3  (192.168.10.150)
    ...
    10:03:48  uzytkownikx   grupa   komputerx  (142.213.10.82)
    ...
    ```

    Ostatnie pole jest adresem IP komputera, z którego nastąpiło połączenie. Jak wydrukować adresy dla połączeń między godziną 9 a 10? Załóżmy, że dane są zapisane w `pliku`. Odpowiedź:

```
sed -e '/^[^9]/d' -e 's/^.*(//' -e 's/).*$//' plik
```

34. Jak wydobyć adresy stron internetowych z dokumentu pochodzącego z witryny internetowej, np.
`http://directory.google.com/Top/Computers/Programming/Languages/PHP/`?

    Kopię strony zapisać w pliku.

    Najpierw rozwiązanie:

```
sed -n 's/http/\
http/gp' plik.html |\
sed  -e '/^http/!d'  -e 's/".*$//'
     -e 's/<.*//' -e 's/>.*//'
     -e 's/ .*$//' -e '/^$/d'
```

    Pierwsze polecenie otwiera nowy wiersz w miejscu pojawienia się adresu strony internetowej. Następnie wynik przekazywany jest do następnego ciągu poleceń, które wykonują kolejno następujące czynności:

    * Usunięcie wierszy nierozpoczynających się ciągiem `http`

    * Usunięcie ciągu znaków od cudzysłowu `"` do końca wiersza

    * Usunięcie ciągu znaków od znaku `<` do końca wiersza

    * Usunięcie ciągu znaków od znaku `>` do końca wiersza

    * Usunięcie ciągu znaków od spacji do końca wiersza

    * Usunięcie pustych wierszy

# 4. Język awk

## 4.1. Wstęp

Awk jest językiem programowania do szeroko rozumianego przetwarzania tekstu i danych. Nazwa `awk` pochodzi od pierwszych liter nazwisk jego autorów: Alfreda V. Aho, Briana W. Kernighana i Petera J. Weinbergera. Pierwsza wersja tego narzędzia powstała w 1977 r. w AT&T Bell Laboratories. Składnia awk jest częściowo podobna do języka C. Z czasem pierwotna wersja awk została zastąpiona przez nawk (new awk). W ramach GNU Project powstała rozszerzona wersja interpretera pod nazwą gawk. GNU awk jest zgodny z definicją języka w standardzie POSIX 1003.2. Zawiera też wszystkie elementy składni nawk. Materiał tego rozdziału dotyczy wersji gawk.

Oto najprostsze uruchomienie interpretera awk w wierszu poleceń:

```
$ awk 'program' pliki
```

gdzie *program* oznacza wyrażenie postaci

```
wzorzec { instrukcje }
```

*Wzorzec* jest wyrażeniem określającym wiersze, na których należy wykonać *instrukcje*. Jest kilka rodzajów takich wyrażeń (patrz niżej). Jeśli wzorzec jest pominięty, przetwarzane są wszystkie wiersze:

```
$ awk '{ instrukcje }' pliki
```

Na przykład, polecenie

```
$ awk '{ print }' pliki
```

drukuje na wyjściu całą zawartość plików. Wykonuje zatem to samo zadanie, co polecenie `cat`, tyle że wolniej.

**Uwaga**. W niektórych wersjach awk konieczne są spacje między nawiasami klamrowymi, a zawartymi w nich instrukcjami.

Jeżeli nie są zdefiniowane żadne czynności, drukowane są wszystkie wiersze pasujące do danego wzorca:

```
$ awk 'wzorzec' pliki
```

Awk nie zmienia postaci plików wejściowych. Wiersze plików wejściowych są przetwarzane jak nieprzerwany strumień danych. Polecenia awk można wczytywać z pliku:

```
$ awk -f plik_polecen pliki
```

Każdy wiersz przetwarzany przez awk jest automatycznie dzielony na pola (ang. *fields*), nieprzerwane ciągi znaków, oddzielone spacjami lub tabulatorami. Jest to domyślne znaczenie spacji i tabulatorów (znaki rozdzielające pola można jednak zdefiniować dowolnie). Według tej definicji, polecenie `ls -l` wyświetla informacje w dziewięciu polach:

```
$ ls -l
-rw-r--r--    1 root    root     1973 Mar 12 01:12 inetd.conf
-rw-r--r--    1 root    root     6912 Mar 22  2002 mime.conf
-rw-r--r--    1 root    root      386 Mar 12 01:12 profile
drwxr-xr-x    1 root    root        0 Mar 12 01:06 profile.d
-rw-r--r--    1 root    root    12306 Mar 12 01:31 termcap
```

Domyślnie pola numerowane są `$1, $2, ..., $NF`, gdzie `NF` jest zmienną równą liczbie pól w danym wierszu. W tym przykładzie w każdym wierszu jest 9 pól. Należy odróżnić `NF`, czyli liczbę pól w wierszu, od `$NF` – numeru ostatniego pola w wierszu. Jeżeli wyświetlone mają być jedynie uprawnienia plików, rozmiar i nazwa, to należy wykonać następujące polecenie:

```
$ ls -l | awk '{ print $1, $5, $9 }'
-rw-r--r-- 1973 inetd.conf
-rw-r--r-- 6912 mime.conf
-rw-r--r-- 386 profile
drwxr-xr-x 0 profile.d
-rw-r--r-- 12306 termcap
```

## 4.2. Interpreter gawk

Gawk wczytuje źródło programu z pliku będącego argumentem opcji `--source` lub z pierwszego argumentu wiersza poleceń, który nie jest nazwą opcji. Opcje `-f` i `--source` mogą być użyte wielokrotnie w tym samym wierszu poleceń. Interpreter odczytuje tekst programu i pliki podlegające przetwarzaniu tak, jakby zostały połączone w całość. Jest to przydatne przy

tworzeniu bibliotek funkcji awk, by nie dołączać ich w każdym nowym programie awk.

Zmienna środowiskowa `AWKPATH` określa ścieżkę poszukiwania plików źródłowych określonych opcją `-f`. Jeżeli zmienna ta nie jest zdefiniowana, domyślną ścieżką jest zwykle `.:/usr/local/share/awk`. Jeżeli nazwa pliku, przekazana za pomocą opcji `-f`, zawiera znak `/`, ścieżka nie jest przeszukiwana.

Najpierw wykonywane są podstawienia zmiennych określonych opcją `-v`. Następnie gawk kompiluje program do postaci wewnętrznej. Wykonywany jest kod umieszczony w bloku `BEGIN`, po czym wczytywane są pliki, których nazwy znajdują się w tablicy `ARGV`.

Jeżeli w wierszu poleceń nie podano nazwy pliku z kodem źródłowym, intepreter wczytuje kod ze standardowego wejścia.

Jeżeli w wierszu poleceń pojawia się tekst w postaci `zmienna=wartosc`, traktowane jest to jako instrukcja przypisania wartości `zmiennej`. To przyporządkowanie wykonywane jest po zakończeniu bloku `BEGIN`. Używane jest także w sytuacji, kiedy skrypt wielokrotnie przetwarza ten sam plik danych. Po zakończeniu wczytywania danych wejściowych wykonywane są instrukcje zawarte w bloku `END`.

## 4.3. Opcje

Opcje interpretera gawk mogą być podawane w postaci jednoliterowej lub pełnej. Opcje zgodne z POSIX rozpoczynają się pojedynczym znakiem `-`, a opcje długie mają na początku dwa znaki `--`.

Zgodnie ze standardem POSIX, opcje specyficzne dla gawk są przekazywane jako argumenty opcji `-W`. Przekazać można wiele opcji `-W`. Każdej opcji `-W` odpowiada opcja w wersji dłuższej. Argumenty dłuższych opcji są złączone z opcją znakiem `=`, bez spacji, lub są przekazane w następnym argumencie wiersza poleceń. Długie opcje mogą być skrócone, o ile skrót jest jednoznaczny.

Oto opcje interpretera gawk.

`-F fs`
`--separator-pol fs`
Określa separator pól `fs` w danych wejściowych

`-v zmienna=wartosc`
`--assign zmienna=wartosc`
Przypisuje zmiennej wartość przed rozpoczęciem przetwarzania programu. Te wartości są dostępne w bloku `BEGIN` skryptu

`-f plik-programu`
`--file plik-programu`
Wczytuje kod źródłowy programu awk z pliku `program-file`. Można podać wiele opcji `-f` (lub `--file`)

`-mf NNN`
`-mr NNN`
Ustawia ograniczenia pamięci na wartość `NNN`. Znacznik `f` określa maksymalną liczbę pól, a `r` określa maksymalny rozmiar rekordu. Ponieważ w gawk nie zdefiniowano żadnych ograniczeń, opcja `-m` i jej znaczniki są ignorowane

`-W traditional`
`-W compat`
`--traditional`
`--compat`
Program wykonywany jest w trybie kompatybilności. Wówczas gawk działa tak, jak tradycyjny interpreter awk; nie są rozpoznawane rozszerzenia charakterystyczne dla wersji GNU. Spośród powyższych 4 możliwości preferowane jest użycie formy `--traditional`

`-W copyleft`
`-W copyright`
`--copyleft`
`--copyright`
Drukuje krótką informację o prawie autorskim

`-W help`
`-W usage`
`--help`
`--usage`
Drukuje na standardowym wyjściu krótki opis dostępnych opcji

`-W lint`
`--lint`
Ostrzega o elementach programu, które są wątpliwej jakości lub nie są zgodne z innymi implementacjami języka awk

`-W lint-old`
`--lint-old`
Drukuje ostrzeżenia o elementach programu, które nie są zgodne z pierwotną wersją interpretera awk w systemie Unix.

`-W posix`
`--posix`
Włącza tryb zgodności z następującymi dodatkowymi ograniczeniami:

Sekwencja `\x` nie jest rozpoznawana

Jedynie spacja i znak tabulacji działają jako separatory pól, jeżeli `FS` ma wartość jednej spacji; znak nowego wiersza nie ma tej własności

Ciąg `func` nie jest rozpoznawany jako synonim słowa kluczowego `function`

Nie można użyć operatorów `**` i `**=` w miejsce `^` i `^=`

Funkcja `fflush()` nie jest dostępna

`-W re-interval`
`--re-interval`
Włącza rozpoznawanie wyrażeń interwałowych w wyrażeniach regularnych. Wyrażenia interwałowe zostały wprowadzone w awk po to, by doprowadzić do zgodności składni wyrażeń regularnych w awk i egrep. Jednak użycie ich może spowodować kłopoty w starszych programach awk i dlatego nie są one rozpoznawane domyślnie. Podobny efekt ma opcja `--posix`

`-W source tekst-programu`
`--source tekst-programu`
Kod źródłowy ma być pobrany z pliku `tekst-programu`. Ta opcja umożliwia łatwe mieszanie funkcji zapisanych w bibliotece funkcji (można je wczytać z pliku bibliotecznego za pomocą opcji `-f` lub `--file`) z kodem źródłowym wpisanym w wierszu poleceń. Jest to przydatne zwłaszcza w skryptach awk średniej i dużej długości używanych w skryptach powłokowych

`-W version`
`--version`
Drukuje informację o wersji interpretera

`--`
Sygnalizuje koniec opcji. Dzięki temu dalsze argumenty programu awk mogą rozpoczynać się znakiem `-`. Obecność tej opcji wynika głównie z wymagania zgodności z konwencją obsługi opcji przyjętą w większości programów POSIX.

W trybie zgodności wszystkie inne opcje są traktowane jako nieprawidłowe i są ignorowane. W zwykłym trybie wszystkie opcje, także

nieznane są przekazywane do programu za pomocą tablicy `ARGV`. Przydatne jest to szczególnie do uruchamiania skryptów awk bezpośrednio ze skryptu za pomocą instrukcji `#!`

## 4.4. Zmienne

Zmienne awk są dynamiczne. Zaczynają istnieć w momencie pierwszego użycia. Ich wartościami są liczby zmiennoprzecinkowe lub łańcuchy znaków. W awk można używać tablic jednowymiarowych. Tablice wielowymiarowe również są dostępne, ale w rzeczywistości są symulowane za pomocą tablic jednowymiarowych. Zmienne domyślne są przedstawione w osobnym podrozdziale.

## 4.5. Zmienne domyślne

Oto pełna lista zmiennych zdefiniowanych domyślnie:

`ARGC`
Liczba argumentów wiersza poleceń

`ARGIND`
Indeks w tablicy `ARGV`, dotyczący przetwarzanego pliku

`ARGV`
Tabela zawierająca argumenty wiersza poleceń

`CONVFMT`
Format konwersji liczb; domyślnie `"%.6g"`

`ENVIRON`
Tablica wartości bieżących zmiennych środowiskowych. Tablica jest indeksowana nazwami zmiennych środowiskowych; na przykład zmienna `ENVIRON["HOME"]` może mieć wartość `/home/ktos`. Zmiana elementów tej tablicy nie wpływa na zmienne środowiskowe w programach uruchamianych w instrukcjach przekierowania lub za pomocą funkcji `system()`. Ta cecha może jednak się zmienić w przyszłych wersjach interpretera gawk

`ERRNO`
Jeżeli błąd systemowy wystąpi w czasie wykonywania instrukcji przekierowania lub w instrukcji `getline`, podczas odczytu w instrukcji `getline`, lub w instrukcji `close()`, zmienna `ERRNO` zawiera łańcuch będący opisem błędu

`FIELDWIDTHS`
Lista szerokości pól. Poszczególne elementy tej listy są od siebie oddzielone spacją. Jeżeli ta zmienna nie jest pusta, każdy rekord jest dzielony na pola o ustalonej długości, ignorując wartość zmiennej `FS`

`FILENAME`
Nazwa przetwarzanego pliku

`FNR`
Numer bieżącego rekordu w bieżącym pliku

`FS`
Separator pól (domyślnie spacja)

`IGNORECASE`
Jeżeli wartość tej zmiennej jest różna od zera, wielkie i małe litery nie są rozróżniane przy porównywaniu ciągów znaków, dzieleniu pól za pomocą zmiennej `FS`, separacji rekordów za pomocą zmiennej `RS`, dopasowaniu wyrażeń regularnych za pomocą operatorów `~` i `!~`, oraz w funkcjach `gensub()`, `gsub()`, `index()`, `match()`, `split()` i `sub()`

`NF`
Liczba pól rekordu

`NR`
Numer rekordu

`OFMT`
Format zapisu liczb na wyjściu; domyślnie `%.6g`

`OFS`
Separator pól na wyjściu (domyślnie spacja)

`ORS`
Separator rekordów na wyjściu; domyślnie `\n`

`RLENGTH`
Długość łańcucha dopasowanego przez funkcję `match`

`RS`
Separator rekordów na wejściu; domyślnie `\n`

`RSTART`
Pozycja początku łańcucha dopasowanego przez funkcję `match`

`RT`
Terminator rekordu. Domyślnie `RT` ma wartość zmiennej `RS`

`SUBSEP`
Separator indeksów tablicy; domyślnie `\034`

Różne wersje awk różnią się nieco zestawem zmiennych definiowanych domyślnie i możliwych operacji.

## 4.6. Rekordy

Domyślnie rekordy są oddzielone od siebie znakami nowego wiersza. To przyporządkowanie można zmienić za pomocą zmiennej wbudowanej `RS`. `RS` może być pojedynczym znakiem lub wieloznakowym wyrażeniem regularnym. Jednak w trybie zgodności jedynie pierwszy znak tego ciągu jest traktowany jako separator rekordów. Jeżeli `RS` ma wartość pustego łańcucha, rekordy są oddzielone pustymi wierszami. W takim przypadku znak nowego wiersza zawsze jest interpretowany jako separator pól, niezależnie od bieżącej definicji zmiennej `FS`. Jeżeli `RS` jest wyrażeniem regularnym, wartość zmiennej `IGNORECASE` ma wpływ na sposób interpretacji tej zmiennej.

## 4.7. Pola

Po wczytaniu każdego rekordu tekst jest dzielony na pola. Separatorem pól jest wartość zmiennej `FS`. `FS` może być pojedynczym znakiem, łańcuchem pustym `""` lub wyrażeniem regularnym. Jeżeli `FS` ma wartość pojedynczej spacji, pola są dzielone spacjami, znakami tabulacji lub znakami nowego wiersza. Jeżeli `FS` jest wyrażeniem regularnym, wartość zmiennej `IGNORECASE` ma wpływ na sposób interpretacji tej zmiennej.

Oto podsumowanie sposobu dzielenia pól w zależności od wartości `FS`.

`FS = ""`
Pola są separowane ciągami spacji. Spacje na początku i na końcu rekordu są ignorowane. Jest to interpretacja domyślna

`FS =` *dowolny pojedynczy znak*
Granice pól wyznacza jeden określony znak. Powtórzenie tego znaku oznacza pole puste. Wystąpienie tego znaku na początku rekordu lub na jego końcu również oznacza pole puste. Wartością zmiennej `FS` może być również metaznak używany w wyrażeniach regularnych. Nie musi być poprzedzony ukośnikiem `\`

`FS =` *wyrazenie regularne*
Pola są rozdzielone ciągami pasującymi do wyrażenia regularnego. Ciągi dopasowane na początku i na końcu rekordu wyodrębniają puste pola

`FS = ""`
Każdy znak jest oddzielnym polem. Taka interpretacja nie jest określona w standardzie POSIX

Jeżeli zmienna `FIELDWIDTHS` jest listą liczb oddzielonych od siebie spacjami, każde pole ma ustaloną szerokość i podział na pola zależy od szerokości pól, a nie wartości zmiennej `FS`. Przypisanie nowej wartości zmiennej `FS` anuluje działanie zmiennej `FIELDWIDTHS` i przywraca tryb domyślny.

Wartości pól rekordu wejściowego są oznaczane domyślnie symbolami `$1`, `$2` itd. Zapis `$0` oznacza cały rekord. Zapis wskazujący na pola nieistniejące, np. `$(NF+1)`, daje w wyniku łańcuch pusty.

## 4.8. Rekordy wielowierszowe

Jeżeli dane należące do jednego rekordu nie mieszczą się w jednym wierszu, można użyć rekordów wielowierszowych. Najpierw należy wybrać odpowiedni format danych.

Separatorem rekordów może być jakiś znak lub ciąg znaków, który nie pojawia się w zbiorze danych. Można np. użyć znaku *formfeed*, `\f`. Wówczas jeden rekord tworzy jedną stronę pliku. Instrukcja przypisania wartości zmiennej `RS` ma następującą postać: `RS = "\f"`

Inny sposób polega na rozdzielaniu rekordów pustymi wierszami. Jeżeli `RS` jest równy pustemu ciągowi znaków, `RS=""`, rekordy są odseparowane pustymi wierszami (jednym lub wieloma). Rekord kończy się po napotkaniu pierwszego pustego wiersza. Kolejny rekord zaczyna się dopiero po napotkaniu następnego niepustego wiersza. Wiersze będące separatorami nie mogą zawierać spacji ani znaków tabulacji.

Efekt identyczny z `RS = ""` można uzyskać, przypisując `RS = "\n\n+"`. To wyrażenie regularne pasuje do znaku nowego wiersza na końcu rekordu i jednego pustego wiersza następującego po rekordzie. Jeżeli kilka łańcuchów pasuje do wyrażenia regularnego, wyrażenie regularne jest zawsze dopasowane do najdłuższego z nich, licząc od lewej. Dlatego następny rekord zaczyna się dopiero po napotkaniu pierwszego niepustego wiersza, niezależnie od liczby kolejnych pustych wierszy.

Przypisanie `RS = ""` różni się od `RS = "\n\n+"`. W pierwszym przypadku puste wiersze na początku pliku są ignorowane, a jeśli po ostatnim rekordzie w pliku nie ma pustych wierszy, ostatni znak `\n` jest usuwany z rekordu. W przypadku drugiej definicji te czynności nie są wykonywane.

Jeżeli `RS` jest łańcuchem pustym, a `FS` jest przypisany pojedynczemu znakowi, pola są separowane w miejscu wystąpienia tego znaku oraz w miejscu wystąpienia znaku `\n`.

Jeżeli znak `\n` nie powinien pełnić roli separatora pól, można wykorzystać funkcję `split` w celu podzielenia rekordu w inny sposób. Jeżeli separatorem jest pojedynczy znak, można zdefiniować `FS` jako zawierające go wyrażenie regularne, np. zamiast `FS = "%"`, użyć `FS = "[%]"`.

Jeszcze inna metoda polega na umieszczeniu każdego pola w osobnym wierszu: `FS = "\n"`.

Oto podsumowanie sposobów dzielenia rekordów na podstawie wartości zmiennej `RS`:

- `RS = "\n"`
  Rekordy są rozdzielone znakiem nowego wiersza `\n`. Każdy wiersz, nawet pusty, jest odrębnym rekordem. Takie jest domyślne ustawienie tej zmiennej.

- `RS = dowolny znak`
  Rekordy są rozdzielone jednym, określonym znakiem. Jeżeli ten znak powtarza się wielokrotnie, kolejne rekordy są puste.

- `RS = ""`
  Rekordy są rozdzielone sekwencjami pustych wierszy. Znak `\n` jest traktowany jako separator pól, niezależnie od innych znaków określonych w definicji `FS`. Znaki `\n` pojawiające się na początku i na końcu pliku są ignorowane.

- `RS =` *wyrazenie regularne*
  Rekordy są rozdzielone ciągami znaków pasującymi do określonego wyrażenia regularnego. Wyrażenia dopasowane na początku i na końcu wiersza ograniczają puste rekordy. Takie zachowanie jest rozszerzeniem wprowadzonym w wersji gawk. Standard POSIX nie mówi nic na ten temat.

## 4.9. Wyrażenia regularne

Gawk obsługuje rozszerzony wariant składni wyrażeń regularnych, podobnie jak egrep. Oto elementy tych wyrażeń:

| Wyrażenie | Znaczenie |
|---|---|
| `a` | Znak `a` niebędący metaznakiem |
| `\a` | Znak `a` |
| `.` | Dowolny pojedynczy znak |
| `^` | Początek łańcucha znaków |
| `$` | Koniec łańcucha znaków |
| `[abc...]` | Klasa znaków, pasuje do dowolnego znaku z tego zbioru |
| `[^abc...]` | Negacja klasy znaków, pasuje do dowolnego znaku nienależącego do tego zbioru |
| `r1\|r2` | Alternatywa: `r1` lub `r2` |
| `r1r2` | Połączenie wyrażeń (konkatenacja): `r1`, po którym następuje `r2` |
| `r+` | Jedno lub więcej wystąpień wyrażenia `r` |
| `r*` | Zero lub więcej wystąpień wyrażenia `r` |
| `r?` | Zero lub jedno wystąpienie wyrażenia `r` |
| `(r)` | Łączenie znaków w grupę: pasuje do `r` |
| `r{n}` | Wyrażenie `r` powtórzone `n` razy |
| `r{n,}` | `r` powtórzone co najmniej `n` razy |
| `r{n,m}` | `r` powtórzone od `n` do `m` razy. **Uwaga**: Aby powyższe trzy typy wyrażeń interwałowych były dostępne, należy w wierszu poleceń użyć opcji `--posix` lub `--re-interval` |
| `\y` | Pusty ciąg znaków na początku lub na końcu słowa |
| `\B` | Pusty ciąg znaków wewnątrz słowa |
| `\<` | Pusty ciąg znaków na początku słowa |
| `\>` | Pusty ciąg znaków na końcu słowa |
| `\w` | Litera, cyfra lub znak podkreślenia `_` |
| `\W` | Dowolny znak niebędący literą, cyfrą lub znakiem podkreślenia |
| `` \` `` | Pusty ciąg na początku bufora |
| `\´` | Pusty ciąg na końcu bufora |

Klasy znaków od pewnego czasu są częścią standardu POSIX. Jest to notacja specjalna do opisu grup znaków mających określony atrybut. Zbiór znaków należących do danej klasy zależy od konfiguracji locale. Na przykład,

zbiory znaków alfabetu w USA i Polsce się różnią.

Klasa znaków oznaczona jest notacją `[:` , po której następuje słowo kluczowe określające klasę, a po nim `:]`. Oto klasy znaków zdefiniowane w standardzie POSIX.

- `[:alnum:]`
  Znaki alfanumeryczne
- `[:alpha:]`
  Znaki alfabetu
- `[:blank:]`
  Spacja lub znak tabulacji
- `[:cntrl:]`
  Znaki sterujące
- `[:digit:]`
  Cyfry
- `[:graph:]`
  Znaki, które są i drukowalne, i widzialne; spacja jest drukowalna, ale nie jest widzialna, a np. litera ą jest i drukowalna, i widzialna
- `[:lower:]`
  Małe znaki alfabetyczne
- `[:print:]`
  Znaki drukowalne
- `[:punct:]`
  Znaki przestankowe
- `[:space:]`
  Znaki odstępów, np. spacja, znak tabulacji, *formfeed*
- `[:upper:]`
  Wielkie znaki alfabetyczne
- `[:xdigit:]`
  Cyfry w notacji szesnastkowej

## 4.10. Symbole porządkujące

Istnieje specjalna notacja ułatwiająca przetwarzanie danych tekstowych. Dotyczy to zwłaszcza sortowania. Mają one zastosowanie w zbiorach znaków innych niż ASCII, którym mogą być przyporządkowane pojedyncze symbole zwane elementami porządkującymi (porównywania, ang. *collating elements*), reprezentowane za pomocą wielu znaków. Drugi rodzaj takich sekwencji to kilka znaków, które mają być traktowane jednakowo w procesie sortowania. Na przykład, w języku francuskim litera e i symbol é są równoważne.

Symbole porządkujące mogą składać się z wielu znaków umieszczonych między `[.` i `.]`. Na przykład, jeżeli `ch` jest elementem porządkującym, to `[[.ch.]]` jest wyrażeniem regularnym pasującym do tego elementu porządkującego, a `[ch]` jest wyrażeniem regularnym pasującym do znaku `c` lub `h`.

W przypadku problemów dotyczących kolejności sortowania warto m.in. sprawdzić ustawienia zmiennych określających locale. Na przykład, jeżeli wszystkie wielkie litery mają poprzedzać małe, należy ustawić zmienną `LC_ALL` i przekazywać jej wartość do procesów potomnych: `export LC_ALL=C`. Aby dokonać trwałej zmiany środowiska użytkownika w tym zakresie, można tę instrukcję umieścić w pliku `$HOME/.profile`.

## 4.11. Klasy równoważności

Klasa równoważności jest grupą znaków, które są sobie równoważne. Nazwa klasy jest ograniczona sekwencjami `[=` i `=]`. Na przykład, nazwa `e` może reprezentować dowolny znak spośród e, é i è. W tym przypadku `[[=e]]` jest wyrażeniem regularnym pasującym do e, é lub è.

Umożliwia to sprawne przetwarzanie tekstu w różnych językach. Funkcje interpretera gawk wykorzystywane do dopasowywania wyrażeń regularnych obsługują jedynie klasy znaków zdefiniowane w standardzie POSIX. Jak dotąd, nie obsługują symboli porządkujących i klas równoważności.

Operatory `\y`, `\B`, `\<`, `\>`, `\w`, `\W`, `\`` i `\´` są specyficzne dla interpretera gawk. Zostały stworzone na podstawie narzędzi przetwarzania wyrażeń regularnych dostępnych w ramach projektu GNU.

## 4.12. Opcje

Sposób interpretowania znaków w wyrażeniach regularnych można dodatkowo określić za pomocą opcji wiersza poleceń.

- Brak opcji

  Działanie domyślne. Wyrażenia interwałowe nie są obsługiwane

- `--posix`

  Obsługiwane są jedynie wyrażenia regularne POSIX. Wyrażenia specyficzne dla gawk nie są obsługiwane. Na przykład, `\w` pasuje do znaku `w`. Można używać wyrażeń interwałowych

- `--traditional`

  Wyrażenia regularne są obsługiwane w sposób zgodny z tradycyjnym Uniksem. Operatory charakterystyczne dla narzędzi GNU nie są rozpoznawane. Wyrażenia interwałowe są niedostępne. Klasy POSIX również nie są obsługiwane (np. `[[:alnum:]]`). Znaki oznaczane ósemkowymi i szesnastkowymi sekwencjami unikowymi (ang. *escape sequences*) są traktowanie jak literały także wtedy, kiedy reprezentują metaznaki wyrażeń regularnych

- `--re-interval`

  Dopuszcza wyrażenia interwałowe nawet wówczas, gdy użyto opcji `--traditional`

## 4.13. Operatory

Operatory w kolejności malejącego pierwszeństwa:

- `(...)`
  Łączenie wyrażeń
- `$`
  Numerator pól
- `++ --`
  Powiększenie, pomniejszenie wartości

- `^`
  Potęgowanie; można także użyć operatora `**` oraz `**=` jako operatora przypisywania wartości
- `+ - !`
  Znak liczby i negacja logiczna
- `*/ %`
  Mnożenie, dzielenie, dzielenie z resztą (modulo)
- `+ -`
  Dodawanie, odejmowanie
- `spacja`
  Łączenie ciągów
- `<>`
  `<= >=`
  `!= ==`
  Operatory relacji
- `~ !~`
  Dopasowanie wyrażenia regularnego, negacja dopasowania; wyrażenie regularne będące stałym ciągiem znaków może być użyte jedynie po prawej stronie relacji; wyrażenie typu `/ciag staly/ ~ wyrazenie regularne` jest traktowane jak `$0 ~ wyrazenie regularne`
- `in`
  Przynależność do tablicy
- `&&`
  Koniunkcja (AND); `wyrazenie1 && wyrazenie2` jest prawdziwe, jeżeli oba wyrażenia są prawdziwe
- `||`
  Alternatywa (OR); `wyrazenie1 || wyrazenie2` jest prawdziwe, jeśli choć jedno z dwóch wyrażeń jest prawdziwe; jeśli prawdziwe jest `wyrazenie1`, prawdziwość `wyrazenia2` już nie jest sprawdzana
- `?:`
  Wyrażenie warunkowe postaci `wyr1 ? wyr2 : wyr3`. Jeżeli `wyr1` jest prawdą, wyrażenie ma wartość `wyr2`, w przeciwnym razie jest nią `wyr3`
- `= += -= *= /= %=`
  Przypisanie wartości

## 4.14. Instrukcje sterujące

- ```
  if (warunek)
    instrukcja1
  else
    instrukcja2
  ```

  Jeżeli wykonywanych jest wiele operacji, można je zgrupować za pomocą nawiasów klamrowych:

  ```
  if (warunek)
  {
    instrukcja1
    instrukcja2
  }
  else
    instrukcja3
  ```

- ```
  while (wyrazenie)
  instrukcja
  ```

  lub

  ```
  while (wyrazenie)
  {
    instrukcja1
    instrukcja2
  }
  ```

- ```
  do { instrukcje }
  while (warunek)
  ```

- ```
  for (wyrazenie1; warunek; wyrazenie2)
    instrukcja
  ```

  lub

  ```
  for (wyrazenie1; warunek; wyrazenie2)
  {
    instrukcja1
    instrukcja2
    ...
  }
  ```

lub

```
for (zmienna in tablica)
{ instrukcje }
```

- `break`
- `continue`
- `delete tablica[indeks]`
- `delete tablica`
- `exit [ wyrazenie ]`
  `{ instrukcje }`

## 4.15. Tablice

W awk można korzystać z tablic, których elementami są liczby lub ciągi znaków. Tablice są deklarowane automatycznie w momencie użycia w programie.

**Przykład**. Drukowanie wierszy w kolejności odwrotnej.

```
{ wiersz[NR] = $0 }
END {
for (i=NR; i>0; i=i-1)
  print wiersz[i]
}
```

## 4.16. Pliki o specjalnym znaczeniu

W instrukcjach `print`, `printf` i `getline` mogą być wykorzystywane nazwy plików specjalnych. Deskryptory plików otwartych dziedziczone są z procesu nadrzędnego, którym jest zwykle powłoka. Niektóre z tych plików są źródłem informacji o bieżącym procesie interpretera gawk:

- `/dev/pid`
  Odczyt tego pliku zwraca identyfikator bieżącego procesu w postaci liczby dziesiętnej zakończonej znakiem nowego wiersza

- `/dev/ppid`
  Odczyt tego pliku zwraca identyfikator procesu nadrzędnego względem procesu bieżącego w postaci liczby dziesiętnej zakończonej znakiem nowego wiersza

- `/dev/pgrpid`
  Odczyt tego pliku zwraca identyfikator grupowy procesu bieżącego w postaci liczby dziesiętnej zakończonej znakiem nowego wiersza

- `/dev/user`
  Odczyt tego pliku zwraca pojedynczy rekord zakończony znakiem nowego wiersza. Pola są rozdzielone spacjami. `$1` jest wartością funkcji systemowej `getuid(2)`, `$2` jest wartością funkcji systemowej `geteuid(2)`, `$3` jest wartością funkcji systemowej `getgid(2)`, a `$4` jest wartością funkcji systemowej `getegid(2)`. Dalsze pola, o ile istnieją, są identyfikatorami grup, będącymi wynikiem funkcji `getgroups(2)`

- `/dev/stdin`
  Standardowe wejście

- `/dev/stdout`
  Standardowe wyjście

- `/dev/stderr`
  Standardowe wyjście dla błędów

- `/dev/fd/`$n$
  Plik odpowiadający deskryptorowi otwartego pliku `n`

Oto przykład:

```
print "To jest komunikat diagnostyczny" > "/dev/stderr"
```

Ta sama instrukcja bez użycia plików specjalnych miałaby następującą postać:

```
print "To jest komunikat diagnostyczny" | "cat 1>&2"
```

## 4.17. Funkcje arytmetyczne

- `atan2(y,x)`
  arcus tangens argumentu y/x w zakresie od $-\pi$ do $\pi$
- `cos(x)`
  cosinus
- `sin(x)`
  sinus
- `exp(x)`
  e^x
- `log(x)`
  logarytm naturalny
- `int(x)`
  część całkowita $x$; obcięcie następuje w kierunku 0
- `sqrt(x)`
  pierwiastek kwadratowy
- `rand(x)`
  liczba pseudolosowa z przedziału $0 < x < 1$
- `srand(x)`
  inicjalizacja zmiennej pseudolosowej `rand()` za pomocą liczby $x$

## 4.18. Operacje na ciągach znaków

Następujące funkcje umożliwiają wykonywanie podstawowych operacji na ciągach znaków:

- `asort(in [, out])`
  Zwraca liczbę elementów tablicy `in`. Jej zawartość jest posortowana zgodnie ze zwykłymi regułami interpretera. Wartość zmiennej `IGNORECASE` ma wpływ na wynik sortowania. Indeksy posortowanych wartości są zastąpione kolejnymi liczbami całkowitymi, poczynając od jedynki. Jeżeli funkcja jest wywołana z dwoma argumentami, pierwotna postać tablicy jest kopiowana do drugiego argumentu, po czym sortowana jest tablica docelowa, a indeksy tablicy `in` pozostają niezmienione.

Na przykład, jeżeli tablica `a` ma następującą postać:

```
a["ostatni"] = "ma"
a["pierwszy"] = "kota"
a["srodkowy"] = "ala"
```

W wyniku instrukcji `asort(a)` otrzymujemy następującą postać `a`:

```
a[1] = "ala"
a[2] = "kota"
a[3] = "ma"
```

Ta funkcja istnieje tylko w gawk. W innych interpreterach jest niedostępna. Staje się niedostępna również wtedy, kiedy gawk jest uruchomiony w tzw. trybie zgodności

- `asorti(in [, out])`
  Zwraca liczbę elementów talicy `in`. Działanie tej funkcji przypomina `asort`. Różnica polega na tym, że tym razem sortowane są indeksy, a nie elementy tablicy. Ponieważ indeksy tablicy są zawsze łańcuchami, podczas sortowania porównywane są łańcuchy. Wartość zmiennej `IGNORECASE` ma wpływ na sortowanie.

  Funkcja `asorti` jest specyficzna dla interpretera gawk. W trybie zgodności jest niedostępna

- `gensub(r,s,h,[ t])`
  Wyszukuje w łańcuchu znaków `t` ciągów pasujących do wyrażenia regularnego `r`. Jeżeli `h` jest łańcuchem rozpoczynającym się znakiem `g` lub `G`, wszystkie ciągi pasujące do `r` są zastępowane ciągiem `s`. W przeciwnym razie `h` jest liczbą określającą liczbę kolejnych podstawień, jakie należy wykonać. W przypadku pominięcia argumentu `t`, działanie jest wykonywane na zmiennej `$0`. W ciągu `s` można użyć sekwencji `\n`, gdzie `n` jest cyfrą od 1 do 9, w celu oznaczenia tekstu pasującego do `n`-tego wyrażenia ujętego w nawias. Ciąg `\0` oznacza cały dopasowany tekst, podobnie jak znak `&`. W odróżnieniu do funkcji `sub()` i `gsub()`, zmodyfikowany ciąg nie jest zwracany i łańcuch `t` nie ulega zmianie

- `gsub(r,s)`
  W miejsce ciągu `r` podstawia ciąg `s` w ciągu `$0`, czyli w całym bieżącym rekordzie; zwraca liczbę wykonanych podstawień

- `gsub(r,s,t)`
  W miejsce `r` podstawia `s` w ciągu `t`; zwraca liczbę podstawień

- `index(s,t)`
  Zwraca pozycję pierwszego wystąpienia ciągu `t` w ciągu `s`; zwraca `0`, jeśli `t` nie pojawia się ani razu

- `length(s)`
  Zwraca liczbę znaków ciągu `s`

- `match(s,r)`
  Sprawdza, czy `s` zawiera ciąg znaków pasujący do `r`; zwraca indeks wskazujący położenie `r` w `s`; nadaje wartości zmiennym `RSTART` i `RLENGTH`

- `split(s,a)`
  Dzieli ciąg znaków s na części, które umieszcza w tablicy `a`; dzielenie następuje domyślnie w miejscach wystąpień zmiennej `FS`; zwraca liczbę powstałych pól

- `split(s,a,fs)`
  Dzieli ciąg znaków s na części, które umieszcza w tablicy `a`; dzielenie następuje w miejscach wystąpień separatora pól `fs`; zwraca liczbę powstałych pól

- `sprintf(fmt,lista_wyr)`
  Zwraca listę wyrażeń sformatowaną według formatu `fmt`

- `strtonum(str)`
  Zwraca numeryczną wartość łańcucha `str`. Jeżeli `str` rozpoczyna się znakiem `0`, funkcja `strtonum` traktuje `str` jako liczbę w formacie ósemkowym. Jeżeli zmienna `str` rozpoczyna się ciągiem `0x` lub `0X`, jest traktowana jako liczba szesnastkowa. Na przykład:

```
$ echo 0x11 |
  > gawk '{ printf "%d\n", strtonum($1) }'
  -| 17
```

  `strtonum` wykonuje automatyczną konwersję tylko w przypadku liczb w formacie dziesiętnym. Ta funkcja jest rozszerzeniem wprowadzonym w gawk. W trybie zgodności jest niedostępna.

- `sub(r,s)`
  Podstawia `s` w miejsce pierwszego (od lewej strony) wystąpienia ciągu znaków zawartego w `$0`, pasującego do `r`; zwraca liczbę podstawień

- `sub(r,s,t)`
  Podstawia `s` w miejsce pierwszego (od lewej strony) wystąpienia ciągu znaków zawartego w `t` i pasującego do `r`; zwraca liczbę podstawień

- `substr(s,p)`
  Zwraca końcową część ciągu znaków `s`, poczynając od miejsca `p`

- `substr(s,p,n)`
  Wybiera z całego ciągu `s` podciąg o długości `n`, zaczynający się od miejsca `p`

- `tolower(s)`
  Zwraca kopię łańcucha `s`, w której w miejsce wszystkich wielkich liter umieszczone są odpowiednie małe litery. Inne znaki pozostają niezmienione

- `toupper(s)`
  Zwraca kopię łańcucha `s`, w której w miejsce wszystkich małych liter umieszczone są odpowiednie wielkie litery. Inne znaki pozostają niezmienione

W przypadku instrukcji `sub`, `gsub` i `gensub` należy pamiętać, że istnieje kilka poziomów przetwarzania sekwencji unikowych (ang. *escape sequences*).

Pierwszym poziomem jest poziom leksykalny. Interpreter wczytuje program i tworzy jego kopię wewnętrzną. Na poziomie wykonawczym (*runtime*) awk skanuje ciąg zastępczy i tworzy łańcuch, który wchodzi na miejsce zastępowanego.

W obu przypadkach awk poszukuje określonych ciągów znaków, następujących po ukośniku `\`. Na poziomie leksykalnym wyszukuje odpowiednie sekwencje unikowe. Na każdy znak `\` przetwarzany na etapie wykonawczym należy wpisać dwa ukośniki na poziomie leksykalnym. Jeżeli po ukośniku następuje znak, który nie tworzy żadnej z rozpoznawanych sekwencji unikowych, awk usuwa początkowy ukośnik `\` i umieszcza następny znak w łańcuchu znaków. Na przykład, ciąg `a\xy` jest traktowany jako `axy`.

Na poziomie wykonawczym różne funkcje różnie przetwarzają sekwencje `\` i `&`. W przeszłości funkcje `sub` i `gsub` różnie przetwarzały ciąg znaków `\&`. W tekście wynikowym ta sekwencja była zastępowana pojedynczym znakiem `&`. Jeżeli w podstawianym tekście znak `\` nie poprzedzał znaku `&`, pojawiał się

niezmieniony w tekście wynikowym. Następująca tabela przedstawia tradycyjny sposób interpretacji różnych sekwencji unikowych w instrukcjach `sub` i `gsub`.

```
tekst      co widzi     wynik dzialania
pierwotny  funkcja sub  funkcji sub

       \&       &       dopasowany tekst
      \\&      \&       literal &
     \\\&      \&       literal &
    \\\\&     \\&       literal \&
   \\\\\&     \\&       literal \&
  \\\\\\&    \\\&       literal \\&
      \\a      \a       literal \a
```

W powyższej tabeli pokazano przetwarzanie na poziomie leksykalnym, kiedy nieparzysta liczba znaków ukośnika zmienia się w parzystą liczbą na poziomie wykonawczym oraz przetwarzanie za pomocą funkcji `sub` na poziomie wykonawczym. Dla uproszczenia, w pozostałej części tego podrozdziału przykłady w tabelach dotyczą jedynie parzystej liczby ukośników na poziomie leksykalnym.

Problem polega na tym, że w tradycyjnym podejściu nie można uzyskać ciągu składającego się z ukośnika, po którym następuje dopasowany tekst.

W standardzie POSIX z 1992 r. próbowano rozwiązać ten problem. Sformułowano to tak, że funkcje `sub` i `gsub` mają wyszukiwać (a) znaku \, lub (b) znaku `&`, następującego po \. W obu przypadkach taki znak jest przeniesiony do danych wyjściowych bez zmian. Sposób interpretacji znaku \ i `&` przedstawia następująca tabela:

```
tekst      co widzi     wynik dzialania
pierwotny  funkcja sub  funkcji sub

      &           &     dopasowany tekst
    \\&          \&     literal &
  \\\\&         \\&     literal \, dalej dopasowany tekst
\\\\\\&        \\\&     literal \&
```

Wydawać by się mogło, że problem usunięto. Niestety, sformułowanie użyte w standardzie ma następujące skutki:

- w łańcuchu podstawianym w miejsce innego ukośniki muszą być podwojone, co skutkuje niekompatybilnością z wcześniejszymi programami w języku awk

- aby mieć pewność, że program awk jest przenośny, każdy znak łańcucha wstawianego w miejsce innego należy poprzedzić ukośnikiem.

W 1996 r. zaproponowano użycie reguł uzupełnionych o wyjątki umożliwiające umieszczenie znaku \ przed dopasowanym tekstem:

```
tekst      co widzi     wynik dzialania
pierwotny  funkcja sub  funkcji sub

  \\\\\\&       \\\&    literal \&
    \\\\&        \\&    literal \, dalej dopasowany tekst
      \\&         \&    literal &
      \\a         \a    literal \a
     \\\\         \\    \\
```

Na poziomie wykonawczym są teraz trzy rodzaje sekwencji unikowych (\\\&, \\& i \&), a w tradycyjnym ujęciu był tylko jeden taki typ. Podobnie jednak, jak w przypadku tradycyjnym, znak & nienależący do żadnego z powyższych trzech typów nie ma specjalnego znaczenia i jest przenoszony do wyniku bez zmian.

W wersji gawk 3.0 i 3.1 zastosowano proponowane reguły POSIX dotyczące funkcji `sub` i `gsub`.

Zgodnie ze standardem POSIX z 2001 r., sekwencja \& w łańcuchu wchodzącym na miejsce innego daje literał &, sekwencja \\ daje literał \, a w pozostałych przypadkach, kiedy po znaku \ następuje inny znak, nie są wykonywane żadne działania i znak \ jest umieszczony na wyjściu.

```
tekst      co widzi     wynik dzialania
pierwotny  funkcja sub  funkcji sub

  \\\\\\&      \\\&      literal \&
    \\\\&       \\&      literal \, dalej dopasowany tekst
      \\&        \&      literal &
      \\a        \a      literal \a
     \\\\        \\      \
```

Różnica jest widoczna jedynie w ostatnim przykładzie: \\\\ jest widziany jako \\ i daje w wyniku \, zamiast \\.

Począwszy od wersji 3.1.4, gawk działa zgodnie ze standardem POSIX, jeżeli użyto opcji `--posix`. W przeciwnym razie interpreter stosuje się do reguł proponowanych w 1996 r.

**Uwaga:** w przyszłości gawk domyślnie będzie stosował reguły standardu POSIX 2001.

Reguły dotyczące funkcji `gensub` są znacznie prostsze. Jeżeli na poziomie wykonawczym po \ następuje cyfra, tekst dopasowany wyrażeniem ujętym w nawiasie zostaje umieszczony na wyjściu. W pozostałych przypadkach każdy znak następujący po \ pojawia się w tekście wynikowym, a \ znika, jak to pokazano w następującej tabeli.

```
tekst      co widzi     wynik dzialania
pierwotny  funkcja sub  funkcji gensub

      &        &        dopasowany tekst
    \\&       \&        literal &
   \\\\       \\        literal \
  \\\\&      \\&        literal \, dalej dopasowany tekst
\\\\\\&     \\\&        literal \&
    \\a       \a        literal a
```

Ze względu na złożoność przetwarzania na etapie leksykalnym i wykonawczym oraz istnienie wyjątków w przetwarzaniu za pomocą funkcji `sub` i `gsub`, zaleca się używać do celów podstawiania tekstu interpretera gawk i funkcji `gensub`.

W awk operator `*` może dopasować łańcuch pusty (jest to szczególnie ważne w przypadku funkcji `su`, `gsub` i `gensub`), na przykład:

```
$ echo abc | awk '{ gsub(/m*/, "X"); print }'
-| XaXbXcX
```

Mimo że wydaje się to sensowne, może czasem być źródłem niespodzianek.

## 4.19. Obsługa czasu

Typowym zadaniem wykonywanym za pomocą języka awk jest przetwarzanie plików dzienników komputerowych (ang. *log files*) zawierających stempel czasu. W tym celu wykorzystywane są dwie funkcje:

- `systime()`
  Zwraca czas bieżący w sekundach, licząc od północy czasu UTC, 1 stycznia 1970

- `strftime([format [, stempelczasu]])`
  Formatuje stempel czasu zgodnie ze specyfikacją podaną w formacie. Stempel czasu powinien mieć tę samą postać, jak wynik polecenia `systime()`. Jeżeli go brak, używa się bieżącego czasu dnia. W przypadku braku formatu używany jest domyślny format równoważny wynikowi instrukcji `date(1)`.

## 4.20. Funkcje użytkownika

Funkcje definiowane są następująco:

```
function nazwa(lista_parametrów) {
  instrukcje }
```

Parametry funkcji są oddzielane przecinkami. Funkcja może zawierać instrukcję `return` powodującą wyjście z funkcji do programu wywołującego w następującej postaci:

```
return wyrazenie
```

Jeżeli instrukcja `return` nie pojawia się nigdzie w funkcji lub jeśli ostatnia wykonywana instrukcja jest inna niż `return`, żadna wartość nie jest zwracana przez funkcję na wyjściu.

**Przykład**. Funkcja zwracająca większy z dwóch argumentów:

```
function max(m,n) {
  return m > n ? m : n }
```

Wywołania funkcji mogą znajdować się zarówno we wzorcach, jak i instrukcjach skryptu. Tablice są przekazywane przez wskazanie. Pozostałe zmienne są przekazywane przez wartość.

Początkowo funkcji nie było w awk. W związku z tym obsługa zmiennych lokalnych jest niezbyt elegancka. Są one deklarowane jako dodatkowe parametry na liście parametrów przekazywanych do funkcji. Zmienne lokalne są oddzielone od parametrów dodatkowymi spacjami. Na przykład:

```
function  f(p, q,     a, b)   # a i b sa zmiennymi lokalnymi
{
  ...
}
/abc/     { ... ; f(1, 2) ; ... }
```

Lewy nawias w wywołaniu funkcji musi pojawiać się bezpośrednio po nazwie funkcji. Ma to na celu uniknięcie syntaktycznej kolizji z operatorem konkatenacji (łączenia). Ta reguła nie dotyczy funkcji wbudowanych.

Wywołania fukcji mogą być zagnieżdżone i rekursywne. Parametry funkcji używane w roli zmiennych lokalnych są inicjalizowane do łańcucha pustego lub liczby zero w chwili wywołania funkcji.

W przypadku użycia opcji `--lint` gawk ostrzega o wywołaniach do niezdefiniowanych funkcji podczas parsowania, zamiast podczas wykonywania skryptu. Próba wywołania niezdefiniowanej funkcji podczas wykonywania skryptu powoduje zakończenie skryptu.

Zamiast słowa kluczowego `function` można użyć skrótu `func`.

## 4.21. Instrukcje wejścia i wyjścia

Instrukcja `print` służy do zwykłego drukowania, a `printf` – do drukowania tekstu sformatowanego. Domyślnie instrukcja `print` drukuje zawartość zmiennej `$0` na standardowym wyjściu. Następująca instrukcja drukuje wyrażenia oddzielone od siebie znakiem (lub ciągiem znaków) przypisanym zmiennej `OFS`:

```
print wyrazenie, wyrazenie, ...
```

Wynik tej instrukcji kończy się znakiem (lub ciągiem znaków) przypisanym zmiennej `ORS`.

Wynik instrukcji drukowania może być przekazany do pliku lub do innego polecenia:

```
print wyrazenie1, wyrazenie2, ... > plik
print wyrazenie1, wyrazenie2, ... >> plik
print wyrazenie1, wyrazenie2, ... | polecenie
```

Instrukcje `printf` mają postać podobną do instrukcji `print`. Różnią się obecnością pierwszego argumentu, definiującego format drukowania, na przykład:

```
printf(format, wyrazenie, wyrazenie, ...)
```

**Przykład**. Następujące polecenie drukuje wartość dwóch zmiennych, `a` i `b`, z których pierwsza jest ciągiem znaków, a druga liczbą całkowitą:

```
printf("%s %d \n", a, b)
```

W tabeli umieszczone są symbole formatów instrukcji `printf`.

| Format | Znaczenie |
|---|---|
| `c` | Znak ASCII |
| `d` | Liczba całkowita w notacji dziesiętnej |
| `e` | Liczba dziesiętna w notacji wykładniczej |
| `f` | Liczba dziesiętna |
| `g` | Liczba dziesiętna zapisana w formacie `e` lub `f`; automatycznie wybrany jest format dający krótszy zapis |
| `o` | Liczba całkowita zapisana w notacji ósemkowej |
| `s` | Ciąg znaków |
| `x` | Liczba całkowita zapisana w notacji szesnastkowej |

Instrukcja `printf("%%")` drukuje znak `%`.

Operatory przekierowania `>` i `>>` służą do kierowania wyjścia do plików, zamiast na standardowe wyjście.

**Przykład**.

```
$2 > 10 { print $1, $2 > "plik1" }
$2 <= 10 { print $1, $2 > "plik2" }
```

Nazwy plików muszą być podane w cudzysłowie. Mogą one być zarówno zmiennymi, jak i wyrażeniami:

```
{ print($1, $2) > ($2 > 10 ? "plik1" : "plik2") }
{ print > $1 }
```

Operator przekierowania otwiera plik tylko raz. Przy kolejnych operacjach zapisywania dane są dopisywane do otwartego pliku. Operator `>>` powoduje, że dotychczasowa zawartość pliku nie jest usuwana, a nowe dane są dopisywane na końcu pliku.

Wynik programu może być przekazany do innego polecenia za pomocą potoku:

```
print | polecenie
```

Oto pozostałe instrukcje wejścia i wyjścia:

- `close(plik)`

  Zamyka *plik* (lub potok)

- `getline`

  Przypisuje zmiennej `$0` zawartość następnego rekordu; nadaje wartości zmiennym `NF`, `NR`, `FNR`

- `getline <plik`

  Przypisuje zmiennej `$0` zawartość następnego rekordu pochodzącego z *pliku*; nadaje wartość zmiennej `NF`

- `getline zmienna`

  Przypisuje *zmiennej* zawartość następnego rekordu; nadaje wartości zmiennym `NR`, `FNR`

- `getline zmienna <plik`

  Przypisuje *zmiennej* zawartość następnego rekordu

- `next`

  Kończy przetwarzanie bieżącego rekordu wejściowego. Odczytywany jest następny rekord wejściowy i przetwarzanie rozpoczyna się od początku programu awk (za blokiem `BEGIN`). Po zakończeniu strumienia danych wykonywane są polecenia zawarte w bloku `END`

- `nextfile`

  Kończy przetwarzanie bieżącego pliku wejściowego. Następny rekord odczytywany jest z kolejnego pliku wejściowego. Uaktualnieniu ulegają wówczas zmienne `FILENAME` i `ARGIND`, zmienna `FNR` otrzymuje wartość równą 1 i przetwarzanie jest wznawiane na początku programu awk (za blokiem `BEGIN`). Po zakończeniu strumienia danych wykonywane są polecenia zawarte w bloku `END`. Uwaga: we wcześniejszych wersjach gawk ta instrukcja składała się z dwóch wyrazów: `next file`. Ten zapis jest chwilowo dopuszczalny, ale w przyszłości stanie się niepoprawny

- `system(polecenie)`

  Wykonuje *polecenie* i zwraca jego kod wynikowy (ang. *exit status*)

- `fflush([plik])`

  Opróżnia wszystkie bufory związane z otwartymi plikami lub potokami. or pipe file. Jeżeli brak nazwy pliku, opróżniany jest bufor dla standardowego wyjścia. Jeżeli plik jest łańcuchem pustym, opróżniane są bufory wszystkich otwartych plików wyjściowych i potoków

## 4.22. Przekazywanie wartości zmiennych powłokowych do skryptu awk

- Jeżeli zmienna powłokowa ma być dostępna w bloku `BEGIN{}`, można wykorzystać opcję `-v`:

  ```
  awk -v zmienna="wartosc zmiennej" ...
  ```

- Zmienne można także przekazywać w wierszu poleceń:

  ```
  $ awk '{...}' zm1="wartosc 1" plik1 \
                zm2="wartosc 2" plik2 ...
  ```

  Dzięki temu można nadawać zmiennym różne wartości podczas przetwarzania różnych plików.

- Wykorzystanie wartości zmiennej zdefiniowanej uprzednio w powłoce:

  ```
  $ zmienna="wartosc zmiennej"
  $ awk '{ print "'"$zmienna"'" }' plik
  ```

- Można wykorzystać tablicę `ENVIRON`, zawierającą wartości zmiennych środowiskowych:

  ```
  $ export ZMIENNA="wartosc zmiennej"
  $ awk '{ print ENVIRON["ZMIENNA"] }'
  ```

## 4.23. Komunikacja z innym procesem

Przesyłanie danych do innego programu i z powrotem można wykonać za pomocą plików tymczasowych:

```
# zapisz dane w pliku tymczasowym
plik_tymcz = ("mojedane." PROCINFO["pid"])
while (dopoki sa dane)
  print dane | ("podprogram > " plik_tymcz)
close("podprogram > " plik_tymcz)
```

```
# odbierz wyniki, usun plik tymczasowy
while ((getline nowedane < plik_tymcz) > 0)
  tu przetwarzanie danych
close(plik_tymcz)
system("rm " plik_tymcz)
```

Nie jest to jednak najbardziej eleganckie rozwiązanie. Nie można, na przykład, uruchamiać tego programu w katalogu, do którego inni użytkownicy mają dostęp, na przykład `/tmp`, ponieważ inny użytkownik mógłby korzystać z pliku o tej samie nazwie.

Począwszy od wersji 3.1, gawk może otworzyć potok do innego procesu i przesyłać dane w obu kierunkach. Proces po drugiej stronie potoku nazywany jest koprocesem, ponieważ biegnie równolegle do intepretera gawk. Potok otwierany jest za pomocą operatora `|&`, zapożyczonego z powłoki Korna (ksh):

```
do {
  print dane |& "podprogram"
  "podprogram" |& getline wyniki
} while (sa dane do przetwarzania)
close("podprogram")
```

Przy pierwszym uruchomieniu operatora gawk tworzy dwukierunkowy potok do procesu potomnego, który wykonuje inny program. Wyniki drukowane instrukcją `print` lub `printf` na standardowym wyjściu programu można odczytać w skrypcie awk za pomocą instrukcji `getline`. Podobnie, jak w przypadku procesów uruchomionych za pomocą operatora `|`, podprogram może być dowolnym programem lub potokiem programów, które mogą być uruchomione z powłoki.

Standardowym wyjściem błędów koprocesu jest standardowe wyjście błędów skryptu gawk.

Buforowanie wejścia i wyjścia może czasem powodować problemy. gawk automatycznie opróżnia bufor potoku, kierując dane do procesu potomnego. Jeżeli jednak koproces nie opróżnia automatycznie bufora danych, gawk może wstrzymać wykonywanie skryptu w oczekiwaniu na zakończenie instrukcji `getline`. Może to prowadzić do sytuacji, w której procesy wzajemnie się blokują w oczekiwaniu na zakończenie czynności drugiego procesu.

Można zamknąć jeden koniec potoku dwukierunkowego do koprocesu, przekazując drugi argument (`to` lub `from`) do funkcji `close`.

Jest to niezbędne zwłaszcza w celu użycia instrukcji `sort` jako składnika koprocesu. `sort` drukuje wynik dopiero po odczytaniu wszystkich danych wejściowych, po zamknięciu potoku po stronie zapisu.

Po zakończeniu przekazywania danych do programu `sort` można zamknąć końcówkę `to` potoku i rozpocząć odczyt danych za pomocą polecenia `getline`. Na przykład:

```
BEGIN {
  polecenie = "LC_ALL=C sort"
  n = split("abcdefghijklmnopqrstuvwxyz", a, "")
  for (i = n; i > 0; i--)
    print a[i] |& polecenie
  close(polecenie, "to")
  while ((polecenie |& getline wiersz) > 0)
    print "otrzymano", wiersz
  close(polecenie)
}
```

Program zapisuje litery alfabetu w kolejności odwrotnej, po jednej w wierszu, do potoku w celu sortowania. Następnie zamyka potok po stronie zapisu, aby `sort` odebrał informację o zakończeniu pliku. Posortowane dane wędrują z powrotem do skryptu gawk. Po odczytaniu wszystkich danych koproces zostaje zamknięty i skrypt kończy działanie.

Dyrektywa `LC_ALL=C` w poleceniu `sort` powoduje, że sortowanie wykonywane jest w sposób tradycyjnie stosowany w Uniksie, tj. zgodnie z porządkiem ASCII.

Począwszy od wersji gawk 3.1.2, komunikacja z innymi procesami może odbywać się także za pomocą pseudoterminali (`pty`), o ile system obsługuje ten rodzaj komunikacji.

```
# polecenie zapisane dla wygody jako zmienna
polecenie = "sort -nr"
# uaktualnienie PROCINFO
PROCINFO[polecenie, "pty"] = 1
# otwarcie dwukierunkowego potoku
print ... |& polecenie
...
```

Użycie pseudoterminali pozwala uniknąć blokady bufora opisanej wcześniej, za cenę niewielkiej utraty wydajności. Jeżeli system nie obsługuje pseudoterminali lub jeśli wszystkie `pty` są zajęte, następuje automatyczne przejście na komunikację za pomocą zwykłych potoków.

## 4.24. Programowanie sieciowe

Gawk umożliwia komunikację z procesami na innym komputerze za pośrednictwem łącza IP. W połączeniach TCP/IP należy wówczas użyć nazw plików specjalnych, rozpoczynających się ciągiem `/inet/`. Oto ogólna postać nazwy pliku odpowiadającego temu połączeniu:
`/inet/protokol/port-lokalny/komp-odlegly/port-na-komp-odleglym`. Oto znaczenie poszczególnych składników:

- `protokol`
  Protokół sieciowy: `tcp`, `udp` lub `raw`

- `port-lokalny`
  Lokalny port TCP lub UDP. Jeżeli ma wartość `0`, system przydziela numer portu według własnych reguł. Takie postępowanie zaleca się w przypadku tworzenia programu dla klienta TCP lub UDP. Można także użyć powszechnie znanych nazw usług, np. `smtp` lub `http`. W takim przypadku gawk sprawdza za pomocą funkcji `getservbyname` języka C, który port jest przydzielony danej usłudze

- `komp-odlegly`
  Adres IP komputera odległego lub jego pełna nazwa

- `port-na-komp-odleglym`
  Numer portu TCP lub UDP na komputerze odległym. Reguły wybierania numeru portu są takie same, jak dla portu lokalnego.

Oto prosty przykład:

```
BEGIN {
  Usluga = "/inet/tcp/0/localhost/daytime"
  Usluga |& getline
  print $0
  close(Usluga)
}
```

Program odczytuje bieżącą datę i czas z serwera czasu `daytime`. Po wydrukowaniu wyniku połączenie zostaje zamknięte.

Zastosowanie gawk do programowania sieciowego to szerszy temat i w celu zapoznania się z nim należy sięgnąć do publikacji specjalnie poświęconych temu tematowi (zobacz *TCP/IP Internetworking with gawk*).

## 4.25. Przetwarzanie opcji wiersza poleceń

Większość narzędzi w systemach zgodnych ze standardem POSIX przyjmuje opcje przekazywane w wierszu poleceń. Często opcje wymagają podania argumentu. Na przykład, uruchomienie awk z opcją `-F` wymaga podania łańcucha pełniącego rolę separatora pól. Opcje awk są zakończone w miejscu pierwszego pojawienia się `--` lub łańcucha nierozpoczynającego się znakiem `-`.

W systemach Unix zwykle dostępna jest funkcja `getopt` w języku C, służąca do przetwarzania argumentów wiersza poleceń. Do funkcji przekazywany jest łańcuch zawierający opcje jednoliterowe. Jeżeli opcja wymaga podania argumentu, jest on umieszczony w tym łańcuchu z dwukropkiem. Przekazywana jest też informacja o liczbie argumentów wiersza poleceń i ich wartościach. Funkcja działa w pętli. W każdym cyklu zwraca znak kolejnej napotkanej opcji lub znak `?`, jeśli nie rozpoznano opcji. Jeżeli zwróconą wartością jest `-1`, opcje w wierszu poleceń zostały wyczerpane.

Opcje niewymagające podania argumentów można łączyć. Argumenty mogą następować bezpośrednio po nazwie opcji lub mogą być od niej oddzielone odstępami.

**Przykład**. Jeżeli program przyjmuje trzy opcje wiersza poleceń, `-a`, `-b` i `-c`, gdzie `-b` wymaga przekazania argumentu, można go uruchomić w różny sposób:

```
prog -a -b wartosc    -c dane1 dane2 dane3
prog -ac -bwartosc -- dane1 dane2 dane3
prog -acbwartosc dane1 dane2 dane3
```

Jeżeli argument jest scalony z odpowiadającą mu opcją, część ciągu następująca po nazwie opcji jest traktowana jako argument opcji. W powyższym przykładzie zapis `-abcwartosc` oznacza, że użyto opcji `-a`, `-b` i `-c`, a `wartosc` jest argumentem opcji `-b`.

Funkcja `getopt` obsługuje cztery zmienne zewnętrzne:

- `optind`
  Indeks elementu tablicy wartości argumentów, `argv`, w którym napotkano pierwszy argument niebędący nazwą opcji

- `optarg`
  Łańcuch będący wartością argumentu opcji

- `opterr`
  Domyślnie `getopt` drukuje komunikat diagnostyczny po napotkaniu niepoprawnej nazwy opcji. Aby zmienić to ustawienie, należy nadać zmiennej `opterr` wartość zero

- `optopt`
  Znak (litera) reprezentujący opcję wiersza poleceń

Oto fragment kodu w języku C, który ilustruje, jak `getopt` może przetworzyć argumenty wiersza poleceń przeznaczone dla skryptu awk:

```
int main(int argc, char *argv[])
{
  ...
  opterr = 0;
  while ((c = getopt(argc, argv, "v:f:F:W:")) != -1)
  {
    switch (c) {
    case 'f':    /* plik */
      ...
      break;
    case 'F':    /* separator pol */
      ...
      break;
    case 'v':    /* nadanie wartosci zmiennej */
      ...
      break;
    case 'W':    /* rozszerzenie */
      ...
      break;
    case '?':
    default:
      usage();
      break;
    }
  }
  ...
}
```

W gawk zastosowano funkcję GNU `getopt_long` w celu przetwarzania opcji również w wersji dłuższej, stosowanej w narzędziach projektu GNU.

Oto wersja `getopt` zaimplementowana w języku awk. Na tym przykładzie widać jedną z podstawowych słabości awk, jaką jest mała wydajność przetwarzania pojedynczych znaków. Odczyt pojedynczych znaków wymaga wielokrotnego wywoływania funkcji `substr`.

```
# getopt.awk

# Zmienne zewnetrzne:
#    Optind -- indeks tablicy ARGV dla pierwszego
#              argumentu nie bedacego nazwa opcji
#    Optarg -- wartosc lancucha - argumentu
#              biezacej opcji
#    Opterr -- jezeli <> 0, drukuj wlasne
#              komunikaty diagnostyczne
#    Optopt -- biezaca jednoliterowa nazwa opcji

# Na wyjsciu:
#    -1     zakonczono przetwarzanie opcji
#    ?      nierozpoznana opcja
#    <c>    znak reprezentujacy biezaca opcje

# Dane prywatne:
#    _opti  -- indeks opcji w zapisie zwiezlym, np. -abc
```

Funkcja rozpoczyna się listą zmiennych globalnych.

Najpierw weryfikowana jest poprawność łańcucha opcji (parametr `options`). Jeżeli `options` ma długość zero, `getopt` zwraca `-1`:

```
function getopt(argc, argv, options, thisopt, i){
  if (length(options) == 0)    # brak opcji
    return -1
  if (argv[Optind] == "--"){
  # koniec przetwarzania opcji
    Optind++
    _opti = 0
    return -1
  }
  else if (argv[Optind] !~ /^-[^: \t\n\f\r\v\b]/){
    _opti = 0
    return -1
  }
}
```

Następnie trzeba znaleźć zakończenie ciągu opcji. Przejście po tablicy wartości argumentów odbywa się za pomocą zmiennej `Optind`. Jej wartość nie zmienia się pomiędzy kolejnymi wywołaniami `getopt`, ponieważ jest zmienną globalną.

Wyrażenie regularne `/^-[^: \t\n\f\r\v\b]/`, służy do wykrycia znaku `-`, po którym następuje dowolny znak różny od znaku białego i dwukropka. Jeżeli bieżący argument nie pasuje do tego wyrażenia regularnego, nie jest opcją i przetwarzanie opcji jest zakończone.

```
if (_opti == 0)
  _opti = 2
thisopt = substr(argv[Optind], _opti, 1)
Optopt = thisopt
i = index(options, thisopt)
if (i == 0)
{
  if (Opterr)
    printf("%c -- nieznana opcja\n",
  thisopt) > "/dev/stderr"
  if (_opti >= length(argv[Optind]))
  {
    Optind++
    _opti = 0
  }
  else
    _opti++
  return "?"
}
```

Zmienna `_opti` śledzi położenie bieżącego argumentu wiersza poleceń (`argv[Optind]`). Jeżeli wiele opcji występuje łącznie, np. `-abx`, należy je zwracać pojedynczo.

Jeżeli `_opti` jest równa zero, otrzymuje wartość 2 – indeksu następnego znaku, który należy sprawdzić (pomijamy znak `-` umieszczony na pozycji pierwszej). Wartość zmiennej `thisopt`, uzyskana za pomocą zmiennej `substr`, oznacza opcję. Wartość trafia do `OprOpt` w celu użycia w programie głównym.

Jeżeli wartości zmiennej `thisopt` nie ma w łańcuchu `options`, opcja jest uznana za nieprawidłową. Jeżeli `Opterr` jest różna od zera, drukowany jest komunikat diagnostyczny na standardowym wyjściu dla błędów, podobnie jak w systemowej funkcji `getopt` w języku C.

Jeżeli wartość zmiennej `_opti` jest większa niż lub równa długości bieżącego argumentu wiersza poleceń, należy przejść do następnego argumentu. Zatem zwiększa się `Optind` i `_opti` otrzymuje wartość zero. Jeżeli `_opti` jest mniejsza od długości bieżącego argumentu, `Optind` nie zmienia się, a `_opti` wzrasta o `1`.

Po napotkaniu ciągu, który nie jest opcją `getopt` zwraca znak zapytania `?`. Program główny może sprawdzać wartość zmiennej `Optopt`, np. w celu sprawdzenia, jaka litera jest niezgodna z nazwą opcji.

Oto kolejny fragment kodu:

```
if (substr(options, i + 1, 1) == ":")
{
  # wczytaj argument opcji
  if (length(substr(argv[Optind], _opti + 1)) > 0)
    Optarg = substr(argv[Optind], _opti + 1)
  else
    Optarg = argv[++Optind]
  _opti = 0
}
else
  Optarg = ""
```

Jeżeli opcja wymaga podania argumentu, po literze będącej nazwą opcji w łańcuchu `options` pojawia się dwukropek. Jeżeli w bieżącym argumencie wiersza poleceń, `argv[Optind]`, są jeszcze znaki do przetworzenia, pozostała część łańcucha jest umieszczona w zmiennej `Optarg`. Dotyczy to zapisu w rodzaju `-xWARTOSC`. W przeciwnym razie wczytany jest następny argument, np. `-x WARTOSC`. W obu przypadkach `_opti` otrzymuje wartość zero, ponieważ nie ma już znaków do sprawdzania w bieżącym argumencie.

Teraz kolejny fragment:

```
if (_opti == 0 || _opti >= length(argv[Optind]))
{
  Optind++
  _opti = 0
}
  else
    _opti++
  return thisopt
}
```

Jeżeli wartość `_opti` jest zero lub większa niż długość bieżącego argumentu, oznacza to, że dany element tablicy `argv` został już przetworzony i `Optind` zwięsza się o jeden, wskazując kolejny element w tablicy `argv`. W przeciwnym razie `_opti` wzrasta o jeden w celu przetworzenia następnej opcji przy następnym wywołaniu funkcji `getopt`.

```
BEGIN {
# Drukuj domyslne komunikaty diagnostyczne
  Opterr = 1
# Nie sprawdzaj ARGV[0]
  Optind = 1
# testuj program
  if (_getopt_test)
  {
    while ((_go_c = getopt(ARGC, ARGV, "ab:cd")) != -1)
      printf("c = <%c>, optarg = <%s>\n", _go_c, Optarg)
    printf("argumenty nie bedace opcjami:\n")
    for (; Optind < ARGC; Optind++)
      printf("\tARGV[%d] = <%s>\n", Optind, ARGV[Optind])
  }
}
```

Pozostała część bloku `BEGIN` jest prostym programem testowym. Oto wyniki dwoch jego uruchomień:

```
$ awk -f getopt.awk -v _getopt_test=1 -- -a -cbARG bax -x
-| c = <a>, optarg = <>
-| c = <c>, optarg = <>
-| c = <b>, optarg = <ARG>
-| argumenty nie bedace opcjami:
-|         ARGV[3] = <bax>
-|         ARGV[4] = <-x>

$ awk -f getopt.awk -v _getopt_test=1 -- -a -x -- xyz abc
-| c = <a>, optarg = <>
error--> x -- invalid option
-| c = <?>, optarg = <>
-| argumenty nie bedace opcjami:
-|         ARGV[4] = <xyz>
-|         ARGV[5] = <abc>
```

W obu przypadkach po napotkaniu sekwencji `--` wczytywanie argumentów jest zakończone.

## 4.26. Przykłady

1. Drukowanie plików z bieżącego katalogu o rozmiarze przekraczającym 1000 bajtów. Oto wynik polecenia `ls -l`:

```
$ ls -l
-rw-r--r--   1 root  root    1973 Mar 12 01:12 inetd.conf
-rw-r--r--   1 root  root    6912 Mar 22  2002 mime.conf
-rw-r--r--   1 root  root     386 Mar 12 01:12 profile
drwxr-xr-x   1 root  root       0 Mar 12 01:06 profile.d
-rw-r--r--   1 root  root   12306 Mar 12 01:31 termcap

ls -l | awk '/^-/ && $5 > 1000 { print $5, $9 }'
```

   Wyrażenie regularne `/^-/` wybiera wiersze, w których pierwszym znakiem jest `-`. Na wyjściu pojawi się informacja o pierwszym, drugim i piątym pliku z powyższego przykładu. Następnie obliczymy łączną wielkość tych plików i wydrukujemy wynik.

```
ls -l | awk '/^-/ && $5 > 1000 { suma+=$(NF-4) }\
  END{ print suma }'
```

2. W pliku danych

```
Jacek 1 20
Andrzej 2 15
Ania 3 17
```

   zapisane są kolejno imiona, liczba przepracowanych godzin i stawka za jedną godzinę. Obliczamy sumę, jaką należy zapłacić każdej osobie:

```
{ printf("%-12s %10d\n", $1, $2*$3) }
```

3. Odwrócenie kolejności słów w wierszu.

```
{ for (i=NF; i>0; i--)
  {
    printf("%s ", $i)
  }
  printf("\n")
}
```

4. Drukowanie wszystkich wierszy pliku w kolejności odwrotnej.

```
{ a[NR]=$0 }
END{
  for ( i=NR; i>=1; i=i-1)
  {
    print i,a[i]
  }
}
```

5. Drukowanie wszystkich słów znajdujących się w pliku tekstowym w kolejności odwrotnej.

```
{ for (i=1; i<=NF; i++) {
    w[NR,i]=$i
  }
}
END{ for (i=NR; i>=1; i--) {
   for (j=NF; j>=1; j--) {
     printf("%s ", w[i,j])
   }
   printf("\n")
 }
}
```

Aby wydrukować wszystkie znaki z pliku w kolejności odwrotnej, wystarczy zdefiniować `FS=""` (ciąg pusty). Wówczas każdy znak będzie odrębnym polem.

6. Plik zawiera dane o liczbie godzin przepracowanych przez pary osób.

```
20 Jacek Andrzej
15 Franek Andrzej
17 Jacek Pawel
10 Piotr Pawel
```

Dla każdej osoby obliczamy sumę przepracowanych godzin.

```
{ suma[$2] = suma[$2] + $1
  suma[$3] = suma[$3] + $1
}
END{ for (imie in suma) {
   print imie, suma[imie]
 }
}
```

## 4.27. Mapa Polski

Awk można wykorzystać również do prezentacji danych w postaci graficznej na terminalu tekstowym. Rozmiary okna tekstowego są często wystarczające do prezentacji poglądowej mapy lub wykresu. Jeżeli jakość obrazu wydaje się niewystarczająca, prezentowane schematy można zastosować także we współpracy z biblioteką graficzną.

```
#!/usr/bin/awk -f
# Program rysuje mape Polski

BEGIN { FS = ","
# wymiary ramki, wysokosc i szerokosc
  wys = 24; szer = 70
# polozenie poczatku ukladu wspolrzednych
  ox = 4; oy = 2
  liczba = "^[-+]?([0-9]+[.]?[0-9]*|[.][0-9]+)"\
           "([eE][-+]?[0-9]+)?$"
  kod = "^[0-9][0-9]-[0-9][0-9][0-9]$"
  nazwa = "^[A-Z][a-z]*"
#  miasto = ARGV[2]; ARGV[2] = ""
}

# oznaczenie osi x
$1 == "nazwax" { sub(/^ *nazwax */, ""); nazwax = $0; next
}

# wspolrzedne na osi x
$1 == "osx" && $2 == "wspolrzedne" {
  for (i = 3; i <= NF; i++) xwspolrz[++nb] = $i
  next
}
```

```
# wspolrzedne na osi y
$1 == "osy" && $2 == "wspolrzedne" {
  for (i = 3; i <= NF; i++) ywspolrz[++nl] = $i
  next
}

# xmin ymin xmax ymax
$1 == "zakres" {
  xmin = $2; ymin = $3; xmax = $4; ymax = $5
  next
}

# ramka wokol wykres: tak, nie
$1 == "ramka" { obram = $2 }

$1 == "znak" { znak = $2; next }

$1 == "znakmiasto" { znakm = $2; next }

$1 == "miasto" { mm++; miasto[mm] = $2 }

$1 == "kod" { kk++; kodpocztowy[kk] = $2 }

$1 == "wysokosc" { wys = $2; next }
$1 == "szerokosc" { szer = $2; next }
# para wspolrzednych
$1 ~ liczba && $2 ~ liczba {
# policz, ile jest par wspolrzednych
  nd++; x[nd] = $1; y[nd] = $2
  if ($3 == "")
  {
    ch[nd] = znak
  }
  else
  {
#   opcjonalny znak do oznaczania punktow na wykresie
    ch[nd] = $3
  }
  next
}
```

```
# obliczenie, ile jest par liczb w zbiorze danych
$1 ~ liczba && $2 !~ liczba {
  nd++
  x[nd] = nd; y[nd] = $1
  if ($2 == "")
  {
    ch[nd] = znak
  }
  else
  {
    ch[nd] = $2
  }
  next
}

# obliczenie, ile jest miast do zaznaczenia;
# sprawdzane jest dopasowanie wyrazenia regularnego
# (a) do nazwy miejscowosci
# (b) do kodu pocztowego
NF == 12 {
  for (i=1; i<=mm; i++)
  {
    if ($2 ~ miasto[i]) {
      nm++
      xm[nm] = $11; ym[nm] = $10
    }
  }

  for (i=1; i<=kk; i++)
  {
    if ($1 ~ kodpocztowy[i]) {
      nm++
      xm[nm] = $11; ym[nm] = $10
    }
  }
    next
}

END {
# rysowanie wykresu
```

```
# jezeli brak danych o zakresie liczb,
# oblicza zakres
  if (xmin == "") {
    xmin = xmax = x[1]; ymin = ymax = y[1]
    for (i = 2; i <= nd; i++) {
      if (x[i] < xmin) xmin = x[i]
      if (x[i] > xmax) xmax = x[i]
      if (y[i] < ymin) ymin = y[i]
      if (y[i] > ymax) ymax = y[i]
    }
  }
  if (obram == "tak") ramka()
  wspolrzedne(); nazwaosix(); dane();
  zaznaczmiasto(); rysuj()
}

function ramka() {  # obramowanie wykresu
# kolejno: dol, gora, lewa strona, prawa strona
  for (i = ox; i < szer; i++) plot(i, oy, "-")
  for (i = ox; i < szer; i++) plot(i, wys-1, "-")
  for (i = oy; i < wys; i++) plot(ox, i, "|")
  for (i = oy; i < wys; i++) plot(szer-1, i, "|")
}

# zaznaczenie wspolrzednych na osiach
function wspolrzedne( i) {
  for (i = 1; i <= nb; i++) {
    plot(skalujx(xwspolrz[i]), oy, "|")
    splot(skalujx(xwspolrz[i])-1, 1, xwspolrz[i])
  }
  for (i = 1; i <= nl; i++) {
    plot(ox, skalujy(ywspolrz[i]), "-")
    splot(0, skalujy(ywspolrz[i]), ywspolrz[i])
  }
}

# centrowanie opisu na osi c
function nazwaosix() {
  splot(int((szer + ox - length(nazwax))/2), 0, nazwax)
}
```

```
# rysowanie punktow na wykresie
function dane( i) {
  for (i = 1; i <= nd; i++)
    plot(skalujx(x[i]),skalujy(y[i]),
         ch[i]=="" ? "*" : ch[i])
}

# drukowanie wykresu na podstawie wartosci w tablicy
function rysuj( i, j) {
  for (i = wys-1; i >= 0; i--) {
    for (j = 0; j < szer; j++)
      printf((j,i) in array ? array[j,i] : " ")
      printf("\n")
  }
}

# skalowanie wartosci x
function skalujx(x) {
  return int((x-xmin)/(xmax-xmin)*(szer-1-ox)+ox+0.5)
}

# skalowanie wartosci y
function skalujy(y) {
  return int((y-ymin)/(ymax-ymin)*(wys-1-oy)+oy+0.5)
}

# zapisanie znaku c w tablicy
function plot(x, y, c) { array[x,y] = c }

# zapisanie lancucha w tablicy
function splot(x, y, s, i, n) {
  n = length(s)
  for (i = 0; i < n; i++)
    array[x+i, y] = substr(s, i+1, 1)
}

function zaznaczmiasto() {
  for (i = 1; i <= nm; i++)
    plot(skalujx(xm[i]),skalujy(ym[i]),znakm=="" ? "+" : znakm)
}
```

Oto format pliku danych. W pierwszym wierszu zapisane jest wyrażenie regularne dla kodu pocztowego. Na mapie zaznaczone zostaną wszystkie miejscowości, których kod pasuje do tego wyrażenia. Parametry `wysokosc` i `szerokosc` określają rozmiary mapy. W wierszu `zakres` określone są wartości minimalne i maksymalne dla szerokości i długości geograficznej, odpowiadające lewej, dolnej, prawej i górnej krawędzi mapy. Parametr `znak` określa znak, jakim rysowane są granice państwa. Do oznaczania miast służy wartość zmiennej `znakmiasto`.

Parametr `ramka` o wartości `tak` lub `nie` decyduje o tym, czy rysowana jest ramka wokół mapy. Dalej następuje sekwencja par współrzędnych geograficznych granic Polski. W ostatniej części pliku znajdują się dane miejscowości w następującym formacie:

kod pocztowy

nazwa miejscowości

dwuliterowy skrót oznaczający województwo

szerokość geograficzna w stopniach

szerokość geograficzna w minutach

długość geograficzna w stopniach

długość geograficzna w minutach

szerokość geograficzna po przeliczeniu minut na dziesiętne części stopni

długość geograficzna po przeliczeniu minut na dziesiętne części stopni

wysokość w metrach n.p.m.

Kolejność, w jakiej poszczególne parametry lub grupy danych są zapisane w pliku nie ma znaczenia. Oto przykład:

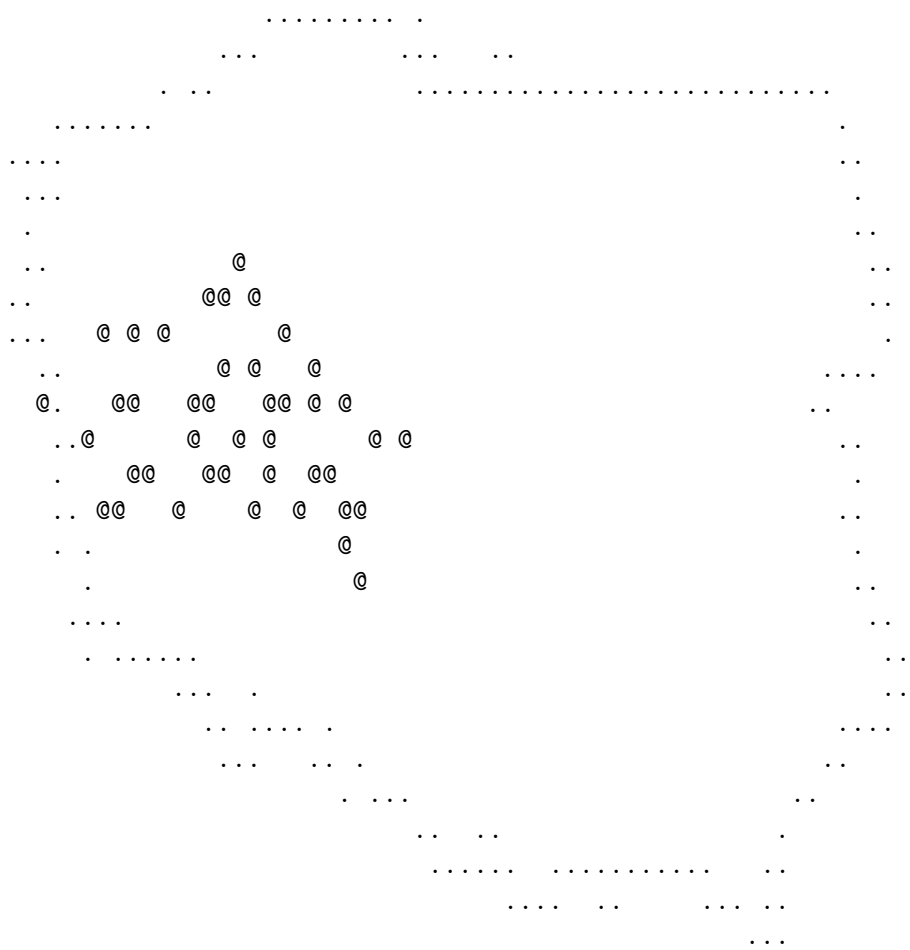
```
kod,6[0-9]-
wysokosc,40
szerokosc,70
zakres,14,49,25,55
znak,.
znakmiasto,@
ramka,nie
14.2267,53.9294
14.2131,53.9189
14.1903,53.9161
```

```
...
57-100,Strzelin,Do,50,47,N,17,4,E,50.7833,17.0667,174
58-100,Swidnica,Do,50,51,N,16,30,E,50.8500,16.5000,209
58-300,Walbrzych,Do,50,46,N,16,17,E,50.7667,16.2833,509
...
```

Uruchomienie skryptu:

```
$ awk -f plmapa.awk PL-dane.d
```

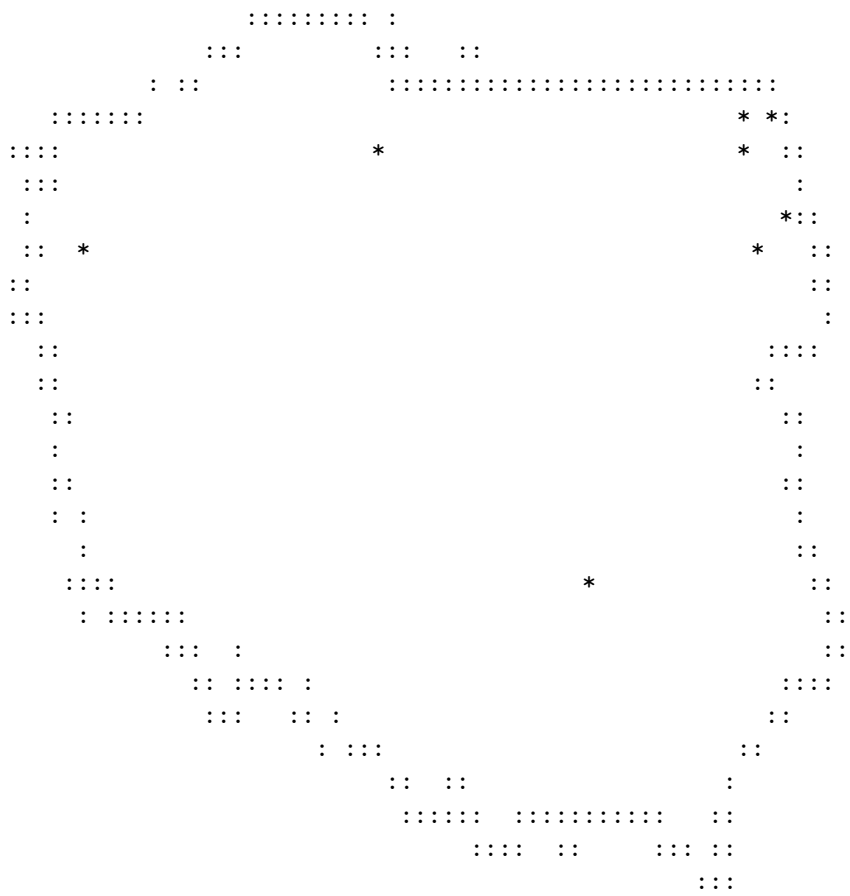
Wynik:



Oto inny przykład, w którym na mapie zaznaczane są miasta, których kod pocztowy rozpoczyna się sekwencją `15-` lub `16-` oraz te, których nazwa rozpoczyna się ciągiem `Star`:

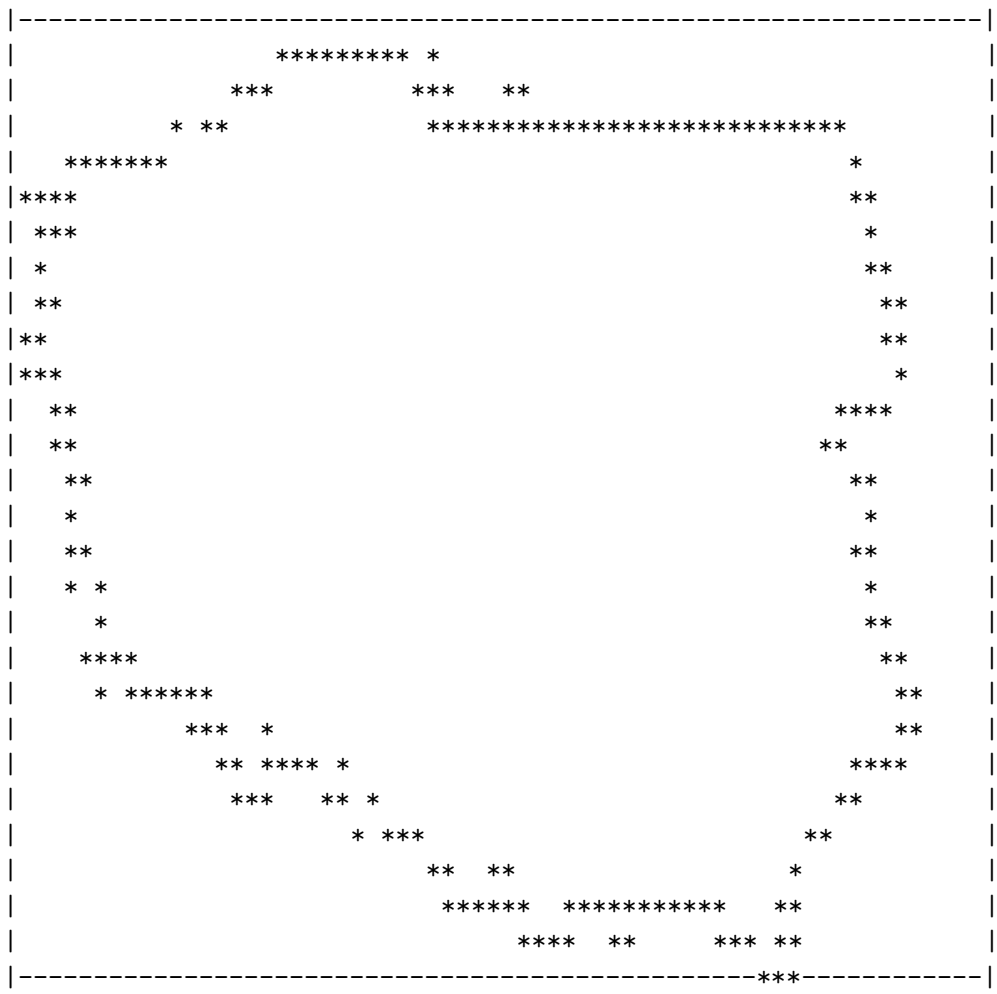
```
kod,1[56]-
miasto,Star
wysokosc,40
szerokosc,70
zakres,14,49,25,55
znak,:
znakmiasto,*
ramka,nie
...
```

Wynik:



I wreszcie mapa bez miejscowości, ale z ramką:

```
wysokosc,40
szerokosc,70
zakres,14,49,25,55
znak,*
ramka,tak
...
```



## 4.28. Wizualizacja obciążenia systemu

Podstawowe dane dotyczące obciążenia systemu są dostępne w pliku `/proc/loadavg`. Oto skrypt, w którym dane odczytywane z pliku `/proc/loadavg` są rysowane w postaci wykresu na terminalu tekstowym.

```
#!/bin/gawk -f
BEGIN{
  TYT = "Obciazenie komputera, srednia z ostatniej minuty"
  WYS = 20
  SZER = 60
# czas miedzy kolejnymi odczytami danych w sekundach
  dt = 5
#  MAXTAB[0] = 0
#  MAXTAB[1] = 1; MAXTAB[2] = 2;
# MAXTAB[3] = 4; MAXTAB[4] = 8
#  YMAX = MAXTAB[1]
   YMAX = 1
   XMAX = 300

# liczba kresek na osiach x i y
  XKR=5; YKR=4
# polozenie poczatku ukladu wspolrzednych
  OX=4; OY=2

#  system(sprintf("trap \" exit\" 2 15"))
  for (i=1; i<=SZER; i++) x[i] = 0
  n=0
  while (1)
  {
    n = n+1
    if (n>SZER) n=SZER
    getline < "/proc/loadavg"
    close("/proc/loadavg")
    for (nn=SZER; nn>=2; nn--) {
      x[nn] = x[nn-1]
      xx[nn]=xx[nn-1]
    }
# wartosci odczytane bezposrednio z /proc/loadavg
    x[1]=$1
    system("clear")
    skala()
    skaluj_x()
```

```
    czysc_a()
    ramka()
    osx()
    osy()
    punkty()
    rysuj()
    system(sprintf("sleep %d", dt))
#    if (system(sprintf("sleep %d", dt)) !=0 )
#    {
#      break
#    }
  }
}
  function skala()
  {
    najwieksza=1
    for (m=1; m<=SZER; m++) {
      k = 0
      while (1) {
        k++
        if (x[m]<=1) {
          ym=1; break
        }
        else
        {
          if (x[m]>=2^(k-1) && x[m]<2^k){
            ym=2^k; break
          }
        }
      }
      if (ym>=najwieksza) najwieksza = ym
    }
    YMAX = najwieksza
    for (j=1; j<=YKR; j++) napy[j]="" j*YMAX/YKR ""
    for (j=1; j<=XKR; j++) napx[j]="" j*XMAX/XKR ""
    return YMAX
  }

  function skaluj_x() {
# Tablica xx zawiera wartosci tablicy przeskalowane
# do skali, w jakiej sa przedstawiane na wykresie
    for (nn=0; nn<=SZER; nn++){
      xx[nn]=int(x[nn]*WYS/YMAX+0.5)
```

```
    }
  }

  function czysc_a() {
    for (nn=1; nn<=SZER; nn++) {
      for (i=1; i<=WYS; i++) {
        a[i,nn]=" "
      }
    }
  }

# obramowanie wykresu
  function ramka() {
    # lewa strona
    for (i = 0; i <= WYS; i++)  a[i,0]   = "|"
    # prawa strona
    for (i = 0; i <= WYS; i++)  a[i,SZER] = "|"
    # dol
    for (i = 0; i <= SZER; i++) a[0,i]   = "-"
    # gora
    for (i = 0; i <= SZER; i++) a[WYS,i]  = "-"
  }

  function osy() {
    for (j=1; j<=YKR; j++) a[j*WYS/YKR,0]="-"
  }

  function osx() {
    for (j=1; j<=XKR-1; j++) a[0,j*SZER/XKR]="|"
  }

  function napisyy() {
    for (j=1; j<=YKR; j++) {
      k = length(napy[j])
      for (p=0; p<k; p++)
      {
        z="" napy[j]
        i1=WYS-(YKR-j)*WYS/YKR+OY
        i2=OX-k+p
        b[i1,i2]=substr(z, p+1, 1)
      }
    }
  }
```

```
function napisyx() {
  for (j=1; j<=XKR; j++) {
    k = length(napx[j])
    for (p=0; p<k; p++)
    {
      z="" napx[j]
      i2=OX - (int(k/2)) + p + j*SZER/XKR
      b[1,i2]=substr(z, p+1, 1)
    }
  }
}

function tytul() {
  t=length(TYT)
  for (k=1; k<=t; k++)
    b[WYS+OY+1, SZER/2-int(t/2)+k-1]=sprintf("%s",
    substr(TYT, k, 1))
}

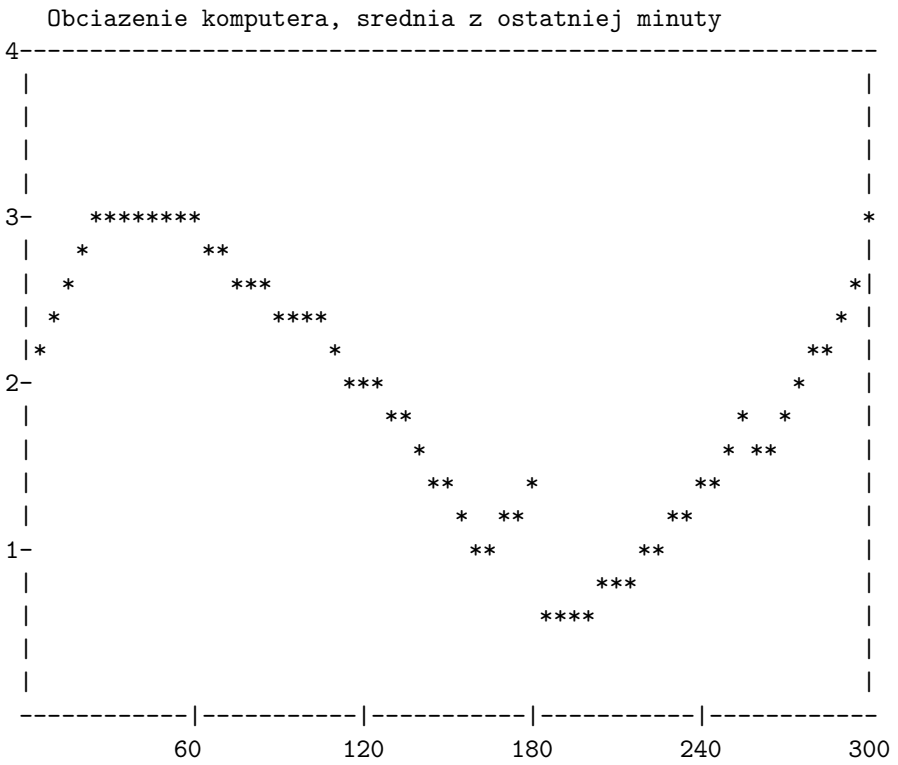
function punkty() {
  for (nn=1; nn<=SZER; nn++) {
    for (i=1; i<=WYS; i++) {
      if (i==xx[nn]) a[i,nn]="*"
    }
  }
}

function rysuj() {
  for (i=0; i<=WYS+OY+1; i++)
    for (nn=0; nn<=SZER+OX+2; nn++) b[i,nn]=" "
  tytul()
  napisyy()
  napisyx()
  for (i=0; i<=WYS; i++)
    for (nn=0; nn<=SZER; nn++) b[i+OY,nn+OX]=a[i,nn]
  for (i=WYS+OY+1; i>=0; i--) {
    for (nn=0;nn<=SZER+OX+2;nn++) { printf("%s",b[i,nn]) }
    printf("\n")
  }
  printf("\n")
}
```

Wykres jest skalowany automatycznie. W każdej iteracji sprawdzana jest

maksymalna wartość na wykresie. Maksymalna wartość na osi pionowej jest odpowiednią potęgą liczby 2. Jeżeli wykres przekracza dotychczasowy zakres na osi pionowej, następuje podwojenie maksymalnej wartości na tej osi. Jeżeli wykres spada poniżej połowy dostępnej skali, maksimum na osi y otrzymuje wartość dwukrotnie mniejszą.

Oto przykładowy wykres.




## 4.29. Obrót tekstu

W tym podrozdziale prezentowany jest obrót dowolnego tekstu, zapisanego w pliku. Kąt obrotu jest dodatnią lub ujemną wielokrotnością 90 stopni.

```
# Znaczenie zmiennych:
# max - ograniczenie na maksymalna
#       liczbe znakow w wierszu
# nfmax - liczba znakow w najdluzszym wierszu

BEGIN{ FS = ""; max = 72
```

```
  obrot = ARGV[ARGC-1] % 360
  if (obrot < 0)
  { obrot = obrot + 360 }
  ARGV[ARGC-1] = ""
}
# tnij, jezeli wiersz jest zbyt dlugi
{ if (NF > max) NF = max
  if (nfmax < NF) nfmax = NF
  for (i=1; i<=NF; i++)
  {
    a[NR,i] = $i
  }
  for (i=NF+1; i<=max; i++)
  {
    a[NR,i] = " "
  }
}
END{
  if (obrot == 90)
  { for (j=nfmax; j>=1; j--)
    { for (i=1; i<=NR; i++)
        printf("%s", a[i,j])
      printf("\n")
    }
  }
  if (obrot == 180)
  { for (i=NR; i>=1; i--)
    { for (j=nfmax; j>=1; j--)
        printf("%s", a[i,j])
      printf("\n")
    }
  }
  if (obrot == 270)
  { for (j=1; j<=nfmax; j++)
    { for (i=NR;  i>=1; i--)
        printf("%s", a[i,j])
      printf("\n")
    }
  }
}
```

Obrotem steruje następujący skrypt powłokowy:

```
# obrot.sh
if [ $1 = "-j" ]
then
  kat=$2
  shift; shift
  awk -f obrot.awk $@ $kat
else
  kat=$1
  shift
  for i in $@
  do
    awk -f obrot.awk $i $kat
  done
fi
```

Na wejściu można podać wiele plików. Ostatnim argumentem polecenia uruchamiającego skrypt jest kąt obrotu.

Jeżeli przed nazwami plików pojawi się opcja `-j`, pliki najpierw zostaną połączone w całość, a dopiero potem obrócone. W pozostałych przypadkach każdy z plików jest obracany oddzielnie. Wynik jest drukowany na standardowym wyjściu.

Aby przetestować działanie skryptu, utwórzmy najpierw dwa pliki zawierające wynik polecenia `banner`:

```
$ banner 12 > plik.12
$ banner 34 > plik.34
```

Zawartość pliku plik.12:

```
  XXX     XXX
    X    X   X
    X       X
    X     XX
    X    X
  XXXXX  XXXXX
```

Zawartość pliku plik.34:

```
  XXX       XX
 X   X     X X
     X    X  X
   XX    XXXXXX
 X   X       X
  XXX       XXX
```

```
$ ./obrot.sh 270 plik.12 plik.34
```

Dodatnia wartość kąta obrotu oznacza obrót w kierunku przeciwnym do ruchu wskazówek zegara.

Oto wynik:

```
X    X
X    X
XXXXXX
X
X


XX  X
X X  X
X X  X
X  X X
X   X



 X  X
X    X
X X  X
X X  X
 X XX



  X
  XX
  X X
X X  X
XXXXXX
X X
```

W powyższym przykładzie obrót tekstu w każdym z plików był wykonany oddzielnie. Na wyjściu pliki są drukowane jeden po drugim. Skrypt można uruchomić z opcją `-j`. Wówczas wszystkie przetwarzane pliki traktowane są jako jedna całość:

```
$ ./obrot.sh -j 270 plik.12 plik.34

    X  X
   X    X       X    X
   X X  X       X    X
   X X  X       XXXXXX
    X XX        X
                X


     X          XX  X
     XX         X X  X
     X X        X X  X
   X X  X       X  X X
   XXXXXX       X   X
   X X


$ ./obrot.sh -j 90 plik.12 plik.34

                   X X
    X   X          XXXXXX
   X X  X          X  X X
   X  X X           X X
   X  X X             XX
    X  XX             X


        X
        X           XX X
   XXXXXX           X  X X
   X    X           X  X X
   X    X           X    X
                     X  X
```

Skrypt obsługuje też ujemne wartości kątów.

## 4.30. Tworzenie wykresów na terminalu graficznym

W tym podrozdziale przedstawiona jest komunikacja interpretera gawk z programem gnuplot, służącym do rysowania wykresów na terminalu graficznym lub zapisującym je do pliku. W tym celu wykorzystana jest obsługa koprocesu, dostępna w interpreterze gawk.

1. Następujący skrypt uruchamia program gnuplot i przekazuje do niego polecenia. Domyślnie wykres powstaje na ekranie.

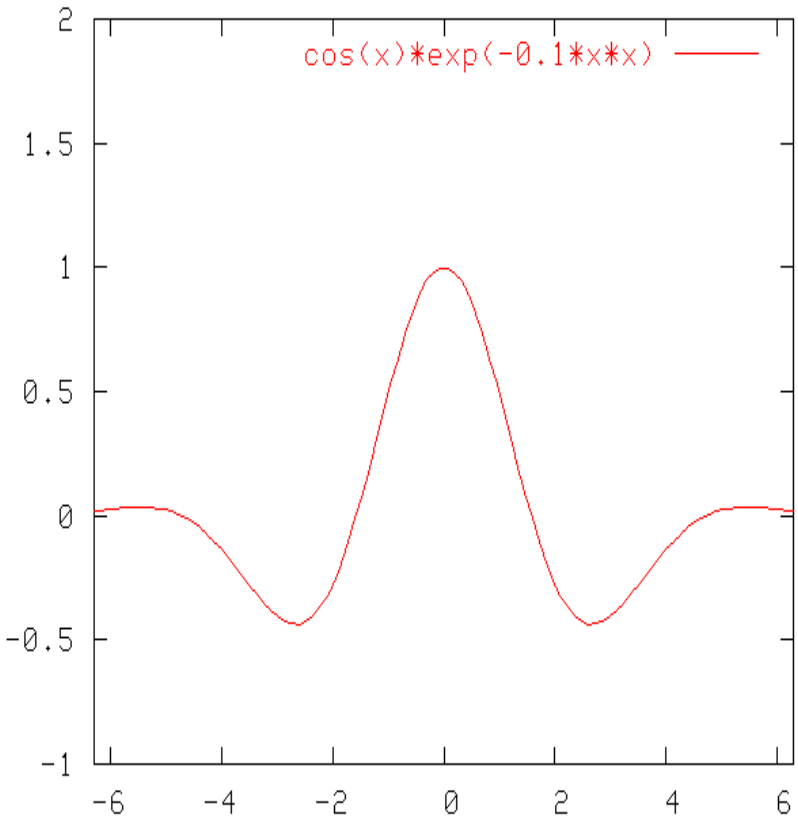
   ```
   #!/bin/awk -f
   BEGIN { GP="gnuplot"
     print "set xrange [-2*pi:2*pi]" |& GP
     print "plot cos(x)*exp(-0.1*x*x)" |& GP
     GP |& getline wykres
     close(GP)
   }
   ```

   Komunikacja skryptu z procesem programu gnuplot odbywa się za pomocą koprocesu. Polecenia programu gnuplot są kierowane do potoku instrukcją `print`.

   Pierwotnie koprocesy wprowadzono w powłoce Korna. Później zaimplementowano je także w interpreterze gawk.

2. W następnym skrypcie wykres jest zapisany w pliku graficznym `plik.png`.

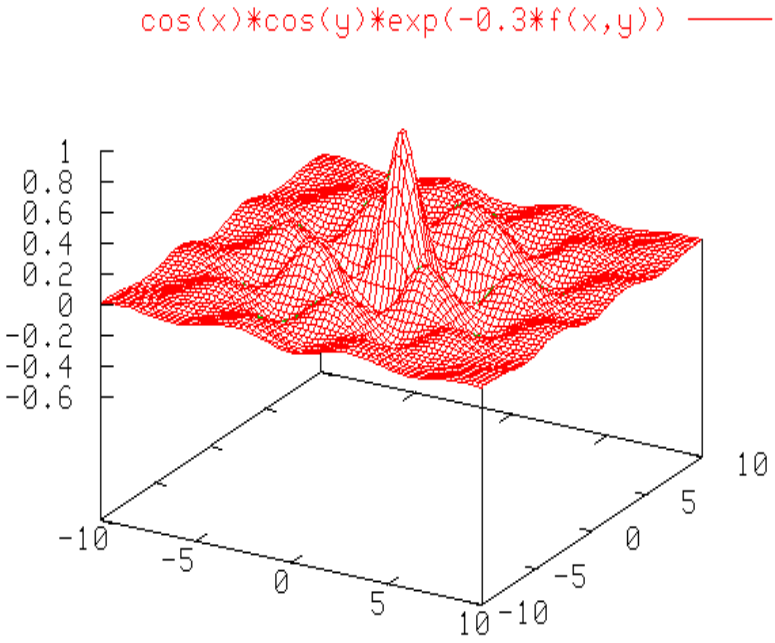
   ```
   #!/bin/awk -f
   BEGIN { GP="gnuplot"
     print "set term png medium color" |& GP
     print "set output 'plik.png' " |& GP
     print "set xrange [-2*pi:2*pi]" |& GP
     print "set yrange [-1:2]" |& GP
     print "plot cos(x)*exp(-0.1*x*x)" |& GP
     GP |& getline wykres
     close(GP)
   }
   ```




Rys. 4.1 Wykres funkcji $\cos(x)\exp(-x^2/10)$

Gnuplot umożliwia również tworzenie wykresów trójwymiarowych. Oto przykład:

3.
```
#!/bin/awk -f
BEGIN { GP="gnuplot"; teta = 65
  print "isosamples 70" |& GP
  print "set hidden" |& GP
  print "set view "teta", 30" |& GP
  print "f(x,y) = sqrt(x*x+y*y)" |& GP
  print "splot cos(x)*cos(y)*exp(-0.3*f(x,y))" |& GP
  GP |& getline wykres
  close(GP)
}
```




Rys. 4.2 Wykres funkcji $\cos(x)\cos(y)\exp(-0.3\sqrt{(x^2+y^2)})$

Kolejny przykład przedstawia całkowanie równania różniczkowego zwyczajnego $dy/dt = f(t,y)$, gdzie $f(t,y) = \cos(y)$. Dla ilustracji użyto najprostszej metody Eulera.

4.
```
#!/bin/gawk -f
BEGIN { tpocz=0; tkonc=10
  t=tpocz; dt=0.01
  y=1
  GP="gnuplot"
  print "set xrange["tpocz":"tkonc"]" |& GP
  print > "dane"
```

```
  metoda_Eulera()
  close("dane")
  print "plot 'dane' with lines" |& GP
  GP |& getline WYKRES
  close(GP,"to")
}
function metoda_Eulera(){
  while (t<tkonc)
  { y=y+f()*dt
    t=t+dt
    print t,y >> "dane"
  }
}

function f(){ return -cos(y) }
```

## 4.31. Pobieranie stron WWW

Możliwości tworzenia koprocesów w gawk można wykorzystać m.in. do komunikacji w sieci WWW. Ponieważ HTTP jest protokołem tekstowym, polecenia HTTP mogą być wysyłane do serwera i przekazywane w odwrotnym kierunku za pomocą programu telnet.

1. ```
   #!/usr/bin/gawk -f
   # Zobacz RFC 1945 - opis protokolu HTTP 1.0
   #        RFC 2616 - opis protokolu HTTP 1.1
   # Skrypt pobiera zawartosc strony internetowej
   BEGIN { RS = ORS = "\r\n"
     adres="main.amu.edu.pl"
     Telnet = "/usr/bin/telnet " adres " 80"
   # wersja HTTP 0.9
     print "GET /"   |& Telnet
   # print "GET / HTTP/1.0" |& Telnet
   # print "GET http://main.amu.edu.pl/:80/~lsb/index.html
   # HTTP/1.1" |& Telnet
   # while ((Telnet |& getline) !~ /html/)
   #       { print "----" $0}
     while ((Telnet |& getline) > 0)
       print $0
     close(Telnet)
   }
   ```
   ```

Oto odpowiedź serwera `main.amu.edu.pl` na powyższy skrypt:

```
Trying 150.254.65.7...
Connected to main.amu.edu.pl.
Escape character is '^]'.
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>302 Found</TITLE>
</HEAD><BODY>
<H1>Found</H1>
The document has moved
  <A HREF="http://www.amu.edu.pl">here</A>.<P>
<HR>
<ADDRESS>Apache/1.3.29 Server at
  main.amu.edu.pl Port 80</ADDRESS>
</BODY></HTML>
```

Oczywiście, adres serwera, port i wersję protokołu HTTP można przekazywać do skryptu jako parametr. To drobne ćwiczenie autor pozostawia czytelnikowi.

2. Do niektórych celów pobieranie całej strony WWW jest zbyteczne. Czasem wystarczy informacja zawarta w nagłówku strony:

```
#!/usr/bin/gawk -f
BEGIN {
  RS = ORS = "\r\n"
  adres="www.apacheweek.com"
  Telnet = "/usr/bin/telnet " adres " 80"
  print "HEAD / HTTP/1.0 \n\n" |& Telnet
# printf("GET / HTTP/1.0\n\n") |& Telnet
# print "GET http://www.amu.edu.pl/:80/~lsb/index.html
#   HTTP/1.1" |& Telnet
# while ((Telnet |& getline) !~ /html/)
#       { print "----" $0}
  while ((Telnet |& getline) > 0)
    print $0
  close(Telnet)
  exit
}
```

Odpowiedź:

```
Trying 66.187.229.34...
Connected to apacheweek.com.
Escape character is '^]'.
HTTP/1.1 200 OK
Date: Fri, 28 Jan 2005 01:37:50 GMT
Server: Stronghold/4.0-RHL Apache/1.3.22
Last-Modified: Wed, 12 Jan 2005 23:14:17 GMT
ETag: "277d7-401b-41e5af49"
Accept-Ranges: bytes
Content-Length: 16411
Connection: close
Content-Type: text/html
```

3. ```
   #!/usr/bin/gawk -f
   BEGIN {
     RS = ORS = "\r\n"
     adres="www.apacheweek.com"
     Telnet = "/usr/bin/telnet " adres " 80"
     print "HEAD / HTTP/1.1 \n\n" |& Telnet
     while ((Telnet |& getline) > 0)
       print $0
     close(Telnet)
   }
   ```

   Wynik:

   ```
   Trying 66.187.229.34...
   Connected to apacheweek.com.
   Escape character is '^]'.
   HTTP/1.1 400 Bad Request
   Date: Fri, 28 Jan 2005 01:38:39 GMT
   Server: Stronghold/4.0-RHL Apache/1.3.22
   Connection: close
   Content-Type: text/html; charset=iso-8859-1
   ```

4. W wersji HTTP 1.1 dostępna jest instrukcja `printf`:

   ```
   #!/usr/bin/gawk -f
   BEGIN {
     RS = ORS = "\r\n"
   ```
   ```

```
  adres="www.apache.org"
  Telnet = "/usr/bin/telnet " adres " 80"
  printf("HEAD / HTTP/1.1\n") |& Telnet
# Nowosc w wersji HTTP 1.1
  printf("Host: www.apache.org\n\n") |& Telnet
  while ((Telnet |& getline) > 0)
    print $0
  close(Telnet)
}
```

Wynik:

```
Trying 209.237.227.195...
Connected to www.apache.org.
Escape character is '^]'.
HTTP/1.1 200 OK
Date: Fri, 28 Jan 2005 01:49:33 GMT
Server: Apache/2.0.52 (Unix)
Last-Modified: Sat, 08 Jan 2005 01:10:45 GMT
ETag: "2da8030-2d98-1332f40"
Accept-Ranges: bytes
Content-Length: 11672
Cache-Control: max-age=86400
Expires: Sat, 29 Jan 2005 01:49:33 GMT
Content-Type: text/html; charset=ISO-8859-1
```

5. Pobranie nagłówka strony głównej serwera `www.amu.edu.pl` według składni HTTP 1.1:

```
#!/usr/bin/gawk -f
BEGIN {
  RS = ORS = "\r\n"
  adres="www.amu.edu.pl"
  Telnet = "/usr/bin/telnet " adres " 80"
  printf("HEAD / HTTP/1.1\n") |& Telnet
  printf("Host: www.amu.edu.pl\n\n") |& Telnet
  while ((Telnet |& getline) > 0)
    print $0
  close(Telnet)
}
```

Wynik:

```
Trying 150.254.65.31...
Connected to zp.amu.edu.pl.
Escape character is '^]'.
HTTP/1.1 200 OK
Date: Fri, 28 Jan 2005 01:53:25 GMT
Server: Apache/1.3.33 (Debian GNU/Linux) PHP/4.3.10-2
X-Powered-By: PHP/4.3.10-2
Last-Modified: Fri, 28 Jan 2005 01:53:25 GMT
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache,
  must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Set-Cookie: PHPSESSID=edb4cb88745c74794f909b14a278a678;
  path=/
Set-Cookie: testCookie=1
Content-Type: text/html; charset=iso-8859-2
```

6. W tym przykładzie sprawdzamy w odpowiedzi serwera, czy strona została przeniesiona pod nowy adres. Jeżeli pojawi się komunikat `301 Moved` i słowo `Location` w tym samym wierszu, ciąg znaków następujący po `Location` wskazuje nowe położenie strony.

```
#!/usr/bin/gawk -f
BEGIN {
 RS = ORS = "\r\n"
 adres="www.rp.pl"
 Telnet = "/usr/bin/telnet " adres " 80"
 printf("HEAD / HTTP/1.1\n") |& Telnet
 printf("Host: www.rp.pl\n\n") |& Telnet
 Telnet |& getline
  if ($0 ~ /301 Moved/ && $0 ~ /Location:/)
  {
   for (i=1; i<=NF; i++) {
    if ($i == "Location:") {
     nowy_adres = $(i+1)
    }
   }
   close(Telnet)
   Telnet1 = "/usr/bin/telnet " \
```

```
                nowy_adres " 80"
    printf("GET " nowy_adres " HTTP/1.1\n")\
          |& Telnet1
    printf("Host: " nowy_adres "\n\n")\
          |& Telnet1
    while ((Telnet1 |& getline) > 0) print $0
    close(Telnet1)
   }
  close(Telnet)
 }
```

```
Trying 195.8.128.104...
Connected to www.rp.pl.
Escape character is '^]'.
HTTP/1.1 301 Moved Permanently
Date: Fri, 28 Jan 2005 01:59:55 GMT
Server: Apache
Location: http://www.rzeczpospolita.pl/index.html
Content-Length: 247
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>301 Moved Permanently</title>
</head><body>
<h1>Moved Permanently</h1>
<p>The document has moved
<a href="http://www.rzeczpospolita.pl/index.html">
  here</a>.</p>
</body></html>
```

7. Pobranie całej strony internetowej. Protokół HTTP 1.1.

```
#!/usr/bin/gawk -f
# skrypt-4.31.7.awk
# Pobiera strone internetowa
# W tym przykladzie jest to www.apache.org
BEGIN {
  RS = ORS = "\r\n"
  adres="www.apache.org"
```

```
  Telnet = "/usr/bin/telnet " adres " 80"
  printf("GET / HTTP/1.1\n") |& Telnet
  printf("Host: " adres "\n\n") |& Telnet
  while(( Telnet |& getline ) > 0)
    print $0
  close(Telnet)
}
```

8. Przykład automatyzacji pobierania danych z Internetu. Skrypt awk można uruchamiać cyklicznie w regularnych odstępach czasu i dzięki temu na bieżąco śledzić dane zależne od czasu bez uruchamiania dodatkowych programów.

   Następujący skrypt pobiera ze strony Giełdy Papierów Wartościowych w Warszawie informację o aktualnej wartości indeksu giełdowego WIG20.

```
#!/usr/bin/gawk -f
BEGIN {
#  RS = ORS = "\r\n"
  xwig = "WIG20"; adres="www.gpw.com.pl"
  plik="//wyniki/indeksy.asp?lang=PL"
  Telnet = "/usr/bin/telnet " adres " 80"
  printf("GET " plik " HTTP/1.1\n") |& Telnet
  printf("Host: " adres "\n\n") |& Telnet
  while(( Telnet |& getline ) > 0) {
    if ($0 ~ />WIG20<\/A><\/TD>/) {
      Telnet |& getline
      p = split($0, a, /<|>/)
      Telnet |& getline
      split($0, b, /<|>/)
      gsub(" ", "", b[11])
      sub(",", ".", b[11])
      print "Indeks " xwig " mial o godz.\
            " a[p-2] " wartosc " b[11]
      break
    }
  }
  close(Telnet)
}
```

Wynik:

```
Indeks WIG20 mial o godz. 16:45:00 wartosc 1863.92
```

## 4.32. Filtrowanie adresów internetowych ze stron WWW

Czasem trzeba odfiltrować odsyłacze ze strony WWW. Można to zrobić za pomocą odpowiedniego wyrażenia regularnego. W następującym przykładzie rekordy są rozdzielone adresami stron WWW. Po napotkaniu adresu WWW bieżący rekord jest zakończony i drukowana jest zawartość zmiennej `RT`, terminatora rekordu, którym jest właśnie odsyłacz do strony WWW.

```
#!/usr/bin/gawk -f
# filtr_adresow.awk
BEGIN { RS = "http://[#%&\\+\\-\\./0-9\\:;
              \\?A-Z_a-z\\~]*" }
RT != "" { print RT }}
```

Adresy ze strony WWW zapisanej na dysku lokalnego komputera można oddzielić następująco:

```
$ cat plik.html | filtr_adresow.awk
```

Jeżeli adresy mają być filtrowane podczas pobierania strony i zapisywane do pliku, można wykorzystać skrypt numer 7 z poprzedniego podrozdziału:

```
$ ./skrypt-4.31.7.awk | ./filtr_adresow.awk > plik_adresow
```

Wyrażenie regularne oznaczające ogólną postać URL nie jest dostatecznie precyzyjne. Dokładna postać tego wyrażenia jest bardziej złożona. Jednak, powyższa wartość zmiennej `RS` w większości przypadków w praktyce wystarcza.

Wyrażenie to nie uwzględnia wewnętrznych odsyłaczy dokumentu HTML. Nie uwzględnia też odsyłaczy do usług ftp, telnet, news, mailto i innych. W razie potrzeby nietrudno je dopisać do zmiennej `RS`.

```
#!/usr/bin/gawk -f
BEGIN {
# Jezeli adres strony i nazwa pliku sa przekazywane
# jako argumenty skryptu, usun znak komentarza z trzech
# kolejnych wierszy i zamien w komentarz dwa nastepne
# wiersze
# adres = ARGV[1]; plik = ARGV[2]
# ARGV[1] = ""; ARGV[2] = ""
#-------------------------------------------------
  adres = "www.w3.org"
  plik = adres "/MarkUp/"
#-------------------------------------------------
  Telnet = "/usr/bin/telnet " adres " 80"
  printf("GET http://" plik " HTTP/1.1\n") |& Telnet
  printf("Host: " adres "\n\n") |& Telnet
  while(( Telnet |& getline ) > 0) {
    sub(/^.*="http/, "http", $0)
    sub(/".*$/, "", $0)
    if ($0 ~ /http:\/\//) print $0
  }
  close(Telnet)
}
```

Jeżeli adresy i nazwy plików są przekazywane jako argumenty, polecenie uruchomienia skryptu ma następującą postać:

```
$ ./geturl2.awk www.w3.org /MarkUp/ 1>www.w3.org\_\_MarkUp,
```

Aby oddzielić uzyskane adresy od ewentualnych komunikatów diagnostycznych, należy dopisać w tym poleceniu `2>bledy`.

# 5. Skrypty powłokowe

## 5.1. Wstęp

Skrypt jest programem składającym się z poleceń interpretowanych przez powłokę, w której jest uruchomiony. Pierwszy wiersz musi rozpoczynać się sekwencją `#!`, po której należy podać pełną ścieżkę do pliku wykonywalnego powłoki lub innego narzędzia interpretującego skrypt, na przykład:

```
#!/bin/sh
```

Interpreterem skryptu może być dowolna powłoka Uniksa (`/bin/sh`, `/bin/bash`, `/bin/csh`, `/bin/ksh`), intepretery innych języków skryptowych (np. `/usr/bin/perl`, `/usr/bin/python`, `/usr/bin/tcl`, `/usr/bin/awk -f`, `/usr/bin/sed -f`) lub dowolny inny plik wykonywalny (np. `/usr/bin/cat`).

Skrypt można uruchomić poleceniem `sh skrypt.sh` lub `./skrypt.sh`. Przed uruchomieniem skryptu drugim sposobem trzeba nadać uprawnienia wykonywania:

```
$ chmod 700 skrypt.sh
```

lub

```
$ chmod +x skrypt.sh
```

Wszędzie poza pierwszym wierszem znak `#` oznacza początek komentarza. Materiał bieżącego rozdziału dotyczy zmodyfikowanej powłoki Bourne'a `bash`.

Instrukcja przypisania wartości zmiennej ma następującą postać:

```
$ x=wartosc
```

Wartość zmiennej można umieścić w pojedynczym lub podwójnym cudzysłowie:

```
$ x='wartosc'
$ y='inna wartosc'
$ z="jeszcze inna wartosc"
```

Dzięki temu wartość zmiennej może zawierać spacje. Jeżeli prawa strona równania jest ujęta w cudzysłów ` `, jest ona najpierw interpretowana przez powłokę, po czym otrzymana wartość jest przypisana zmiennej, na przykład:

```
$ x=`date +%F`
$ echo $x
2005-02-18
```

Zapis `$x` oznacza wartość zmiennej `x`. Innym sposobem odwoływania się do wartości zmiennej jest notacja `${nazwazmiennej}`.

Umieszczenie wywołania wartości zmiennej w cudzysłowie nie przeszkadza przy podstawianiu wartości. Jest to nazywane cytowaniem częściowym (ang. *partial quoting* lub *weak quoting*), w odróżnieniu od cytowania pełnego (ang. *full quoting* lub *strong quoting*), jakim jest użycie cudzysłowu pojedynczego, `'$'`. Znaczenie tej notacji ilustruje następujący skrypt:

```
zmienna="Dzien dobry"
echo $zmienna
echo "$zmienna"
echo '$zmienna'
echo \$zmienna

echo Wpisz dzisiejsza date
# Instrukcja read czyta tekst ze standardowego wejscia
read zmienna
echo Dzis jest "$zmienna"
```

Oto wynik działania programu:

```
Dzien dobry
Dzien dobry
$zmienna
$zmienna
Wpisz dzisiejsza date
18 lutego 2005
Dzis jest 18 lutego 2005
```

Niektóre zmienne są definiowane domyślnie przy każdym uruchomieniu skryptu:

| Zmienna | Znaczenie |
|---|---|
| `$0` | Nazwa skryptu powłokowego |
| `$#` | Liczba parametrów przekazanych do skryptu |
| `$$` | Identyfikator procesu (PID, ang. *process identifier*) skryptu |
| `$!` | Numer procesu ostatniego polecenia wykonanego w tle |
| `$?` | Stan końcowy (ang. *exit code*) ostatniego wykonanego polecenia |
| `$1`, `$2`, ... | Wartości poszczególnych parametrów |
| `$*` | Lista wszystkich parametrów, traktowana jako jedna zmienna, poszczególne parametry oddzielone są znakiem separatora |
| `$@` | Lista wszystkich parametrów, traktowana jako jedna zmienna, poszczególne parametry nie są od siebie oddzielone |

## 5.2. Podstawianie wartości zmiennych

Manipulacje związane z podstawianiem wartości zmiennych ułatwia nawias klamrowy. Dzięki temu można łączyć wartości zmiennych z łańcuchami znaków:

```
a=Pan
b=${a}" Tadeusz"
echo $b
```

Wynik skryptu:

```
Pan Tadeusz
```

Nawiasy klamrowe są potrzebne m.in. wtedy, kiedy zmienne pozycyjne mają więcej niż jedną cyfrę.

Oto inne czynności wykonywane na zmiennych za pomocą nawiasów klamrowych:

- `${zmienna:-wartosc}`
  Wynikiem podstawienia jest wartość zmiennej lub `wartosc`, jeżeli `zmienna` nie była dotąd zdefiniowana

- `${zmienna:=slowo}`
  Jeżeli `zmienna` nie była dotąd zdefiniowana, otrzymuje wartość; stosowane do podstawiaia wartości domyślnych

- `${zmienna:?[slowo]}`
  Jeżeli `zmienna` nie ma wartości lub ma wartość null, na standardowym wyjściu komunikatów diagnostycznych drukowane jest rozwinięcie ciągu `slowo` (lub komunikat informujący o braku wartości *zmiennej*) i powłoka zostaje zamknięta (jeśli była uruchomiona w trybie wsadowym) z niezerowym kodem wynikowym; w przeciwnym razie w miejsce wyrażenia podstawiona jest wartość zmiennej

- `${zmienna:+slowo}`
  Jeżeli `zmienna` nie ma wartości lub ma wartość null, wynikiem wyrażenia jest null; w przeciwnym razie podstawione jest rozwinięcie ciągu `slowo`

- `${#zmienna}`
  Długość łańcucha `$zmienna`

- `${zmienna%slowo}`
  Wartość zmiennej po usunięciu najkrótszej części łańcucha `$zmienna` pasującej do ciągu `slowo`, licząc od końca

- `${zmienna%%slowo}`
  Wartość zmiennej po usunięciu najdłuższej części łańcucha `$zmienna` pasującej do ciągu `slowo`, licząc od końca

- `${zmienna#slowo}`
  Wartość zmiennej po usunięciu najkrótszej części łańcucha `$zmienna` pasującej do ciągu `slowo`, licząc od początku

- `${zmienna##slowo}`
  Wartość zmiennej po usunięciu najdłuższej części łańcucha `$zmienna` pasującej do ciągu `slowo`, licząc od początku

**Uwaga**: w instrukcji przypisania wartości zmiennej nie ma spacji.

**Przykłady**

- W tym przykładzie polecenie `ls` jest wykonywane jedynie wtedy, kiedy zmienna `x` nie jest zdefiniowana lub jej wartość anulowano poleceniem `unset`:

  ```
  ${x:-$(ls)}
  ```

- ```
  x=plik.c
  echo ${x%.c}.o
  ```

  Wynik: `plik.o`

- ```
  x=kat1/kat2/kat3
  echo ${x%%/*}
  ```

  Wynik: `kat1`

- ```
  x=$HOME/kat1/kat2
  echo ${x#$HOME}
  ```

  Wynik: `/kat1/kat2`

- ```
  x=/kat1/kat2/kat3
  echo ${x##*/}
  ```

  Wynik: `kat3`

Sposób interpretacji znaków specjalnych zależy od położenia cudzysłowu. Na przykład, w wyrażeniu `"${x#*}"` gwiazdka jest metaznakiem, a w wyrażeniu `${x#"*"}` nie ma znaczenia specjalnego.

## 5.3. Polecenie `test`

Polecenie `test warunek` zwraca kod stanu końcowego `0`, jeśli sprawdzane wyrażenie jest prawdziwe, lub niezerowy kod stanu, jeśli wyrażenie jest nieprawdziwe, przy czym prawdą w tym kontekście jest również niepusty wynik dowolnego polecenia Uniksa.

**Przykład**. Następujący skrypt sprawdza, czy plik istnieje i drukuje odpowiedni komunikat.

```
if test -f plik
  then echo "plik istnieje"
  else echo "plik nie istnieje"
fi
```

Równoważną formą instrukcji `test` jest zapis `[ ]`, na przykład `[ -f plik ]`.

Oto pełna lista warunków. Wyrażenia liczbowe dotyczą liczb całkowitych:

- `( wyrazenie )`
  `wyrazenie` jest prawdziwe
- `! wyrazenie`
  `wyrazenie` jest fałszywe
- `w1 -a w2`
  oba wyrażenia są prawdziwe
- `w1 -o w2`
  prawdziwe jest `w1` lub `w2`
- `-n lancuch`
  `lancuch` ma długość większą od zera
- `-z lancuch`
  `lancuch` ma długość zero
- `l1 = l2`
  łańcuchy są równe
- `l1 != l2`
  łańcuchy się różnią
- `n1 -eq n2`
  liczby są równe
- `n1 -ge n2`
  liczba `n1` jest większa lub równa `n2`
- `n1 -gt n2`
  liczba `n1` jest większa od `n2`
- `n1 -le n2`
  liczba `n1` jest mniejsza lub równa `n2`
- `n1 -lt n2`
  liczba `n1` jest mniejsza niż `n2`
- `n1 -ne n2`
  liczba `n1` nie jest równa `n2`

- `plik1 -ef plik2`
  `plik1` i `plik2` mają ten sam numer urządzenia
  i ten sam numer i-węzła
- `plik1 -nt plik2`
  `plik1` ma późniejszą datę ostatniej modyfikacji niż `plik2`
- `plik1 -ot plik2`
  `plik1` jest starszy niż `plik2`
- `-b plik`
  plik istnieje i jest specjalnym plikiem blokowym
- `-c plik`
  plik istnieje i jest specjalnym plikiem znakowym
- `-d plik`
  plik istnieje i jest katalogiem
- `-e plik`
  plik istnieje
- `-f plik`
  plik istnieje i jest zwykłym plikiem
- `-g plik`
  plik istnieje i ma ustawiony bit *set-group-ID*
- `-G plik`
  plik istnieje, a jego grupa ma efektywny GID
- `-k plik`
  plik istnieje i ma ustawiony sticky bit
- `-L plik`
  plik istnieje i jest dowiązaniem symbolicznym
- `-O plik`
  plik istnieje, a jego właściciel ma efektywny UID
- `-p plik`
  plik istnieje i jest potokiem z przydzieloną nazwą
- `-r plik`
  plik istnieje i ma uprawnienia odczytu

- `-s plik`
  plik istnieje i ma rozmiar większy od zera
- `-S plik`
  plik istnieje i jest gniazdem (ang. *socket*)
- `-t [FD]`
  deskryptor pliku `FD` (domyślnie stdout)
  jest otwarty na terminalu
- `-u plik`
  plik istnieje i ma ustawiony bit *set-user-ID*
- `-w plik`
  plik istnieje i ma uprawnienia zapisu
- `-x plik`
  plik istnieje i ma uprawnienia wykonywania

## 5.4. Instrukcje sterujące

Szczegóły składni niektórych instrukcji w różnych powłokach mogą być różne. Na przykład, w powłoce C instrukcje `if` i `then` muszą znajdować się w tym samym wierszu.

- ```
  if [ warunek ]
  then
     polecenie1
     polecenie2
     ...
  else
     polecenie3
     polecenie4
     ...
  fi
  ```

  W tym przykładzie najpierw sprawdzany jest warunek, czy plik istnieje:

  ```
  if test -f abc
  then
      ls abc
  fi
  ```
  ```

Warunek można zapisać również z użyciem nawiasów `[]`:

```
if [ -f abc ]
then
   ...
fi

if [ -f abc ]; then
   ...
fi
```

- ```
  for zmienna in lista_wartosci
  do
     ...
  done
  ```

  W pętli `for` można pominąć listę wartości zmiennej. Wówczas domyślnie pętla jest wykonywana po elementach listy argumentów skryptu `$@`. Wartościami listy mogą być również nazwy plików i w takim przypadku można skorzystać ze znaków specjalnych (np. gwiazdki lub znaku zapytania – patrz przykład w dalszej części rozdziału). Elementy listy wartości można wypisać jawnie na początku pętli:

  ```
  for i in rower czlowiek samochod
  do
    ...
  done
  ```

- ```
  while warunek do
     ...
  done
  ```

  Instrukcje pętli `while` są wykonywane, dopóki warunek jest prawdziwy.

  W tym przykładzie wartość zmiennej a zwiększa się o 1 w każdej iteracji, dopóki a jest mniejsze lub równe 10:

  ```
  a=1
  while [ "$a" -le 10 ]
  do
     echo "a="$a
     a=$(($a+1))
  done
  ```

Czasem konieczne jest wykonywanie pętli nieskończonej `while`. W tym celu należy użyć warunku, który jest zawsze prawdziwy. Najprostszym wyrażeniem, które pełni taką rolę, jest dwukropek `:`.

```
while :
do
   instrukcje
done
```

**Uwaga**. Nie można przekierować wyjścia całej pętli `while`.

- ```
  until warunek
  do
     ...
  done
  ```

- Instrukcja `case` umożliwia wybór jednej z wielu możliwych wartości łańcucha.

```
case zmienna in
   wzorzec1) grupa_instrukcji_1;;
   wzorzec2) grupa_instrukcji_2;;
   ...
esac
```

Oto prosty przykład dialogu z użytkownikiem:

```
x="Czerwone kwiaty sa piekne"
y="Szkoda"
echo "Czy lubisz kolor czerwony? [T/N]"
read $kolor
case "$kolor" in
  "Tak") echo "$x";;
  "Nie") echo "$y";;
  "T")   echo "$x";;
  "N")   echo "$y";;
  *)     echo "Niezrozumiala odpowiedz";;
esac
```
```

Wyrażenie `*` pasuje do dowolnego łańcucha i służy do obsługi odpowiedzi użytkownika, które nie są zgodne z uwzględnionymi wcześniej wariantami. Zwykle umieszcza się je na końcu instrukcji `case` w celu objęcia nim wszystkich pozostałych możliwości.

Można ten przykład nieco ulepszyć, aby reakcja programu nie zależała od wielkości liter użytych przez użytkownika.

```
x="Czerwone kwiaty sa piekne"
y="Szkoda"
echo "Czy lubisz kolor czerwony? [T/N]"
read $kolor
case "$kolor" in
  "[Tt]") | "[Tt][Aa][Kk]") echo "$x";;
  "[Nn]") | "[Nn][Ii][Ee]") echo "$y";;
  *)     echo "Niezrozumiala odpowiedz";;
esac
```

Tym razem wielkość liter w odpowiedzi użytkownika nie odgrywa roli. Poprawnie zinterpretowana zostanie każda odpowiedź składająca się z jednej lub trzech liter, np. `t` lub `tAk`. Znak `|` oznacza alternatywę (lub). Zapis `[Tt]` oznacza literę `T` lub `t`.

## 5.5. Funkcje

Oto przykład funkcji dokonującej zamiany małych liter na wielkie:

```
male_na_wielkie()
{ echo $1 | tr 'abcdefghijklmnopqrstuvwxyz' \
    'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
}
```

Sposób użycia w skrypcie:

```
male_na_wielkie "dzien dobry"
```

lub

```
a="dzien dobry"
male_na_wielkie "$a"
```

Tę funkcję można zastosować na przykład w celu uproszczenia obsługi dialogu z użytkownikiem. Jeżeli odpowiedź użytkownika ma być obsłużona za pomocą instrukcji `case`, wystarczy najpierw uruchomić powyższą funkcję i uwzględnić jedynie odpowiedzi składające się z małych liter.

**Uwaga:**

- Definicje funkcji powinny znajdować się na początku skryptu lub w pliku ~/.profile.

- Funkcje mogą być zagnieżdżone.

## 5.6. Inne polecenia

- `( . plik)`
  Wczytuje i wykonuje polecenia z `pliku`. Jeżeli pliku nie ma w katalogu bieżącym, sprawdzane są katalogi należące do zmiennej `PATH`

- `break [ n ]`
  Powoduje wyjście z pętli `for` lub `while`. Opcjonalny parametr $n$ określa liczbę poziomów pętli, z których należy wyjść. Domyślnie $n$ jest równe 1

- `continue [ n ]`
  Powoduje kontynuowanie następnej iteracji pętli `for` lub `while` (opcjonalnie $n$-tej pętli)

- `eval [ argument ]`
  Wczytuje argumenty i próbuje wykonać powstałe polecenie. Umożliwia to podstawienie wartości parametrów ukrytych parametrów lub poleceń

- `exec [ argument ]`
  Powoduje wykonanie instrukcji będącej argumentem polecenia `exec`. Nowe polecenie jest wykonywane zamiast bieżącej powłoki, nie tworząc nowego procesu. Można przy tym podać argumenty modyfikujące wejście i wyjście.

- Polecenie `exit [ n ]`zamyka powłokę z kodem stanu `n`. Jeżeli brak parametru `n`, kodem stanu jest kod ostatniego polecenia wykonanego w powłoce

- `export [ zmienna ]`
  Zmienia zakres `zmiennej` z lokalnego na globalny. Jeżeli nie podano żadnego argumentu, wydrukowana zostanie lista zmiennych, których zakres zmieniono z lokalnego na globalny

- `hash [ -r ] [ nazwa ]`
  Zapamiętuje położenie pliku wykonywalnego polecenia `nazwa` w katalogu będącym częścią zmiennej `PATH`. Opcja `-r` powoduje anulowanie pamięci o lokalizacji pliku `nazwa`. To samo polecenie wykonane bez żadnych opcji drukuje listę poleceń z zapamiętaną lokalizacją pliku wykonywalnego

- `readonly [ nazwa ]`
  Uniemożliwia zmianę wartości zmiennej `nazwa`

- `return [ n ]`
  Powoduje wyjście z funkcji, gdzie $n$ jest wartością zwracaną przez funkcję. Jeżeli brak jest argumentu w instrukcji `return`, zwracany jest kod wynikowy (*exit code*) ostatniego polecenia wewnątrz funkcji

- `shift [ n ]`
  Przesuwa parametry pozycyjne o $n$ miejsc w lewo. Domyślnie $n$ jest równe 1. Na przykład, wartość zmiennej `$2` staje się wartością zmiennej `$1`, wartość zmiennej `$3` staje się wartością zmiennej `$2` itd.

- `times`
  Drukuje łączny czas użytkownika i systemu zużyty przez procesy uruchamiane z bieżącej powłoki

- `trap [ argument ] [ n ]`
  Umożliwia wykonanie poleceń zawartych w argumencie, pod warunkiem otrzymania sygnału $n$ (w postaci numerycznej lub symbolicznej)

- `type [ nazwa ]`
  Drukuje informację o tym, jak interpretowana będzie `nazwa`, gdy zostanie użyta jako nazwa polecenia

- `unset [ name ]`
  Anuluje wartości zmiennych lub funkcji. Nie można anulować wartości zmiennych `PATH`, `PS1`, `PS2`, `MAILCHECK` i `IFS`

- `wait [ n ]`
  Powoduje oczekiwanie na zakończenie procesu o identyfikatorze $n$ działającego w tle. W przypadku braku argumentu oczekiwanie dotyczy wszystkich procesów uruchomionych w tle

## 5.7. Proste przykłady

1. Prosta iteracja w pętli `for`

```
# Inicjowanie zmiennej
alfabet="a b c d e"
# Inicjowanie licznika
licz=0
for litera in $alfabet
do
  licz=`expr $licz + 1`
  echo "litera nr $licz : [$litera]"
done
```

2. Prosta iteracja w pętli `while`

```
alfabet="a b c d e"
licz=0
while [ $licz -lt 5 ]
do
  licz=`expr $licz + 1`
  # Pozycja nastepnej litery
  pozycja=`bc $licz + $licz - 1`
  # Pobranie nastepnej litery
  litera=`echo "$alfabet" | cut -c$pozycja-$pozycja`
  echo "Litera $licz to [$litera]"
done
```

3. Iteracja po elementach będących wynikiem innego polecenia

```
echo "Nastepujace pliki biezacego"
echo "katalogu zawieraja ciag abc:"
for i in `grep -l 'abc' *`
do
  printf "%6d %s\n" `grep -c 'abc'` $i
done
```

4. Sortowanie wyników z poprzedniego przykładu według liczby wystąpień poszukiwanego ciągu; wyjście całej pętli `for` jest kierowane do instrukcji `sort`

```
echo "Nastepujace pliki biezacego katalogu"
echo "zawieraja ciag abc:"
( for i in `grep -l 'abc' *`
  do
    echo $i
  done
) | sort -nr
```

5. Nawiasy `[]` nie muszą pojawiać się w instrukcji `echo`. Są konieczne w instrukcjach `if` i `elif`

```
if [ -f $katalog/$plik ]
then
  echo "Plik [$plik] istnieje"
elif [ -d $katalog ]
then
  echo "Katalog [$katalog] istnieje"
else
  echo "Nie istnieje ani [$katalog] ani [$plik]"
fi
```

6. Symbol `||` oznacza alternatywę, a `&&` oznacza koniunkcję. Słowo `then` musi być umieszczone w osobnym wierszu

```
if [ -f $kat/$plik ] || [ -f $kat/$nowyplik ]
then
  echo "Istnieje plik [$plik]"
  echo "lub istnieje plik [$nowyplik]"
elif [ -d $kat ]
then
  echo "Katalog [$dir] istnieje"
else
  echo "Nie istnieje katalog [$kat]"
  echo "nie istnieje tez plik [$plik]"
  echo "ani plik [$nowyplik]"
fi
```

7. Między nawiasami `()` lub `{}` można umieścić wiele poleceń. Polecenia w nawiasach okrągłych `( )` są wykonywane w osobnej podpowłoce. Polecenia w nawiasach klamrowych `{ }` są wykonywane w tej samej powłoce. Nawias klamrowy musi być oddzielony spacją od tekstu

```
if [ -f "$1" ]
then
  rozmiar=`ls -l "$1" | awk '{ print $5 }' `
  if [ $rozmiar -eq 0 ]
  then
    echo "plik pusty"
  else
    echo "plik niepusty"
  fi
fi
```

8. Parametry przekazywane do skryptu

```
echo "przekazano $# parametrow"
# drukuje dziesiec pierwszy parametrow
echo "$1 $2 $3 $4 $5 $6 $7 $9 $10"
# drukuje liste parametrow jako jedna zmienna
echo "$*"
# drukuje liste parametrow
# jako jedna zmienna bez odstepow
echo "$@" | sed 's/ //g'
for i in $*
do
  # tworzy pusty plik o nazwie rownej
  # wartosci zmiennej i
  > $i
done
```

9. Usuwanie plików o rozmiarze równym `0` bajtów

```
for i in *
do
  if [ ! -s $i ]
  then
```

```
    echo "plik $i jest pusty, usuwamy go"
    rm -f $i
  fi
done
```

10. Zmienne domyślne `$0`, `$#`, `$$`

```
echo "wykonuje skrypt zapisany w pliku $0" |\
 tee -a $0.tmp.$$
echo "do skrypty przekazano $# parametrow" |\
 tee -a $0.tmp.$$
echo "numer procesu (pid) tego skryptu = $$" |\
 tee -a $0.tmp.$$
```

11. W skrypcie obliczana jest suma bajtów wszystkich elementów bieżącego katalogu. Dobrym ćwiczeniem jest wprowadzenie w tym skrypcie zmian, aby np. sumować tylko rozmiary zwykłych plików

```
ls -la | awk '{ s+=$5 } END{ print s }'
```

12. Sprawdzamy czy istnieje `plik`. Jeśli nie istnieje, instrukcja `exit` powoduje zakończenie skryptu

```
# exit.sh
if [ ! -f plik ]
then
  exit 1
  echo "czy zdaze wydrukowac do widzenia?"
fi
```

13. Sprawdzanie kodu wynikowego ostatniego wykonanego polecenia `$?`

```
# uruchamiamy skrypt z poprzedniego przykladu
./exit.sh
if [ $? -ne 0 ]
then
  echo "exit code (kod stanu) jest rozny 0"
else
  echo "exit code (kod stanu) = 0"
fi
```

14. W kolejnym przykładzie skrypt zmienia końcową część nazwy tych plików z bieżącego katalogu, które mają rozszerzenie `.xxx`. Nowe rozszerzenie ma postać `.yyy`. Pierwszy człon nazwy pozostaje bez zmian

```
for plik in *.xxx
do
   nowy=`basename $plik .xxx`
   mv $plik $nowy.yyy
done
```

15. Połączenie w jedną całość wszystkich plików z bieżącego katalogu mających rozszerzenie `.txt`:

```
for plik in $(ls *.txt); do
  cat $plik >> calosc.txt
done
```

16. Zamiana wielkich liter w nazwach plików na małe we wszystkich plikach bieżącego katalogu

```
for f in *; do
  mv $f `echo $f | tr '[A-Z]' '[a-z]'`
done
```

17. Instrukcja `case`. Mamy pliki zawierające dane osobowe. W jednym pliku są dane jednej osoby, na przykład:

```
# Plik Adam.Kowalski
Adam Kowalski
ur. 1 1 1980
ul. Dluga 5
01-900 Miasto
```

Następujący skrypt oferuje wybór danych do wyświetlenia:

```
a=plik1
b=plik2

echo "Czyje dane chcesz obejrzec? Wybierz:"
echo "[K]dam Kowalski"
echo "[N]arbara Nowicka"
echo "[JN]an Nowak"

read kto

case "$kto" in
  "K") cat "Adam.Kowalski"
       echo "skonczone"
    ;;
  "N") cat "Barbara.Nowicka"
    echo "skonczone"
    ;;
  "JN") cat "Jan.Nowak"
       echo "skonczone"
    ;;
  *) echo "niezrozumiala odpowiedz"
    ;;
esac
```

18. Drukowanie tylko nazw katalogów, z pominięciem nazw plików

```
echo "katalogi:"
for i in $1/*
do
  if [ -d $i ]
  then
    echo $(a)"/$i"
  fi
done
```

19. Inny sposób drukowania zawartości katalogu. Najpierw drukowane są nazwy katalogów, potem plików, przy czym sprawdzany jest rozmiar pliku i uprawnienia dostępu do pliku. Odpowiednie komunikaty są drukowane, jeżeli plik ma rozmiar 0 lub ma uprawnienia uruchamiania przez wszystkich użytkowników

```
# uruchomienie skryptu:
# ./skrypt.sh katalog

echo "katalogi:"
for i in $1/*
do
  if [ -d $i ]
  then
    echo "$i"
  fi
done

echo "----------------------------------------"
echo "pliki:"
for i in $1/*
do
  if [ -f $i ]
  then
    if [ -s $i ]
    then
      echo "$i"
      if [ -x $i ]; then
        c=`ls -l $i | grep '^.........x'`
        if [ -n "$c" ]
        then
          echo '--> uprawnienia wykonywania'
          echo 'dla wszystkich'
          chmod 700 $i
        fi
      fi
    else
      echo "plik $i ma rozmiar 0"
    fi
  fi
done
```

20. Następujący skrypt zmienia nazwy wszystkich plików znajdujących się w katalogu bieżącym w taki sposób, że rozmiar pliku w bajtach jest ostatnim członem nazwy

```
for i in *
do
  if [ -f $i ]; then
    a=`ls -la $i | awk '{ print $5 }'`
    nazwa=`echo "$i.$a"`
    cp $i $nazwa
  fi
done
```

21. Dokument typu *here*; wygodny sposób na drukowanie dłuższych komunikatów

```
cat << DOTAD
--------------
Uwaga:

To jest komunikat skryptu
--------------
DOTAD
```

22. Tryb *restricted*

```
#!/bin/bash -r
```

Można też tak: `set -r`

W tym trybie nie można zmienić wartości większości zmiennych środowiskowych, np. `PATH`, `SHELL`, `ENV`. Nie można np. zmienić katalogu poleceniem `cd`.

23. Obsługa przerwań. Można to zrobić za pomocą polecenia `trap`.

```
trap `echo Nacisnales Ctrl+C` INT
echo "Nacisnij Ctrl+C, aby przerwac petle"
while [ : ]
do
  echo "W petli"
  sleep 2
done
echo "Petla zakonczona"
trap - INT
```

## 5.8. Optymalna kompresja plików

Następujący skrypt wykonuje kompresję plików metodą gzip i bzip2. Rozmiary plików po kompresji obiema metodami są porównywane. Większy plik jest usuwany.

```
for i in $*
do
# rozmiar przed kompresja
  rozmiar=`ls -l $i | awk '{ print $5 }'`
  cp $i $i.tmp1
  cp $i $i.tmp2
  time gzip $i.tmp1
  time bzip2 $i.tmp2

# rozmiar po kompresji
  b=`ls -l $i.tmp1.gz | awk '{ print $5 }'`
  g=`ls -l $i.tmp2.bz2 | awk '{ print $5 }'`
  echo "$i  rozmiar=$rozmiar  g=$g  b=$b"

# usuwamy mniejszy plik
  if [ $g -le $b ]; then
    mv $i.tmp1.gz $i.gz
    rm -f $i.tmp2.bz2
  else
    mv $i.tmp2.bz2 $i.bz2
    rm -f $i.tmp1.gz
  fi
done
```

Należałoby sprawdzać, czy w czasie działania skryptu pojawiły się błędy i w przypadku zaistnienia ich nie usuwać plików nieskompresowanych. Jeśli błędów nie ma, usunąć oryginały. Można użyć zmiennej domyślnej `$?`, ale wówczas należałoby sprawdzać wynik każdego polecenia. Inna możliwość polega na skierowaniu komunikatów diagnostycznych do jednego pliku i sprawdzeniu, czy jego rozmiar jest większy od 0. Jeśli tak jest, przerwać działanie skryptu.

## 5.9. Masowa wysyłka poczty

Jak wysłać pocztę elektroniczną do wielu adresatów jednocześnie? Jeden sposób polega na użyciu aliasów pocztowych zapisanych w pliku `.mailrc`. Jedna nazwa zastępcza odpowiada większej liczbie użytkowników. Aby wysłać wiadomość do jednej takiej grupy użytkowników, należy wykonać polecenie `mail -s "temat" alias < wiadomosc`, gdzie `alias` jest odpowiednią nazwą zastępczą.

1. Oto przykład, w którym temat wiadomości jest przekazywany jako ciąg parametrów podczas uruchomienia skryptu:

   ```
   echo "podaj adres odbiorcy:"
   while read adres
   do
     echo $adres
     mail -s "$*" $adres < wiadomosc
   done
   ```

   Adres odbiorcy lub nazwa zastępcza grupy odbiorców jest przekazywany interakcyjnie podczas działania skryptu. Uruchomienie skryptu:

   ```
   $ ./mail.sh temat wiadomosci
   ```

   Wadą tego rozwiązania jest to, że w nagłówku poczty u odbiorcy widoczne są adresy wszystkich pozostałych adresatów.

2. Oto inny skrypt. Tym razem odbiorca ujrzy w nagłówku przesyłki tylko swój adres pocztowy. Temat wiadomości znajduje się w pierwszym wierszu pliku `wiadomosc`. Tekst wiadomości obejmuje wiersze od 2 do końca pliku. Plik, w którym są zapisane adresy pocztowe odbiorców, po jednym w wierszu, jest przekazywany do skryptu jako parametr.

   ```
   temat=`head -1 wiadomosc`
   ladresow=`cat "$1" | wc -l`
   i=0
   while [ : ]
   do
    i=$(($i+1))
    adres=`sed -e -n '"$i"p' `
   ```

```
 mail -s "$temat" $adres < `sed -n '2,$p' $wiadomosc`
 echo "$?"
 if [ "$?" != 0 ]
 then
  break
 fi
done
if [ "$i" == "$ladresow" ]
then
 echo "OK"
else
 echo "wszystkich adresow jest $ladresow"
 echo "wiadomosc wyslano do $i adresow"
fi
```

3. W kolejnym przykładzie temat wiadomości jest przekazany jako argument skryptu. Wysyłka poczty do kolejnych użytkowników jest wykonywana w nieskończonej pętli `while`. Poszczególne adresy odbiorców są wczytywane instrukcją `read`.

```
temat="$*"
ladresow=`cat "$1" | wc -l`
i=0
while [ : ]
do
  read adres
  mail -s "$temat" $adres < wiadomosc
  echo "$?"
  if [ "$?" != 0 ]
  then
    break
  fi
  i=$(($i+1))
done
if [ "$i" == "$ladresow" ]
then
  echo "OK"
else
  echo "wszystkich adresow jest $ladresow"
  echo "wiadomosc wyslano do $i adresow"
fi
```

## 5.10. Wyszukiwanie adresów www w dowolnej grupie plików

1. W omawianym tutaj przykładzie do skryptu przekazywane są dwa parametry: nazwa pliku wejściowego i nazwa pliku wynikowego, w którym zapisane zostaną adresy wyodrębnione z pliku wejściowego. Wyrażenia regularne użyte w tym pliku nie są najbardziej ogólne z możliwych i zostały przetestowane przez autora na kilku niezbyt skomplikowanych przypadkach, kiedy plik wejściowy jest zwykłym plikiem html. Wyrażenia regularne należałoby uogólnić. Zmiany może też wymagać główna instrukcja skryptu filtrująca adresy.

```
# uruchomienie skryptu:
# ./adresy_www.sh plik_wej plik_wyj 'wzorzec'
# wzorzec jest poszukiwanym wyrazeniem regularnym;
# plik_wej zwykle jest plikiem html, stad
# wzorcem moze byc ciag http lub np. org, jezeli
# poszukujemy adresow nalezacych do tej domeny

# wynik jest sortowany, a powtorzone
# wiersze sa usuwane
# zamiast sekwencji "sort | uniq"
# mozna uzyc "sort -u"

sed -e 's/ /\
/g' -e '/^[ \t]/d' -e '/^$/d' $1 | grep $3 |\
sed 's/^.*http/http/; s/["<>].*$//;\/^http:\/\//!d'\
| sort | uniq> $2

# komunikat o liczbie adresow w pliku wynikowym
polecenie='wc -l $2 | cut -c1-7'
echo "znaleziono `eval $polecenie` adresow stron"

echo ""
a=`grep -c '\.com' $2`
echo "$a adresow .com"

echo ""
echo "`grep -c '\.org' $2` adresow .org"
```

2. W następnym przykładzie przeszukiwane są wszystkie pliki mające w nazwie ciąg `.htm` z katalogu, którego nazwa jest pierwszym parametrem skryptu.

```
# Uruchomienie skryptu:
# ./adresy_www.sh katalog plik_wyj 'wzorzec'

if [ -s $2 ]
then
  echo
  echo "Kontynuowac? [T/N]"
  read a
  if [ "$a" != "T" ]
  then
    exit 1
  fi
fi

for plik in $1/*.htm*
do
  echo "$plik"
  sed -e 's/ /\
  /g' -e '/^[ \t]/d' -e '/^$/d' $plik | grep $3 |\
  sed 's/^.*http/http/; s/["<>].*$//; /^http:\/\//!d'\
  | sort -u >> $2
done
```

Skrypt nie jest zoptymalizowany. Zachęcam do stworzenia własnej, prostszej wersji podobnego skryptu.

Adresy www mogą zawierać wielkie litery. Jednak `WWW.ADRES.PL`, `www.adres.pl` i `www.Adres.pl` oznaczają ten sam adres. Zatem należałoby przed sortowaniem zmienić wielkość liter, na przykład poleceniem `tr '[A-Z]' [a-z]'`.

## 5.11. Pobieranie wyników z wyszukiwarki Google

1. W tym podrozdziale prezentowany jest skrypt automatyzujący filtrowanie danych uzyskanych z zapytań kierowanych do popularnej wyszukiwarki internetowej Google.

```
# Uruchomienie skryptu:
# ./skrypt.sh kwerenda
# np. ./skrypt.sh rzadki gatunek kaczki

query=`echo "${@}" | sed 's/  */+/g'`
echo "$query"

URL=http://google.com/search?q="$query"

# Wykorzystujemy tekstowa przegladarke
# internetowa lynx; opcja -dump powoduje
# drukowanie pobranej tresci
# na standardowym wyjsciu. Wynik jest
# kierowany do pliku tymczasowego

lynx -dump $URL 2>/dev/null > google.tmp

# Plik z zawartoscia strony pobranej
# z google.com jest filtrowany, posortowane
# adresy sa zapisywane w pliku
# google.search, przy tym adresy zawierajace
# ciag google sa odrzucane

sed -e 's/ /\
/g' -e '/^[ \t]/d' -e '/^$/d' google.tmp |\
grep http | sed 's/^.*http/http/;\
     s/["<>].*$//; /^http:\/\//!d'|\
sort -u | grep -v google > google.search

# Usuwamy plik tymczasowy

rm -f google.tmp
```

```
#######################################################
# Liczymy, ile wierszy jest w pliku.
# Drukujemy tylko pierwsze pole (drugim polem
# w wyniku wc -l jest nazwa pliku)
#
# lwierszy=`wc -l google.search | awk '{ print $1 }'`
#
#######################################################
# Mozna inaczej. Usuwamy poczatkowe spacje
# i pobieramy pierwsze pole tekstowe, przyjmujac,
# ze pola oddzielone sa spacjami, cut -f1 -d ' '
#
# lwierszy=`wc -l google.search |\
#        sed 's/^[ ]*//' | cut -f1 -d ' '`
#
#######################################################
# Inny wariant. Jesli przyjrzec sie wynikowi
# polecenia wc -l, okazuje sie, ze liczba wierszy
# drukowana jest na 7 polach z wyrownaniem do
# prawej. Aby otrzymac liczbe wierszy, wystarczy
# pobrac 7 pierwszych znakow z wyniku polecenia
# wc -l, czyli cut -c1-7

lwierszy=`wc -l google.search | cut -c1-7`

# Uwaga. W tej wersji zmienna lwierszy zawiera
# teraz 7 znakow. Gdybysmy chcieli jej uzyc
# jako lancucha znakow, to wywolanie wartosci
# zmiennej $lwierszy da inny wynik niz
# "$lwierszy". W tym drugim przypadku
# wydrukowane beda poczatkowe spacje nalezace
# do wartosci zmiennej.

echo "Liczba wierszy w pliku google.search:"
echo "$lwierszy"
echo $lwierszy""

# Jesli zmienna lwierszy uzyta jest w kontekscie,
# w ktorym oczekiwana jest wartosc liczbowa, jak
# ponizej w instrukcji while z operatorem -le,
```

```
# ta roznica nie ma znaczenia.
# W dwoch poprzednich wariantach uzyskiwania
# wartosci lwierszy lancuchy "$lwierszy"
# i $lwierszy sa rowne
#######################################################

i=1
echo "Znaleziono adres:"
while [ "$i" -le "$lwierszy" ]
do
  i=$(($i+1))
  adres=`sed -n "$i p" google.search`
  echo $adres
done
```

Adresy stron są zapisane w pliku `google.search`. Można je pobrać poleceniem `wget -i google.search`. Czytelników zainteresowanych dalszą automatyzacją pobierania danych z Internetu autor odsyła do dokumentacji narzędzia `wget`.

2. W poprzednim skrypcie pobierano jedynie adresy z pierwszej strony wyników zwracanych przez wyszukiwarkę Google. Warto pójść nieco dalej i umożliwić automatyczne pobranie wszystkich adresów zwracanych przez wyszukiwarkę. Pętla służy do przejścia po wszystkich wynikach wyszukiwania.

```
# Uruchomienie skryptu: ./adresy_www.sh plik.html plik.wynik

strona=10
query=`echo "${@}" | sed 's/  */+/g'`
echo "$query"

URL=http://google.com/search?q="$query"\&num="$strona"
lynx -dump $URL 2>/dev/null >> google.tmp

i=10
i=$(($i+$strona))
grep start="$i" google.tmp
a=`grep start="$i" google.tmp`
echo ${#a}
while [ -n "$a" ]
```

```
do
  echo $i
  URL=http://google.com/search?q="$query"\&num="$i"
  lynx -dump $URL 2>/dev/null >> google.tmp
  i=$(($i+$strona))
  a=`grep start="$i" google.tmp`
done

sed -f a.sed google.tmp | grep 'http' | sed -f a1.sed \
  | sort -u | grep -v 'google' \
  |  grep '\.' > google.search

# rm -f google.tmp

lwierszy=`wc -l google.search | awk '{ print $1 }'`

echo "Liczba wierszy w pliku google.search:"
echo $lwierszy

i=1
echo "Znaleziono adres:"
while [ "$i" -le "$lwierszy" ]
do
  i=$(($i+1))
  adres=`sed -n "$i p" google.search`
  echo $adres
done
```

## 5.12. Sortowanie adresów internetowych

1. Adresy internetowe z poprzedniego podrozdziału są zapisane w pliku `google.search`. Należy je posortować w porządku alfabetycznym względem kolejnych pól adresu, poczynając od prawej strony, czyli od domen krajowych i funkcjonalnych (np. pl, de, com, org, edu, net)

```
sed -e 's/^www\.//; s/\./ /g' google.search |\
awk '{
  printf("%-4s %-20s", $NF, $(NF-1))
  for (i=NF-2; i>0; i=i-1) printf(" %s", $i)
  printf("\n")
}' |\
sort
```

Uruchomienie skryptu:

```
./google_www_1.sh rzadki gatunek kaczki
```

Oto przykładowy wynik:

```
104  9                    102 66
2    33                   14 212
30   165                  55 80
com  polambeirut
com  yahoo                groups
pl   agroturystyka
pl   bazafirm             edycja
pl   bird
pl   cgm                  alt
pl   chelm                opus mors
pl   kujawsko-pomorskie
pl   linia                eksplor
pl   net                  kki
pl   ngo                  free
pl   org                  bocian
pl   przyrodapolska
```

2. Ten sam skrypt po małej modyfikacji formatu drukowania

```
sed -e 's/^www\.//; s/\./ /g' google.search |\
awk '{
  printf("%-4s %-12.10s", $NF, $(NF-1))
  for (i=NF-2; i>0; i=i-1) printf(" %s", $i)
  printf("\n")
}' |\
sort |\
awk '{ printf("%4s %-12.10s", $1, $2)
  for (i=3; i<=NF; i++)
  {
    printf(" %s", $i)
  }
  printf("\n")
}'
```

Wynik:

```
 104 9            102 66
   2 33           14 212
  30 165          55 80
 com polambeiru
 com yahoo        groups
  pl agroturyst
  pl bazafirm     edycja
  pl bird
  pl com          kurierbytowski
  pl com          suwalszczyzna
  pl edu          amu main
  pl expedition
  pl gov          uw bialystok dziennik
  pl gt           birding
  pl ids          sandomierz lo1
  pl kujawsko-p
  pl linia        eksplor
  pl lubelskie    powiatlublin
  pl net          invar zc
  pl net          kki
  pl org          bocian
  pl przyrodapo
```

3. W poleceniu `sort` można wskazać separator pól i dzięki temu można sortować tekst w kolejnych polach

```
awk 'BEGIN{ FS="." }{
  for (i=NF; i>0; i=i-1) printf("%s ", $i)
  printf("\n")
}' google.search |\
sort -t '.'
```

Wynik:

```
104 9 102 66
2 33 14 212
30 165 55 80
com polambeirut
```

```
com yahoo groups
pl agroturystyka www
pl bazafirm edycja
pl bird www
pl cgm alt www
pl chelm opus mors www
pl cidernet www
pl com kurierbytowski www
pl com suwalszczyzna
pl edu amu main
pl expeditions www
pl gov um gizycko www
pl gov uw bialystok dziennik
pl gt birding www
pl ids sandomierz lo1 www
pl kujawsko-pomorskie www
pl linia eksplor
pl lublin 14lo eximus www
pl net invar zc www
pl net kki www
pl org bocian www
pl org most mto www
pl org salamandra www
pl przyrodapolska www
pl techweb www
pl zielonasiec biuletyn
```

4. Pobranie stron www znajdujących się pod adresami zapisanymi w pliku

```
# uruchomienie: ./skrypt adresy_www.d
lwierszy=`wc -l $1 | awk '{ print $1 }' `
i=1
while [ "$i" -le "$lwierszy" ]
do
  i=$(($i+1))
  adres=`sed -n "$i p" $1`
  wget -O $adres.html $adres
done
```

Polecenie `wget -i plik` umożliwia odczytywanie adresów URL z pliku. Pożytecznym ćwiczeniem jest zapisywanie kopii stron do oddzielnych

plików. Oprócz programu `wget`, który powstał specjalnie w celu automatyzacji pobierania zasobów internetowych, można skorzystać z wielu innych narzędzi. Przykład użycia w tym celu skryptów awk przedstawiono w poprzednim rozdziale.

Inna możliwość polega na zastosowaniu tekstowej przeglądarki internetowej lynx:

```
$ lynx -dump $adres > $adres.html
```

5. Skrypt wyszukujący adresy http w plikach binarnych. W tym przykładzie przeszukiwany jest katalog `/usr/bin`. Można to zmienić na dowolny inny katalog. Pierwszy skrypt zapisuje nazwy plików zawierających wyszukiwany ciąg do pliku `http_usr_bin.d`.

```
for plik in /usr/bin/*
do
  if strings $plik | grep -q 'http:'
  then
    echo "$plik" >> http_usr_bin.d
  fi
done
```

Drugi skrypt zapisuje wiersze zawierające ciąg `http` do pliku wynikowego

```
# uruchomienie skryptu poleceniem
# ./skrypt http_usr_bin.d
# if [ -f http_adresy_usr_bin.d ]

# Sprawdzamy, ile wierszy jest w pliku
lwierszy=`wc -l $1 | awk '{ print $1 }' `
echo $lwierszy

i=1
while [ "$i" -le "$lwierszy" ]
do
  i=$(($i+1))
  # odczytujemy $i-ty wiersz
  plik=`sed -n "$i p" $1`
```

```
    # Wiersze zawierajace wyszukiwany ciag
    # sa dopisywane do pliku wynikowego
    echo "$plik `strings $plik |\
      grep 'http:' `" >> http_adresy_usr_bin.d
    echo ""
  done
```

## 5.13. Przeszukiwanie grup dyskusyjnych w wyszukiwarce Google

Wyszukiwarka Google umożliwia sprawne przeszukiwanie archiwów grup dyskusyjnych. Pokażemy to na przykładzie grupy dyskusyjnej `pl.praca.oferowana`.

```
# nazwa przeszukiwanej grupy
grupa="pl.praca.oferowana"
# wyszukiwane slowo
slowo="$1"
# plik, do ktorego maja byc zapisane wyniki
plik="$2"

biez_miesiac=`date +"%m" | sed 's/^0//'`
biez_rok=`date +"%Y" `
# najstarsze dane w grupie pl.praca.oferowana
# pochodza z pazdziernika 1996
l=10
r=1996
# petla po wszystkich latach i miesiacach
while [ $r -le $biez_rok ]
do
  for m in 1 2 3 4 5 6 7 8 9 10 11 12
  do
    if [ $r -eq $biez_rok ]
    then
      if [ $m -gt $biez_miesiac ]
        then break
      fi
    fi
# W skladni wyszukiwania zaawansowanego w grupach
# dyskusyjnych Usenet w wyszukiwarce Google
```

```
# nalezy podac date poczatkowa wyszukiwania
# i date koncowa; tutaj date poczatkowa okreslaja
# zmienne dp, mp, rp - dzien, miesiac i rok
# dla daty poczatkowej oraz
# dk, mk, rk - dzien, miesiac i rok dla daty koncowej

    dp=1
    mp=$m
    rp=$r
    dk=31
    mk=$m
    rk=$r
    URL1="http://groups.google.pl/groups?"
    URL2="hl=pl&as_q=${slowo}&"
    URL3="as_ugroup=${grupa}&as_drrb=b&"
    URL4="as_mind=${dp}&as_minm=${mp}&"
    URL5="as_miny=${rp}&as_maxd=${dk}&"
    URL6="as_maxm=${mk}&as_maxy=${rk}&lr=&num=${l}"
    URL=$URL1$URL2$URL3$URL4$URL5$URL6

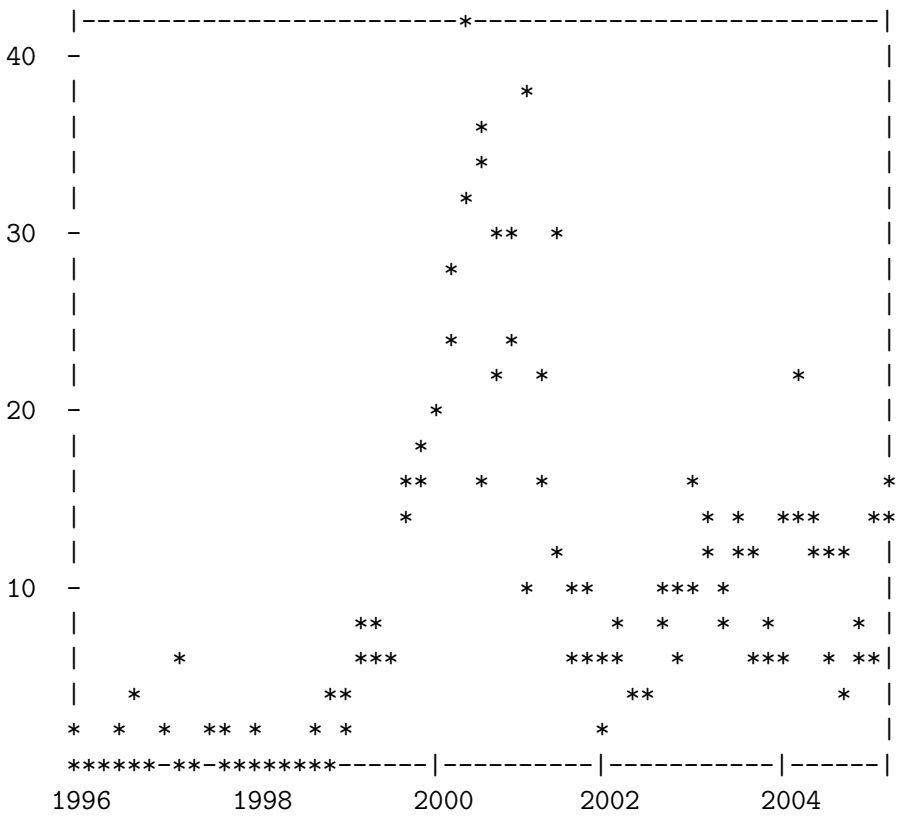
    wzorzec='Grupy dyskusyjne.*Wyniki'

# n jest liczba ogloszen w danym miesiacu
# zawierajacych poszukiwany wzorzec

    n=`lynx -dump "$URL" | egrep "$wzorzec" |\
    sed -e 's/ w przedziale.*$//' | awk '{ print $NF }'`
    if [ -z $n ]; then n=0; fi

    echo "$r $m $n" | tee -a $plik
  done
  r=$(($r+1))
done
```

Wynik:




Jeżeli chcemy dowiedzieć się, ile było wiadomości w grupie dyskusyjnej w danym okresie, wystarczy zrezygnować z podawania wzorca. Zamiast `plik="$2"` mamy `plik="$1"`, a zmienna `URL2` ma postać `URL2="hl-pl"`. Zmieniają się także zmienne `wzorzec` i instrukcja `n=...`:

```
wzorzec=' z wszystkich.*'
n=`lynx -dump "$URL" | egrep "$wzorzec" |\
sed 's/ w pl.praca.oferowana.*$//' | awk '{ print $NF }'`
```

## 5.14. Pobieranie wiadomości z serwera grup dyskusyjnych

1. W tym przykładzie pobierane są najnowsze ogłoszenia w grupie `pl.praca.oferowana`, dostępne na serwerze `usenet.gazeta.pl`. To rozwiązanie zostało przetestowane w kwietniu 2004.

```
URL1=http://usenet.gazeta.pl
URL2=/usenet/0,1.html?group=pl.praca.oferowana

echo "URL1 = $URL1"
echo "URL2 = $URL2"
URL=${URL1}${URL2}
echo "URL = $URL"

i=1
until ! lynx -dump "$URL" > $$.$i
do
  lynx -dump "$URL" 2>/dev/null > $$.$i
  URL=`egrep '[0-9]*. h.*\&pagen' $$.$i |\
     sed 's/^[ ]*[0-9]*\. //'`
  i=$(($i+1))
  echo "$i   $URL"
# W pliku adresy_stron gromadzone sa adresy
# do poszczegolnych ogloszen
  grep 'tid=' $$.$i |\
  sed 's/^.*http:/http:/' >> adresy_stron'
done
```

   Następnie można za pomocą polecenia `lynx -dump` lub `wget -i` pobrać ogłoszenia, których adresy zostały zapisane w pliku `adresy_stron`. Na początku pliku znajdują się najnowsze ogłoszenia.

2. Tym razem wiadomości są pobierane z serwera `niusy.onet.pl`. Skrypt przetestowano w kwietniu 2004. Adresy poszczególnych wiadomości są zapisywane w pliku `onet_rok_miesiac`, na przykład `onet_2004_4`.

```
URL1=http://niusy.onet.pl/\
  niusy.html?t=watki\&group=pl.praca.oferowana
```

```
echo $URL1

# Serwer niusy.onet.pl przechowuje wiadomosci
# grup dyskusyjnych od pierwszej polowy 1999 roku
# mp - miesiac poczatkowy w roku
# mc - miesiac koncowy w roku

for rok in 1999 2000 2001 2002 2003 2004
do
  if [ $rok -eq 1999 ]
  then
    mp=5
  else
    mp=1
  fi

  if [ $rok -eq `date +"%Y"` ]
  then
    mc=`date +"%m"`
  else
    mc=12
  fi

  miesiac=$mp
  while [ $miesiac -le $mc ]
  do
    strona=0
    while [ : ]
    do
      echo $rok $miesiac $strona
      URL2=\&year=${rok}\&month\
            =${miesiac}\&page=${strona}
      URL=${URL1}${URL2}
      echo ${URL1}${URL2}
      lynx -dump $URL 2>/dev/null |\
        grep 'aid=' | sed 's/^.*http:/http:/' \
        >> onet_$rok_$miesiac
      if [ $? -ne 0 ]
      then
        break
```

```
      fi
  #    URL=`egrep '[0-9]*. h.*\&pagen' $$.$i |\
  #       sed 's/^[ ]*[0-9]*\. //'`
      strona=$(($strona+1))
    done
    miesiac=$(($miesiac+1))
  done
done
```

Oto przykład zawartości pliku onet_2000_4:

```
http://niusy.onet.pl/niusy.html?t=artykul&group=
  pl.praca.oferowana&aid=4708695
http://niusy.onet.pl/niusy.html?t=artykul&group=
  pl.praca.oferowana&aid=4708696
http://niusy.onet.pl/niusy.html?t=artykul&group=
  pl.praca.oferowana&aid=4708699
http://niusy.onet.pl/niusy.html?t=artykul&group=
  pl.praca.oferowana&aid=4708700
...
```

W pliku każdy adres mieści się w całości w jednym wierszu. Następny program wykorzystuje te dane i pobiera strony internetowe o adresach zapisanych w pliku.

```
# Uruchomienie skryptu:
# ./.onet1.sh plik
# gdzie plik jest nazwa jednego z plikow
# utworzonych w poprzednim skrypcie onet.sh, np.
# ./onet1.sh onet_2004_10

URL1=http://niusy.onet.pl/niusy.html?\
t=artykul\&group=pl.praca.oferowana\&aid=

lwierszy=`cat $plik | wc -l`
echo "$plik $lwierszy"
i=0
while [ $i -lt $lwierszy ]
do
  i=$(($i+1))
```

```
  adres=`sed -n "$i p" $plik`
  URL=${URL1}${aid}
  echo "$i $adres"
  lynx -dump $URL 2>/dev/null | \
    sed -n '/poprzedni.*ul.*pny.*ul/,\
      /poprzedni.*ul.*pny.*ul/p' >> wynik_$plik
done
```

## 5.15. Monitorowanie obciążenia systemu

1. **Liczba użytkowników**

   Polecenie `users` daje następujący wynik:

   ```
   $ users
   bishop g_leskiewicz kangoo leon lichota lsb mcl
   mmarciniak mstgeo potworek steve

   users | awk '{ print NF }' | uniq | wc -l
   ```

   Inny sposób uzyskania listy zalogowanych użytkowników:

   ```
   who | awk '{ print $1 }' | sort | uniq | wc -l
   ```

2. **Średnie obciążenie systemu**

   Uzyskiwanie informacji o średnim obciążeniu systemu

   ```
   uptime | sed 's/[ ][ ]*/\
   /g' | sed -n '/average/,/[0-9]p' | sed -n '2s/,//p'
   ```

   Dane o liczbie użytkowników i wielkości obciążenia systemu są drukowane na ekranie co 10 sekund:

   ```
   echo "godzina (HH MM SS) liczba_uzytk obciazenie"
   i=0
   while [ : ]
   do
     liczba_u=`who | awk '{ print $1 }' |\
     sort | uniq | wc -l`
   ```

```
  obciazenie=`w | head -1 | sed 's/^.*age: //' |\
  sed 's/,.*$//' `
#  godzina=`date "+%H %M %S"`
#  echo "$godzina $liczba_u $obciazenie"
  echo "$i $liczba_u $obciazenie"
  i=$(($i+1))
  sleep 10
done
```

3. **Monitorowanie obciążenia komputera**

   Tym razem dane są zapisywane do pliku

```
data=`date "+%d_%m_%y_%H_%M"`
# i=0
while [ : ]
do
# liczba uzytkownikow zalogowanych w systemie
  uz=`who | awk '{ print $1 }' |\
    sort | uniq | wc -l`

# obciazenie
  load=`w | head -1 |\
    sed 's/^.*age: //' | sed 's/,.*$//'`

# biezacy czas (godzina, minuta, sekunda)
  czas=`date | awk '{ print $4 }' | sed 's/:/ /g'`

# data
  dzien=`date "+%d_%m_%y"`
  echo "$czas $uz $obciaz" |\
  tee -a obciaz.$data.$$ |\
  tee -a obciaz.$dzien
  sleep 60
done
```

   Dane są zapisywane przez skrypt powłokowy działający przez wiele dni w pętli. Co minutę do pliku zapisywane są informacje o liczbie użytkowników zarejestrowanych (zalogowanych) w danym momencie w systemie, numer ostatniego uruchomionego procesu oraz średnie obciążenie

systemu w ciągu ostatniej minuty. Skrypt był testowany w systemie FreeBSD na serwerze `hoth.amu.edu.pl`.

Oczywiście, są też inne metody monitorowania systemu. Jednak nie zawsze zwykły użytkownik systemu ma do nich dostęp.

4. Teraz prześledźmy cały proces pozyskiwania danych i ich obróbki, aż do uzyskania wykresu.

   Najpierw skrypt powłoki bash pozyskujący dane:

```
# Skrypt jest uruchamiany w tle; dziala takze
# po wylogowaniu uzytkownika z systemu:
# nohup ./load.sh &

a=`date |awk '{ printf("%s_%s_%s_%s",$2,$3,$4,$5) }'`
echo "$a"
i=0
echo "czas liczba_uzytk. last_pid obciazenie_systemu"
while [ : ]
do
  i=$(($i+1))
  users=`who | sort -u | wc -l`
  pid_i_load=`top -b | head -1 | sed 's/[,;]//g'\
    | awk '{ print $3, $6 }'`
  godz=`date "+%w %H %M"`
  echo "$i $godz $users $pid_i_load" | tee -a load.$a
  sleep 60
done
```

   Powyższy skrypt jest w postaci uruchamianej na serwerze `hoth.amu.edu.pl`, działającym w systemie FreeBSD. W innych wersjach systemu Unix może wymagać drobnych zmian, zależnie od szczegółów implementacji poszczególnych poleceń. Oto wersja tego skryptu działająca na serwerze `main2.amu.edu.pl`:

```
# load3.sh; wersja dzialajaca na main.amu.edu.pl
# Zbierane sa m.in. dane o numerze dnia tygodnia,
# np. sobota ma nr 6. Dzieki temu mozna bedzie
# narysowac wykres sredniego obciazenia systemu
# na podstawie danych z wielu tygodni.
```

```
a=`date |awk '{ printf("%s_%s_%s_%s",$2,$3,$4,$5) }'`
echo "$a"
i=0
echo "czas liczba_uzytkownikow obciazenie_systemu"
while [ : ]
do
  i=$(($i+1))
  users=`who | awk '{ print $1 }' |\
  sort | uniq | wc -l`
  load=`w | head -1 | sed 's/[,;]//g' |\
  awk '{ print $10 }'`
  godzina=`date "+%w %H %M"`
  echo "$i $godzina $users $load" | tee -a load.$a
  sleep 60
done
```

Oto fragment pliku danych:

```
1 5 11 35       24 38611 0.00
2 5 11 36       23 38661 0.08
3 5 11 37       23 38697 0.02
4 5 11 38       23 38735 0.01
5 5 11 39       23 39204 0.00
6 5 11 40       23 39262 0.00
7 5 11 41       23 39314 0.00
8 5 11 42       21 39347 0.10
9 5 11 43       22 39382 0.04
10 5 11 44       22 39431 0.26
...
```

Aby obliczyć średnie obciążenie systemu w dłuższym okresie, należy wykonać parę dodatkowych działań. Łatwo to zrobić za pomocą skryptu awk:

```
{ sub("^0","",$3)
  sub("^0","",$4)
  users[$3,$4]=users[$3,$4]+$5
  load[$3,$4]=load[$3,$4]+$7
  m[$3,$4]++
```

```
#  print users[$3,$4], m[$3,$4]
}
END {
for (i=0; i<=23; i++)
{
  for (j=0; j<=59; j++)
  {
    print i, j, users[i,j]/m[i,j],
          m[i,j] >> "users1.srednia"
    print i, j, load[i,j]/m[i,j],
          m[i,j] >> "load1.srednia"
  }
}
}
```

Uśrednione dane z całej doby należy przeskalować. Ponieważ wykresy te są rysowane w terminalu tekstowym, wygodnie jest reprezentować jeden 30-minutowy okres w jednej kolumnie.

```
# Uruchomienie:
# awk -f skrypt.awk users1.srednia

{ a = int(NR/30)
  users[a]=users[a]+$2
}
END {
  for (i=0; i<=47; i++)
  {
    print i, users[i]/30 > "users30.srednia"
  }
}
```

W przeskalowanym zbiorze danych należy umieścić podstawowe parametry wykresu:

```
label liczba otwartych sesji, hoth.amu.edu.pl, data:
range 9 5 18 30
left ticks 5 10 15 20 25 30
bottom ticks 10 12 14 16 18
```

Skrypt awk używany do utworzenia wykresu:

```
# Jest to zmodyfikowana wersja programu graph z ksiazki
# A.V. Aho, B.W. Kernighan, P.J. Weinberger,
# The AWK Programming Language, Addison-Wesley, 1988.
# Na wejsciu: dane i specyfikacja wykresu
# Na wyjsciu: wykres danych w okreslonym obszarze ekranu

BEGIN {
# wymiary ramki, wysokosc i szerokosc
  wys = 24; szer = 50
# polozenie poczatku ukladu wspolrzednych
  ox = 4; oy = 2
  liczba = "^[-+]?([0-9]+[.]?[0-9]*|[.][0-9]+)"
  liczba = liczba "([eE][-+]?[0-9]+)?$"
}

# oznaczenie osi x
$1 == "nazwax" {
  sub(/^ *nazwax */, "")
  nazwax = $0
  next
}

# wspolrzedne na osi x
$1 == "osx" && $2 == "wspolrzedne" {
  for (i = 3; i <= NF; i++) xwspolrz[++nb] = $i
  next
}

# wspolrzedne na osi y
$1 == "osy" && $2 == "wspolrzedne" {
  for (i = 3; i <= NF; i++) ywspolrz[++nl] = $i
  next
}

# xmin ymin xmax ymax
$1 == "zakres" {
  xmin = $2; ymin = $3; xmax = $4; ymax = $5
  next
}

$1 == "wysokosc" { wys = $2; next }
```

```
$1 == "szerokosc" { szer = $2; next }
# para wspolrzednych
$1 ~ liczba && $2 ~ liczba {
  nd++
# policz, ile jest par wspolrzednych
  x[nd] = $1; y[nd] = $2
# opcjonalny znak do oznaczania punktow na wykresie
  ch[nd] = $3
  next
}

# pojedyncza liczba
$1 ~ liczba && $2 !~ liczba {
# policz, ile jest par liczb w zbiorze danych
  nd++
  x[nd] = nd; y[nd] = $1; ch[nd] = $2
  next
}

# rysowanie wykresu
# jezeli brak danych o zakresie liczb, oblicz zakres
END { if (xmin == "") {
    xmin = xmax = x[1]
    ymin = ymax = y[1]
    for (i = 2; i <= nd; i++) {
      if (x[i] < xmin) xmin = x[i]
      if (x[i] > xmax) xmax = x[i]
      if (y[i] < ymin) ymin = y[i]
      if (y[i] > ymax) ymax = y[i]
    }
  }
  ramka(); wspolrzedne(); nazwaosix();
  dane(); rysuj()
}

# obramowanie wykresu
function ramka() {
# dol
  for (i = ox; i < szer; i++) plot(i, oy, "-")
# gora
  for (i = ox; i < szer; i++) plot(i, wys-1, "-")
# lewa strona
  for (i = oy; i < wys; i++) plot(ox, i, "|")
```

```
# prawa strona
  for (i = oy; i < wys; i++) plot(szer-1, i, "|")
}

# zaznacz wspolrzedne na osiach
function wspolrzedne( i) {
  for (i = 1; i <= nb; i++) {
    plot(skalujx(xwspolrz[i]), oy, "|")
    splot(skalujx(xwspolrz[i])-1, 1, xwspolrz[i])
  }
  for (i = 1; i <= nl; i++) {
    plot(ox, skalujy(ywspolrz[i]), "-")
    splot(0, skalujy(ywspolrz[i]), ywspolrz[i])
  }
}

# centrowanie opisu na osi x
function nazwaosix() {
  splot(int((szer+ox-length(nazwax))/2),0,nazwax)
}

# rysowanie punktow na wykresie
function dane( i) {
  for (i = 1; i <= nd; i++)
    plot(skalujx(x[i]),skalujy(y[i]),
      ch[i]=="" ? "*" : ch[i])
}

# drukowanie wykresu na podstawie wartosci w tablicy
function rysuj( i, j) {
  for (i = wys-1; i >= 0; i--) {
    for (j = 0; j < szer; j++)
    printf((j,i) in array ? array[j,i] : " ")
    printf("\n")
  }
}

# skalowanie wartosci x
function skalujx(x) {
  return int((x-xmin)/(xmax-xmin)*(szer-1-ox)+ox+0.5)
}

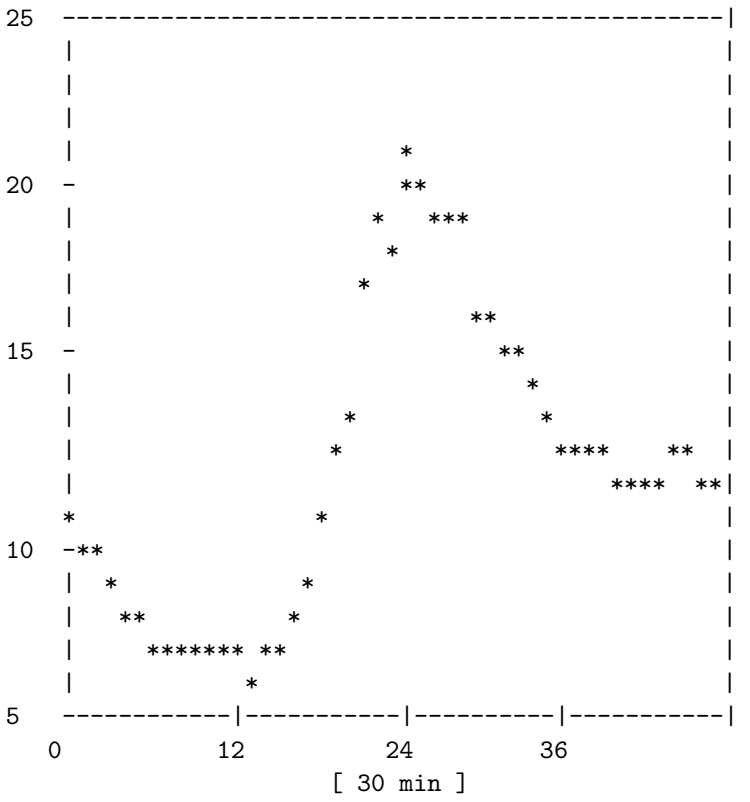
# skalowanie wartosci y
```

```
function skalujy(y) {
  return int((y-ymin)/(ymax-ymin)*(wys-1-oy)+oy+0.5)
}

# zapisz znak c w tablicy
function plot(x, y, c) { array[x,y] = c }

# zapisz lancuch w tablicy
function splot(x, y, s, i, n) {
  n = length(s)
  for (i = 0; i < n; i++)
  array[x+i, y] = substr(s, i+1, 1)
}
```

Oto wykres przedstawiający średnią liczbę użytkowników na serwerze `hoth.amu.edu.pl` w dniach od 26 marca do 5 maja 2004; na osi odciętych zaznaczono odcinki 30-minutowe:



## 5.16. Zamykanie procesów

Następujący skrypt drukuje numery procesów, których nazwa jest zgodna z wyrażeniem przekazanym do skryptu.

```
process_id=`ps -e \
          | grep $1 \
          | grep -v grep \
          | sed 's/^[ ]*//' \
          | cut -f1 -d ' '`

echo "$process_id"
```

W wyniku numer każdego procesu pojawia się w oddzielnym wierszu, np.

```
$ ./process_id.sh nazwa
980
1027
1154
```

Następujący skrypt zamyka procesy, których nazwa jest zgodna z parametrem przekazanym do skryptu. Przed przekazaniem numerów procesów do polecenia `kill` są one umieszczane w jednym wierszu

```
process_id=`ps -e \
          | grep $1 \
          | grep -v grep \
          | sed 's/^[ ]*//' \
          | cut -f1 -d ' '`

ile_procesow=`echo "$process_id" |\
wc -l | sed 's/^[ ]*//g' `
echo "ile procesow: $ile_procesow"

# Polecenie sed w tym wierszu zastepuje znaki nowego
# wiersza spacjami

kill -9 `echo "$process_id" |\
  sed -e :etykieta -e 'N; s/\n/ /g; tetykieta' `
```

## 5.17. Pobieranie plików z archiwów internetowych

Następujący skrypt wykonuje lokalną kopię drzewa katalogów znajdującego się na serwerze zdalnym

```
# ftpget.sh
komp="sunsite.icm.edu.pl"
kat="/pub/programming/books"
uzytkownik="anonymous"
email="kowalski@polska.com"

ftp -n $komp << KONIEC_POLECEN_FTP > ftp.tmp
user anonymous kowalski@polska.com
cd $kat
binary
ls -lR
bye
KONIEC_POLECEN_FTP

./ftpwrite.sh > ftp.polecenia

biezacy=`pwd`
./ftp.polecenia
cd $biezacy

# ftpwrite.sh
# Skrypt generuje polecenia, ktore
# zostana wykonane podczas polaczenia ftp
# do komputera zdalnego, tutaj jest to
# serwer sunsite.icm.edu.pl

komp="sunsite.icm.edu.pl"
kat="/pub/programming/books"
uzytkownik="anonymous"
email="kowalski@polska.com"

ftpdir=`cat ftp.tmp`
n=`echo "$ftpdir" | wc -l`
```

```
# Usuniecie ostatniego ukosnika
biezacykat=`pwd`
mkdir -p $biezacykat$kat

echo '#!/usr/bin/bash'
echo "ftp -n $komp << KONIEC"
echo "user $uzytkownik $email"
echo "binary"
echo "cd $kat"

i=0
while [ $i -lt $n ]
do
  i=$(($i+1))
  a=`echo "$ftpdir" | head -1`
  tmp=`echo "$ftpdir" | sed -n '2,$p'`
  ftpdir="$tmp"
  if [ -n "$a" ]
  then
    dir=`echo $a | sed -n 's/:$//p'`
    if [ -n "$dir" ]
    then
      echo "cd $kat"
      echo "cd $dir"
      echo "lcd $biezacykat$kat"
      cd "$biezacykat$kat"
      mkdir -p $dir
      echo "lcd $dir"
    else
      plik=`echo $a | grep '^-' | awk '{ print $8 }'`
      if [ -n "$plik" ]
      then
        echo "get $plik"
      fi
    fi
  fi
done

echo "bye"
echo "KONIEC"
```

Oto początkowy fragment pliku `ftp.tmp`:

```
.:
total 192
-rw-r--r--    1 10    27449 Nov 27 01:37 .mirror
-r--r--r--    1 10     3144 Oct 18  1992 comer.internetworking2.
  errata.gz
-r--r--r--    1 10    23981 Oct 18  1992 comer.internetworking3.
  src.tar.gz
-r--r--r--    1 10    17948 Feb 17  1993 dns.tar.gz
-r--r--r--    1 10    48932 Oct 18  1992 sa-book.tar.gz
drwxr-xr-x    2 10      512 Feb  7  2004 stevens.apue
drwxr-xr-x    2 10      512 Feb  7  2004 stevens.tcpipiv1
drwxr-xr-x    2 10      512 Feb  7  2004 stevens.tcpipiv2
drwxr-xr-x    2 10      512 Feb  7  2004 stevens.tcpipiv3
drwxr-xr-x    2 10      512 Feb  7  2004 stevens.unp
drwxr-xr-x    2 10      512 Feb  7  2004 stevens.unpv12e
drwxr-xr-x    2 10      512 Feb  7  2004 stevens.unpv22e

./stevens.apue:
total 928
-r--r--r--    1 10       38 Aug 22  2000 README
-r--r--r--    1 10      357 Nov 25  1995 apue.hpux.9.txt
-r--r--r--    2 10   162199 May 26  1998 apue.linux.tar.Z
-r--r--r--    2 10   162199 May 26  1998 apue.linux.tar.Z.1
-r--r--r--    1 10   195841 Oct  5  1998 apue.linux2.tar.Z
-r--r--r--    1 10   169181 May 25  1999 apue.linux3.tar.Z
-r--r--r--    1 10   169163 May 29  1992 apue.tar.Z
-r--r--r--    1 10    17695 May 25  1999 typos.apue.txt

./stevens.tcpipiv1:
total 200
-r--r--r--    1 10       42 Aug 22  2000 README
-r--r--r--    1 10    30449 May 31  1996 pocketguide1.ps
-r--r--r--    1 10    50855 Jun 16  1996 tcpipiv1.appe.update1.ps
-r--r--r--    1 10    67097 Dec  8  1993 tcpipiv1.tar.Z
-r--r--r--    1 10    27145 May 25  1999 typos.tcpipiv1.txt
```

A to zawartość pliku `ftpcommands.sh`:

```
# ftpcommands.sh
ftp -n sunsite.icm.edu.pl << KONIEC
user anonymous kowalski@polska.com
binary
cd /pub/programming/books
cd /pub/programming/books
cd .
```

```
lcd /home/tktos/pub/programming/books
lcd .
get .mirror
get comer.internetworking2.errata.gz
get comer.internetworking3.src.tar.gz
get dns.tar.gz
get sa-book.tar.gz
cd /pub/programming/books
cd ./stevens.apue
lcd /home/tktos/pub/programming/books
lcd ./stevens.apue
get README
get apue.hpux.9.txt
get apue.linux.tar.Z
get apue.linux.tar.Z.1
get apue.linux2.tar.Z
get apue.linux3.tar.Z
get apue.tar.Z
get typos.apue.txt
cd /pub/programming/books
cd ./stevens.tcpipiv1
lcd /home/tktos/pub/programming/books
lcd ./stevens.tcpipiv1
get README
get pocketguide1.ps
get tcpipiv1.appe.update1.ps
get tcpipiv1.tar.Z
get typos.tcpipiv1.txt
...
bye
KONIEC
```

# 6. Tcl

## 6.1. Wstęp

Autorem języka Tcl (Tool Command Language) jest John Ousterhout. Tcl powstał na przełomie lat 80. i 90. Początkowo był pomyślany jako język przeznaczony do zastosowań w systemach osadzonych.

Wartości zmiennych przydzielane są za pomocą instrukcji `set`:

```
#!/usr/bin/tclsh
set a 5
set x 1.6   x=1.6
# mozna tez tak:
set a 5; set x 1.6
```

Interpreter języka Tcl można uruchomić w trybie wsadowym, `./skrypt.tcl` lub interakcyjnym,

```
$ tclsh
% set a 5
5
% echo a
a
% echo $a
5
% exit
```

Każde polecenie składa się z jednego lub więcej słów (ang. *words*). Pierwsze jest nazwą instrukcji, a kolejne są jej argumentami. Słowa są rozdzielane spacjami i znakami tabulacji. Na przykład instrukcja `set 5 a` nadaje zmiennej o nazwie `5` wartość `a`.

Do obliczania wartości liczbowych służy instrukcja `expr`:

```
% expr 5/1.3
3.84615384615
```

Polecenie `eval` oblicza wartość skryptu zawartego w nawiasie klamrowym:

```
% eval { set a 5 }
5
```

Również instrukcje `if` i `while` akceptują skrypt Tcl w roli argumentu.

```
% lindex { kot pies owca } 2
owca
```

Wyrażenie w nawiasie klamrowym jest listą. Instrukcja `lindex` zwraca element listy o indeksie 2. Pierwszy element listy ma indeks zero.

```
%string length Ala
3
% string length "Ala ma kota."
12
```

Parser języka Tcl nie nadaje słowom żadnego znaczenia. Wykonuje to procedura obsługująca dane polecenie.

```
%set x 1
1

%set y x+3
x+3

%set y $x+3
1+3

%set y [expr $x+1]
4
```

Zapis `$x` oznacza wartość zmiennej `x`. W nawiasie kwadratowym może znajdować się wiele poleceń.

```
% set p Ala\ ma\ kota
Ala ma kota

% set p Ala\ \nma\ kota
Ala
ma kota
```

```
% set p Ala\ \n\ma\ kota
Ala
ma kota

% set p "Ala\n ma kota"
Ala
ma kota

% set p "Ala
ma kota"
Ala
ma kota

% set p "Ala powiedziala \"mam kota\""
Ala powiedziala "mam kota"

% set r {1zl=$0.33}
1zl=$0.33
```

W nawiasie klamrowym znaki specjalne tracą znaczenie. Nawias klamrowy oznacza *deferred evaluation* – znaki specjalne nie są intepretowane przez parser. Zajmuje się tym procedura polecenia.

Oto inny przykład – obliczenie silni `6!`.

```
set silnia 1; set i 6
while { $i > 0 } {
  set silnia [expr $silnia*$i]
  set i [expr $i-1]
}
```

Wartości zmiennych podstawiane są w każdym kroku pętli.

Komentarze w kodzie muszą być umieszczone w osobnych wierszach:

```
# poprawnie
set a 5 # niepoprawnie
```

## 6.2. Podstawianie

Są trzy sposoby podstawiania wartości zmiennych.

- Podstawianie zmiennej, np.
  `set a 10`
  `expr $a*5`
- Podstawianie poleceń, np.
  `set a 10 set b [expr $a*5]`
- Podstawianie z ukośnikiem \,
  `set tekst Ala\ ma\ kota.\nTo ladny kot.`

  Ukośnik zmienia znaczenie znaku. Na przykład, znak `$` zwykle poprzedza nazwę zmiennej i wówczas ta notacja oznacza wartość zmiennej. Aby znak był interpretowany dosłownie, należy wpisać `\$`.

  Ukośnik na końcu wiersza oznacza kontynuację polecenia w następnym wierszu.

**Reguły cytowania**

- `""`
  Wyłącza separatory słów i poleceń

  Wszystkie reguły podstawiania wartości działają bez zmian, np.

  ```
  set a 10
  set x "a = $a"; a^2 = [expr $a*$a]"
  ```

- `{}`
  Wyłącza prawie wszystkie znaki specjalne Wykonywane są jedynie podstawienia `\n`. Podstawienia nie są wykonywane przez parser Tcl. Znaki specjalne są przekazane do procedury realizującej polecenie jako część argumentu. Nawiasy mogą być zagnieżdżane.

Nazwy zmiennych lub ich wartości mogą być dowolnym ciągiem znaków. Dotyczy to także elementów i indeksów tablic.

```
set suma 0
foreach m {Sty Lut Mar Kwi Maj Cze
           Lip Sie Wrz Paz Lis Gru}{
```

```
            set suma [expr $suma+$zarobki($m)]
}
set zarobki(Sty) 2000
set zarobki(Lut) 1900
...
```

Wartości zmiennych można anulować poleceniem `unset`. W tym przykładzie anulowana jest wartość zmiennej `x` oraz wartość elementu tablicy o indeksie `Sty`.

```
unset x zarobki(Sty)
```

Tablice mogą być wielowymiarowe:

```
set a(1,1) 1.2
set a(1,2) 3.1
set i 1; set j 2
set b $a($i,$j)
```

Łańcuchy można łączyć za pomocą instrukcji `append`, np. `append x $tekst`.

Nazwy wszystkich indeksów tablicy można wydrukować instrukcją `array names nazwa_tablicy`. Rozmiar tablicy:
`array size nazwa_tablicy`. Oto przykład:

```
% set kraj(PL) Warszawa
% set kraj(DE) Berlin
% set kraj(IT) Rzym
```

Polecenie `array names kraj` daje wynik `PL DE IT`.

```
if { $x == "Jan Kowalski"}{
  ...
}
```

## 6.3. Arytmetyka

Domyślnie w działaniach na liczbach zmiennoprzecinkowych używanych jest 6 cyfr znaczących. Można to zmienić poleceniem
`set tcl_precision liczba_cyfr`, np. `set tcl_precision 17`.

Dostępne są następujące funkcje arytmetyczne: `acos`, `asin`, `atan`, `atan2`, `ceil`, `cos`, `cosh`, `exp`, `floor`, `fmod`, `hypot`, `log`, `log10`, `pow`, `sin`, `sinh`, `sqrt`, `tan`, `tanh`

## 6.4. Przetwarzanie łańcuchów

Grupa poleceń `string` ma następującą postać ogólną:

```
string [opcja] arg [arg1 ...]
```

- `string first x y`
  Zwraca indeks pierwszego wystąpienia ciągu `x` w `y` lub `-1`, jeśli `x` nie pojawia się w `y`
- `string last x y`
  Zwraca indeks ostatniego wystąpienia ciągu `x` w `y` lub `-1`, jeśli `x` nie pojawia się w `y`
- `string compare x y`
  Porównuje łańcuchy; zwraca `-1`, `0` lub `1`, podobnie jak funkcja `strcmp` języka C
- `string index lancuch n`
  Zwraca znak o indeksie $n$
- `string length lancuch`
  Zwraca długość łańcucha
- `string match wzorzec lancuch`
  Zwraca `1`, jeśli wzorzec pasuje do łańcucha
- `string range lancuch m n`
  Zwraca część łańcucha, od indeksu $m$ do indeksu $n$; parametr `end` w miejscu $n$ oznacza koniec łańcucha
- `string tolower lancuch`
  Konwersja łańcucha do małych liter
- `string toupper lancuch`
  Konwersja łańcucha do wielkich liter
- `string trim lancuch [znaki]`
  Usuwa wskazane znaki na początku i końcu łańcucha
- `string trimleft lancuch [znaki]`
  Usuwa wskazane znaki na początku łańcucha
- `string trimright lancuch [znaki]`
  Usuwa wskazane znaki na końcu łańcucha

- `string wordstart lancuch indeks`
  Zwraca indeks pierwszego znaku należącego do słowa zawierającego `indeks`, np.

  ```
  %string wordstart "Ala ma kota" 5
  4
  ```

- `string wordend lancuch indeks`
  Zwraca indeks pierwszego znaku po słowie zawierającym `indeks`, np.

  ```
  %string wordend "Ala ma kota" 5
  6
  ```

- `append zmienna wartosc1 wartosc2 ...`
  Dołączanie do `zmiennej` wartości innych zmiennych lub literałów.

  ```
  set x "Dzien "
  set y "dobry "
  set z "pani"
  append x $y $z
  ```

  Ostatnie polecenie jest równoważne `set x$y$z`

- `regexp [opcje] wyrazenie lancuch [zm] [zm1] ...`
  Wyszukiwanie dopasowania wyrażenia regularnego w łańcuchu. Instrukcja zwraca `1`, jeśli znaleziono dopasowanie, i `0` w przeciwnym razie.

  Opcja `-nocase` powoduje, że wielkie i małe litery nie są rozróżniane; `-indices` powoduje, że zmienne `zm1`, `zm2` i ewentualnie dalsze wskazują numer znaku (indeks) rozpoczęcia i zakończenia dopasowanych podciągów

- `regsub [opcje] wyrazenie lancuch podstawienie zmienna`
  Dopasowanie wyrażenia regularnego w łańcuchu i wykonanie podstawienia. Wynik jest zapisany w `zmiennej`

## 6.5. Tablice

Indeksami tablic mogą być dowolne łańcuchy znaków, nie tylko liczby całkowite.

Polecenie `arrays` ma następującą postać:

```
array opcja nazwatablicy [argumenty]
```

Oto lista opcji:

- `array exists nazwatablicy`
  zwraca `1`, jeżeli `nazwatablicy` jest tablicą, `0` w przeciwnym razie
- `array get nazwatablicy [wzorzec]`
  Zwraca listę par elementów zawierających indeksy i odpowiadające im wartości. Jeżeli podany jest opcjonalny `wzorzec`, zwracane są jedynie te elementy, których indeksy pasują do wzorca
- `array names nazwatablicy [wzorzec]`
  Zwraca listę indeksów tablicy. Jeżeli podano `wzorzec`, zwracane są tylko nazwy indeksów pasujące do wzorca
- `array set nazwatablicy lista`
  Elementy tablicy otrzymują wartości listy, gdzie każdy element listy jest parą indeks - wartość
- `array size nazwatablicy`
  Zwraca liczbę elementów tablicy

## 6.6. Działania na listach

Listy służą m.in do obsługi złożonych poleceń. Poszczególne elementy polecenia mogą też być listami i mogą zawierać zmienne.

- `set x { a b }`
  Definiowanie elementów listy
- `concat $x $y`
  Łączenie list, np. `set z [ concat $x $y ]`
- `lappend z X Y`
  Dołączanie elementów do listy

- `list $z`
  Drukowanie zawartości listy

- `lindex lista i`
  Zwraca element o indeksie `i`. Indeks pierwszego elementu listy jest równy `0`

- `linsert $z 1 X Y Z`
  Umieszczenie nowych elementów w miejscu o indeksie 1. Na przykład, niech lista `$z` składa się z trzech elementów `a b c`. Po wykonaniu powyższego działania `$z` ma postać `a X Y Z b c`.

  Inny przykład. `linsert $z 0  X Y  Z`. Tym razem nowe elementy pojawiają się na początku listy. Elementem listy może być inna lista.

- `llength $z`
  Drukowanie liczby elementów listy

- `lrange $z 1 4`
  Drukowanie elementów o indeksach od 1 do 4

- `lreplace $z 3 5`
  Usunięcie z listy elementów o indeksach od 3 do 5

- `lreplace $z 3 5 X Y Z`
  Zastąpienie elementów o indeksach od 3 do 5 elementami `X Y Z`

- `lsearch $z X`
  Drukowanie indeksu danego elementu. Opcje:
  `-exact`, dopasowanie dokładne
  `-glob`, dopasowanie dokładne lub częściowe
  `-regexp`, dopasowanie wyrażenia regularnego

- `lsort { Ola Ala Marek Jan}`
  `Ala Jan Marek Ola`
  Alfabetyczne sortowanie listy. Kolejność sortowania można zmieniać opcjami `-decreasing`, `-integer`

- `split $z xy`
  Podział łańcucha na fragmenty; podział jest dokonywany w miejscach pojawienia się znaku `x` lub `y`. Jeżeli `z` jest równa `axbxyxcydxye`, wynikiem działania jest `a b c d e`. Na przykład, jeżeli zmienna `z` jest równa `a/b/c`, wynikiem instrukcji `split $z /` jest lista `a b c`.

- `join { {} usr local lib } /`
  Połączenie elementów listy `{} usr local lib` znakiem `/`, co daje `/usr/local/lib`. Zapis `{}` oznacza listę pustą. Pominięcie jej jako pierwszego elementu w powyższym przykładzie dałoby wynik `usr/local/lib`. Instrukcję `join` można wykorzystać też do obliczenia sumy jej elementów:

  ```
  set z { 1.2 5.7 10 }
  expr [join $z +]
  16.9
  ```

## 6.7. Instrukcje sterujące

1. ```
   if( ... ) {
   }


   if ($x < 0) {
     } elseif ($x == 0) {
     } elseif ($x == 1) {
   }
   ```

2. ```
   set b ""
   set i [ expr [llength $a] -1]
   while { $i >= 0} {
     lappend b [lindex $a $i]
     incr i -1
   }
   ```

3. ```
   set b ""
   for {set i [expr [llength $a] - 1]}
     {$i >= 0} {incr i-1}{
     lappend b [lindex $a $i]
   }
   ```

4. ```
   set b ""
   foreach i $a {
     set b [linsert $b 0 $i]
   }
   ```

```
set b ""
foreach i $a {
  if {$i == "ZZZ"} break
  set b [linsert $b 0 $i]
}

set b ""
foreach i $a {
  if {$i == "ZZZ"} continue
  set b [linsert $b 0 $i]
}
```

5. 
```
switch $z {a {incr t1}
           b {incr t2}
           c {incr t3}
}

switch $z {a {incr t1} b {incr t2} c {incr t3}}

switch $z a {incr t1} b {incr t2} c {incr t3}

switch $x \
  a {incr t1}
  b {incr t2}
  c {incr t3}
}
```

Opcje instrukcji `switch`:

`-exact`
Wymaga dokładnego dopasowania łańcucha zmiennej do jednej z możliwych wartości

`-glob`
Dopasowanie w rodzaju stosowanego w powłokach Uniksa do nazw plików. Rozpoznawane są trzy typy metaznaków: `*` – oznacza dowolny ciąg znaków, `?` – jeden dowolny znak, `[abc]` – jeden znak ze zbioru znaków w nawiasie

`-regexp`
Dopasowane wyrażeń regularnych

`--`
Przydatne, jeśli dopasowywana wartość może rozpoczynać się znakiem `-`

```
set t1 0
set t2 0
set t3 0
foreach i $z {
  switch -regexp $i in {
    a {incr t1}
    ^[0-9]*$ {incr t2}
    default {incr t3}
  }
}

switch $z {
  a -
  b -
  c {incr t1}
  d {incr t2}
}
```

Wartość zmiennej `t1` zwiększa się w każdym z warunków `a`, `b` lub `c`.

## 6.8. Inne polecenia

- `eval`
  Przyjmuje dowolną liczbę argumentów. Służy do generowania poleceń. Na przykład:

  ```
  set polecenie "set a 5"
  eval $polecenie
  ```

  W tym przykładzie `eval` wymusza dodatkowy poziom analizy składniowej i wykonuje polecenie `set a 5`. Domyślnie tcl wykonuje jeden poziom podstawień, a potrzebnych może być więcej.

  Oto kolejny przykład, w którym celem jest anulowanie wartości zmiennych należących do listy `zmienne`.

```
foreach i $zmienne {
  unset $i
}
```

Instrukcja `unset $zmienne` nie daje oczekiwanego rezultatu. Aby osiągnąć cel, można użyć `eval unset $zmienne`. Ta ostatnia instrukcja jest równoważna

`eval [concat unset $zmienne]`.

- `source skrypt.tcl`
  Uruchomienie skryptu. Zwraca wynik ostatniego polecenia skryptu

## 6.9. Procedury

```
proc dodawanie {a b} {expr $a + $b}
plus 2 4
6
```

Wartość zwracaną przez procedurę można wskazać instrukcją `return`.

Zmienne globalne w procedurze są definiowane dyrektywą `global`, na przykład

```
proc P {} {
  global x y
  ...
}
```

Procedurę można wywołać bez argumentów.

W definicji procedury można podać wartości domyślne argumentów, np.

```
proc P {x {przyrost 1}} {
  expr $x + $przyrost
}
```

Polecenie `P 10 2` daje wynik `12`, a polecenie `P 10` daje wynik `11`.

Liczba argumentów procedury nie musi być ustalona z góry.

```
proc suma argumenty {
  set s 0
  foreach i $argumenty {
    incr s $i
  }
  return $s
}
```

Wywołanie: `suma 1 2 3 4`
Wynik: `10`.

Instrukcja `upvar` udostępnia wartości zmiennych lokalnych lub zmiennych zdefiniowanych w innych procedurach. Jest używana w celu przekazywania wartości przez odwołanie.

## 6.10. Instrukcje wejścia i wyjścia

- `open plik tryb_dostepu uprawnienia`
  Otwiera plik lub połączenie do potoku innego polecenia. Zwraca identyfikator kanału, który później wykorzystywany jest w innych poleceniach odczytu i zapisu Tryby dostępu: `a`, `a+`, `r`, `r+`, `w`, `w+`

- `close identyfikator_kanalu`
  Zamyka kanał

- `read [-nonewline] kanal [n_bajtow]`
  Wczytuje całość danych dostępnych na wejściu lub określoną liczbę bajtów. W przypadku użycia opcji `-nonewline` znak nowego wiersza nie jest przekazywany do programu, jeśli jest ostatnim znakiem na wejściu

- `gets kanal [zmienna]`
  Wczytuje łańcuch z `kanalu`. Łańcuch można przypisać `zmiennej`

- `puts [-nonewline] [kanal] lancuch`
  Zapisuje wynik do kanału. Domyślnie jest nim standardowe wyjście

- `format lancuch_formatu [arg1 arg2 arg3 ...]`
  Instrukcja działa tak, jak funkcja sprintf języka C

- `scan lancuch lancuch_formatu zmienna [zmienne]`
  Funkcja równoważna `sscanf` języka C

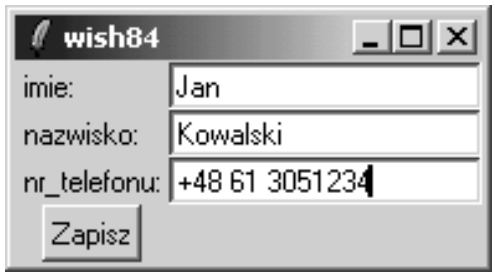
- `file opcja nazwa [arg1 arg2 arg3 ...]`
  Wykonuje operacje na plikach

## 6.11. Tk

Interfejs graficzny skryptu Tcl zwykle tworzony jest za pomocą modułu Tk. Narzędzie to stworzono specjalnie w tym celu. Później przystosowano Tk do interpreterów innych języków.

Oto prosty przykład, w którym dane kontaktowe dopisywane są do pliku `Kontakty`. Interfejs graficzny skryptu jest przedstawiony na rysunku 6.1.

```
#!/usr/bin/wish -f
# okno.tcl
set row 0
foreach i {imie nazwisko nr_telefonu e-mail} {
  label .$i-label -text "${i}:"
  entry .$i-entry -width 20
  grid .$i-label -row $row -column 0 -sticky w
  grid .$i-entry -row $row -column 1 -sticky "ew"
  incr row
}
button .b1 -text "Zapisz" -command {
  set plik_id [open "Kontakty" a 0600]
  puts $plik_id ""
  foreach i {imie nazwisko nr_telefonu e-mail} {
    puts $plik_id [.$i-entry get]
  }
  close $plik_id
}
grid .b1 -row $row -column 0
grid columnconfigure . 1 -weight 1
```




Rys. 6.1 Wynik działania skryptu `okno.tcl`

# 7. Python

## 7.1. Wstęp

Python powstał na przełomie lat 80. i 90. ubiegłego wieku. Jego autorem jest Guido van Rossum. Zaletami tego narzędzia są ogromne możliwości, modułowa struktura, przejrzysta składnia i obiektowy model programowania.

Dokładniejsze omówienie języka Python wymaga oddzielnej książki. W tym rozdziale spróbowano przede wszystkim przedstawić zarys jego składni i typowy dla niego styl programowania.

Interpreter języka Python można uruchomić w trybie interakcyjnym:

```
$ python
>>> print "abc"
abc
>>>
```

lub za pomocą skryptu. Powiedzmy, że skrypt ma następującą postać:

```
#!/usr/bin/python
print To jest Python
```

Zapiszmy go w pliku w pliku `a.py`. Po nadaniu uprawnień wykonywania, np. `chmod 700 a.py`, należy wykonać polecenie

```
$ ./a.py
To jest Python
```

W blokach instrukcji sterujących niezbędne są wcięcia:

```
for x in range(1,5):
  if x>3:
    print "x = ", x
  else:
    print "x <= 3"
```

Głębokość wcięć jest dowolna, ale musi być jednakowa dla całego bloku. Aby dołączyć odpowiedni moduł, należy użyć instrukcji `import`:

```
import math
x = 1.0
s = math.cos(x)
```

Wskazanie konkretnej funkcji znajdującej się w module skutkuje szybszym wykonaniem skryptu:

```
from math import cos
x = 1.0
s = cos(x)
```

Wyszukiwana jest tym razem tylko jedna nazwa, a nie dwie.

## 7.2. Typy liczbowe i wbudowane funkcje arytmetyczne

- `int`
  Liczby całkowite z przedziału `[-2147483648, 2147483647]`; liczby całkowite spoza tego zakresu są automatycznie konwertowane do typu `long` w celu uniknięcia błędu nadmiaru

- `long`
  Liczby całkowite dowolnej wielkości; jedynym ograniczeniem jest rozmiar dostępnej pamięci; notacja: `58L`

- `float`
  Liczby zmiennoprzecinkowe podwójnej precyzji

- `complex`
  Liczby zespolone, na przykład `(1.2+3.5j)`, `(0+2.7e-5j)`. Część rzeczywistą lub urojoną liczby zespolonej `z` oblicza się następująco: `z.real`, `z.imag`. Funkcja `complex(x,y)` zamienia liczby `x` i `y` do postaci zespolonej `(x+yj)`

Oto lista wbudowanych funkcji arytmetycznych

- `abs(x)`
  Wartość bezwzględna liczby `x`. Jeżeli argumentem jest liczba zespolona, wynik jest modułem liczby.

- `coerce(a,b)`
  Zwraca tuplę `(a, b)` argumentów przekształconych do jednakowego typu

- `divmod(a,b)`
  Dzielenie modulo. Wynik jest tuplą `(j, k)`, gdzie `j=a/b`, a `k` jest resztą `a%b`

- `pow(x,y)`
  `x` do potęgi `y`. Jeżeli podany jest trzeci argument, `pow(x, y, z)`, wynik jest równy `(x**) %z`

- `round(x)`
  Zaokrąglenie liczby do najbliższej liczby całkowitej

Inne funkcje arytmetyczne są dostępne w module `math`.

## 7.3. Typy sekwencyjne

- `str`
  `Łańcuchy znaków ASCII`

- `unicode`
  Łańcuchy znaków Unicode, na przykład `u'Ala ma kota'`, `u'\u03fa'`

- `list`
  Sekwencja wartości dowolnego typu

- `tuple`
  Sekwencja wartości dowolnego typu. Tupla nie podlega modyfikacji. Nie można z niej usunąć, zmienić ani dodać do niej żadnych elementów.

Funkcje wspólne dla wszystkich typów sekwencyjnych

- `s[i]`
  `i`-ty element ciągu

- `s[i:j]`
  Wycinek ciągu

- `s*n`
  Wynik jest sekwencją zawierającą `n` kopii sekwencji `s`

- `len(s)`
  Liczba elementów w `s`
- `min(s)`
  Najmniejsza wartość w `s`
- `max(s)`
  Największa wartość w `s`
- `x in s`
  Prawda, jeżeli przynajmniej jeden element sekwencji `s` jest równy `x`
- `x in s`
  Prawda, jeżeli żaden z elementów sekwencji `s` nie jest równy `x`
- `s+t`
  Połączenie sekwencji `s` i `t`
- `list(s)`
  Konwertuje sekwencję `s` na listę
- `tuple(s)`
  Konwertuje sekwencję `s` na tuplę
- `del s[i]` lub `del s[i:j]`
  Usunięcie elementu (nie dotyczy tupli) lub wycinka

## 7.4. Metody listy

- `L.append(x)`
  Dołącza element `x` na końcu listy `L`
- `L.count(x)`
  Liczba wystąpień `x` w `L`
- `L.index(x)`
  Najmniejszy indeks `i`, taki że element `L[i]` jest równy `x`
- `L.insert(i,x)`
  Umieszcza `x` przed elementem o indeksie `i`
- `L.pop([i])`
  Zwraca `i`-ty element i usuwa go z listy. Jeżeli brak argumentu, operacja dotyczy ostatniego elementu listy

- `L.remove(x)`
  Usunięcie pierwszego elementu, który jest równy `x`
- `L.reverse()`
  Odwrócenie kolejności elementów listy. Wartości nie są zwracane
- `L.sort([f])`
  Sortowanie za pomocą funkcji porównującej pary elementów. Brak argumentu oznacza sortowanie za pomocą reguł domyślnych.

## 7.5. Metody łańcuchów

- `S.capitalize()`
  Zmienia pierwszą literę `S` na dużą
- `S.center(szer)`
  Wyśrodkowanie napisu w polu długości `szer`
- `S.count(t [, i [, j]])`
  Liczba wystąpień ciągu `t` w `S`. Jeżeli podany jest drugi i trzeci argument, pod uwagę brany jest wycinek `S[i:j]`
- `S.endswith(t [, i [, j]])`
  Sprawdza, czy `S` kończy się łańcuchem `t`
- `S.expandtabs([tabsize])`
  Zamiana wszystkich spacji na znaki tabulacji
- `S.find(t [, i [, j]])`
  Znajduje pierwsze wystąpienie ciągu `t` w napisie. Jeżeli `t` nie występuje w `S`, zwraca `-1`
- `S.index(t [, i [, j]])`
  Znajduje pierwsze wystąpienie ciągu `t` w napisie. Jeżeli `t` nie występuje w `S`, zgłasza wyjątek
- `S.isalnum()`
  Prawda, jeśli wszystkie znaki są znakami alfanumerycznymi
- `S.isalpha()`
  Prawda, jeśli wszystkie znaki są literami
- `S.isdigit()`
  Prawda, jeśli wszystkie znaki są cyframi

- `S.islower()`
  Prawda, jeśli wszystkie znaki są małymi literami
- `S.isspace()`
  Prawda, jeśli `S` składa się wyłącznie z białych znaków (spacja, `\n`, `\r`, `\t`, `\f`, `\v`)
- `S.isupper()`
  Prawda, jeśli wszystkie znaki są wielkimi literami
- `S.join(L)`
  Wynik jest łańcuchem utworzonym z elementów listy `L` rozdzielonych ciągiem `S`
- `S.ljust(szer)`
  Wyrównanie `S` do lewej w polu o szerokości `szer`
- `S.lower()`
  Zamiana wszystkich wielkich liter na małe
- `S.lstrip([c])`
  Usunięcie z początku `S` wszystkich znaków należących do łańcucha `c`. Domyślnie usuwane są wszystkie znaki białe
- `S.replace(stary, nowy [, maks])`
  Zamiana wszystkich wystąpień ciągu `stary` na ciąg `nowy`. Można ograniczyć liczbę podstawień, podając trzeci argument
- `S.rfind(t [, i [, j]])`
  Indeks ostatniego wystąpienia `t` w `S`
- `S.rjust(szer)`
  Wyrównanie `S` do prawej w polu o szerokości `szer`
- `S.rstrip([c])`
  Usunięcie z końca `S` wszystkich znaków należących do łańcucha `c`. Domyślnie usuwane są wszystkie znaki białe
- `S.split([sep [, maks]])`
  Dzieli `S` w miejscach wystąpień łańcucha `sep` i zwraca listę powstałą po podziale. Domyślnie separatorami są białe znaki
- `S.splitlines([koncowki])`
  Podział `S` na listę wierszy. Jeżeli `koncowki` wynosi `1`, separatory wierszy są zachowane

- `S.startswith(t [, pocz[, koniec]])`
  Prawda, jeśli łańcuch rozpoczyna się ciągiem `t`

- `S.strip([c])`
  Usuwa początkowe i końcowe znaki należące do łańcucha `c`. Domyślnie usuwane są białe znaki

- `S.swapcase()`
  Zamienia wielkie litery na małe, a małe na wielkie

- `S.translate(tablica [, usun])`
  Zamienia poszczególne znaki zgodnie z porządkiem tablicy translacyjnej. Argument `tablica` jest łańcuchem 256 znaków. W wyniku tego w miejsce `x` podstawiony jest znak `tablica[ord(x)]`

- `S.upper()`
  Zamiana wszystkich liter na wielkie

- `S.zfill(szer)`
  Zwraca napis `S` długości tt szer z dopisanymi na początku zerami. Na przykład `'123'.zfill(5)` daje `'00123'`.

## 7.6. Słowniki

Słowniki są jedynym typem mapującym w języku Python. Obiekt mapujący nie jest uporządkowany. Elementami słownika są uporządkowane pary `(k, w)`, gdzie `k` jest kluczem, a `w` wartością. Klucze są niepowtarzalne i nie podlegają zmianie. Wartościami mogą być obiekty dowolnego typu.

Oto operacje, które można wykonywać na słownikach.

- `len(D)`
  Liczba elementów słownika `D`

- `D[k]`
  Wartość przyporządkowana kluczowi `k`

- `D[k] = w`
  Przypisanie wartości

- `del D[k]`
  Usunięcie elementu z kluczem `k`

- `D.clear`
  Usunięcie wszystkich elementów słownika
- `D.has_key(k)`
  Prawda, jeśli `D` zawiera element z kluczem `k`
- `D.items()`
  Lista tupli `(klucz, wartosc)` należących do `D`
- `D.keys()`
  Lista kluczy słownika
- `D.values()`
  Lista wartości słownika `D`, w tej samej kolejności, jak lista `D.keys()`
- `D.update(E)`
  Dołącza do `D` słownik `E`. Jeżeli w `E` istnieje element z tym samym kluczem co w `D`, wynik zawiera wartość ze słownika `E`
- `D.get(k [,x])`
  Zwraca `D[k]`, a jeśli brak elementu z takim kluczem, zwraca `x` lub `None`, jeśli brak argumentu `x`
- `D.setdefault(k [, x])`
  Zwraca `D[k]`, a jeśli brak elementu z takim kluczem, zwraca `x` i nadaje wartość `D[k]=x`
- `D.iteritems()`
  Zwraca iterator po elementach `D`
- `D.iterkeys()`
  Zwraca iterator po kluczach `D`
- `D.itervalues()`
  Zwraca iterator po wartościach `D`
- `D.popitem()`
  Zwraca losowo wybrany element `klucz, wartosc` i usuwa go ze słownika
- `x in D`
  Prawda, jeśli `x` jest kluczem w `D`
- `x not in D`
  Prawda, jeśli `x` nie jest kluczem w `D`

## 7.7. Instrukcje wejścia i wyjścia

Do otwierania plików służy funkcja wbudowana `open(nazwa [, tryb [, rozmiar_bufora]])`. Argument `nazwa` jest ścieżką dostępu do pliku, a `tryb` może składać się z trzech elementów. Pierwszy z nich może mieć wartość `r`, `w` lub `a`, co oznacza tryb odczytu, zapisu lub dodawania (*append*). Drugim może być znak `+`, co oznacza otwarcie do aktualizacji, a trzecim – `b`, co oznacza dostęp do danych binarnych,

```
plik = open('dane', wb)
```

Oto metody działające na plikach.

- `f.read([n])`
  Odczytuje cały plik i zwraca go w postaci łańcucha. Opcjonalny argument `n` określa liczbę bajtów przeznaczoną do odczytu

- `f.readline([n])`
  Odczytuje maksymalnie `n` znaków z bieżącego wiersza

- `f.readlines()`
  Odczytuje wszystkie wiersze pliku i zwraca je w postaci listy

- `f.write(s)`
  Zapisuje łańcuch `s` do pliku `f`

- `f.writelines(L)`
  Zapisuje listę łańcuchów `L` do pliku `f`

- `f.close()`
  Zamyka plik

- `f.seek(poz [, gdzie])`
  Przechodzi do nowego miejsca w pliku. Wartość `gdzie` określa sposób użycia wartości `poz`:
  `0` – umieszczenie wskaźnika na bajcie nr `poz` (działanie domyślne)
  `1` – przesunięcie o `poz` bajtów do przodu
  `2` – umieszczenie wskaźnika `poz` bajtów przed końcem pliku

- `f.truncate([poz])`
  Usuwa zawartość pliku od położenia `poz` do końca pliku. Domyślnie `poz` jest położeniem bieżącym

- `f.tell()`
  Zwraca bieżący wskanik do pliku

- `f.flush()`
  Opróżnia bufor wyjściowy

- `f.isatty()`
  Prawda, jeśli plik jest terminalem interaktywnym

## 7.8. Instrukcje sterujące

- ```
  if wyrazenie1:
    instrukcje
  elif wyrazenie2:
    instrukcje
  elif wyrazenie3:
    instrukcje
    ...
  else:
    instrukcje
  ```

  Klauzule `else` i `elif` nie są obowiązkowe.

- ```
  for i in z:
    instrukcje
  ```

  gdzie `z` jest zbiorem wartości.

- ```
  while wyrazenie:
    instrukcje
  ```

W celu przejścia do następnej iteracji pętli `while` lub `for`, zanim wykonane zostaną wszystkie instrukcje pętli, można użyć instrukcji `continue`. Wyjście z pętli `while` lub `for` może nastąpić za pomocą instrukcji `break`.
  ```

## 7.9. Definiowanie funkcji

Funkcje są definiowane instrukcją `def`:

```
def maks(x,y):
  if x>=y:
    m = x
  else:
    m = y
  return m
```

1. W argumentach funkcji można przekazywać listy, np.

   ```
   def f(x)
     print x

   l=['a', 'b', 'c']
   f(l)
   ```

2. W definicji funkcji można podać wartości domyślne, np.

   ```
   def f(x,y,z = 3.5):
   ```

   Parametr, mający wartość domyślną, można pominąć. Wywołanie tej funkcji może mieć następującą postać:

   ```
   f(x,y)
   ```

   Parametr z podaną wartością domyślną i wszystkie następujące po nim parametry są nieobowiązkowe.

   ```
   a = 5
   def f(x = a):
     print x
   a = 2
   # Powoduje wyswietlenie wartosci 5
   f()
   ```

3. Jeżeli przed ostatnim parametrem pojawia się gwiazdka `*`, funkcja może przyjmować dowolną liczbę argumentów,

```
def f(x, *parametry):
  for i in parametry:
    print i

x='abc'
f(x, 1, 2, 3)
```

4. Argumenty funkcji można też przekazywać jawnie, podając nazwę każdego parametru i podając jego wartość:

```
def f(d,m,r):
  print d,m,r

f(d=1, m='kwietnia', r=2004)
```

Jest to tzw. wywołanie z użyciem argumentów hasłowych. Wówczas kolejność parametrów nie ma żadnego znaczenia.

Jeżeli w tym samym wywołaniu funkcji występują argumenty pozycyjne (parametry funkcji w zwykłym rozumieniu) i hasłowe, najpierw muszą być podane argumenty pozycyjne.

Jeżeli ostatni parametr w definicji funkcji zaczyna się dwoma gwiazdkami **, wszystkie dodatkowe argumenty hasłowe (takie, które nie pasują do żadnej nazwy parametru) są umieszczane w słowniku i przekazywane do tej funkcji.

```
def f(**parametry):
  for i in parametry.keys():
    print k, parametry[k]

f(x=1, y="dwa", l=(1,2,3,4))
```

```
# tupla
a = (2.5, "abc")
# slownik
b = {'imie' : 'Jan', 'nazwisko' : 'Kowalski' }
f(*a, **b)
```

5. Parametry funkcji są przekazywane przez odwołanie. Jeżeli do funkcji przekazano obiekt zmienny (np. listę lub słownik) i zmodyfikowano go w niej, zmiana jest widoczna również w miejscu wywołania funkcji (jak dla zmiennej globalnej w przypadku zmiennych skalarnych). Na przykład:

```
a=[1,2,3,4,5]
def f(x):
  x[2]=1000000

f(a)
print a
```

   Wartości funkcji są zwracane za pomocą instrukcji `return`. Aby zwrócić wiele wartości, należy je umieścić w tupli, na przykład `return (a,b,5)`.

6. Limit rekurencyjnych wywołań funkcji:

```
sys.getrecursionlimit()
```

   Domyślnie maksymalną liczbą wywołań rekurencyjnych jest 1000. Można to zmienić za pomocą funkcji
   `sys.setrecursionlimit()`.

## 7.10. Proste przykłady

Moduł `os` udostępnia m.in. zmienne środowiskowe w postaci słownika. Kluczami słownika są nazwy zmiennych, `os.environ.keys()`.

**Przykłady**

1. 
```
import os
for z in os.environ.keys():
  print z,"------>",os.environ[z]
```

2. Oto przykład odczytania wartości zmiennej `PATH` i zmiany tej wartości przez dodanie do niej nowego członu.

```
import os
print "Wartosc zmiennej PATH:"
print os.environ["PATH"]
print
print "Dolaczenie katalogow do zmiennej PATH:"
os.environ["PATH"] = os.environ["PATH"] + ":/opt"
print os.environ["PATH"]
```

3. Dane o procesach

```
import os

# Drukowanie nr biezacego procesu
print os.getpid()

# Otwarcie pliku do zapisu
f = open("os2.d","w")

# drukowanie do pliku
print >>f, "Nr biezacego procesu: ", os.getpid()

# zamkniecie pliku
f.close()
```

4. Wykonywanie poleceń systemowych

```
os.system("ls -l")
os.system("cat os2.d")

# Usuniecie pliku
os.remove("os2.d")
# lub os.unlink("os2.d")
```

5. Drukowanie wszystkich elementów listy `sys.path`.

```
import sys
for s in sys.path:
  print s
```

6. Dane dotyczące pliku

```
import stat
import os, time
# Zwraca 10-elementowa tuple zawierajaca
# dane o pliku (rozmiar, czas modyfikacji,
# numer i-wezla itd.)
st = os.stat("sys.py")
print "tryb dostepu",\
      oct(stat.S_IMODE(st[stat.ST_MODE]))
print "rozmiar", st[stat.ST_SIZE]
print "data ostatniej modyfikacji",
      time.ctime(st[stat.ST_MTIME])
```

7. Otwarcie potoku do innego procesu

```
wykres=os.popen('gnuplot -persist', 'w')
wykres.write("""
...
""")
wykres.close()
```

W powyższym skrypcie znalazła się konstrukcja analogiczna do *here document*, rozwiązania znanego w powłoce Uniksa:

```
wykres << KONIEC
...
quit
KONIEC
```

8. Procesy uruchomione za pomocą instrukcji `os.system`, strumienia (`os.popen`) lub `commands.getstatusoutput` kończą się po zakończeniu instrukcji. Jak uruchomić proces w tle? Są dwie metody:

   (i) Uruchomienie polecenia w tle – znak `&` na zakończenie polecenia w Uniksie, a w systemie Windows wykonanie polecenia za pomocą programu `start`

   (ii) Uruchomienie polecenia `os.system` w osobnym wątku

9. Tworzenie ścieżki dostępu do pliku metodą niezależną od systemu operacyjnego:

```
plik=os.path.join(os.pardir,'dane','plik1')
shutil.copy(plik,os.curdir)
```

W powyższym skrypcie `os.pardir` oznacza katalog nadrzędny (skrót od słów *parent directory* (`..`), a katalog `os.curdir` - katalog bieżący oznaczany kropką. W systemie Unix ścieżka do pliku ma postać `../dane/plik1`, a w systemie Windows `..\dane\plik1`.

10. Uruchomienie procesu w tle w sposób niezależny od systemu operacyjnego

`os.name` – nazwa systemu operacyjnego

`sys.platform` – identyfikator platformy

```
# os.name - nazwa systemu operacyjnego
# sys.platform - identyfikator platformy
if os.name == 'posix':
  os.system(polecenie + '&')
elif sys.platform[:3] == 'win':
  os.system('start ' + polecenie)
else
  os.system(polecenie)
```

11. Przejście rekursywne po drzewie katalogowym i drukowanie plików o rozmiarze przekraczającym 1000 bajtów.

```
import os
def rozmiar(arg,katalog,pliki):
  for plik in pliki:
    os.path.join(katalog,plik)
    p=os.path.join(katalog,plik)
    if os.path.isfile(p):
# Sprawdzamy rozmiar pliku
      rozmiar=os.path.getsize(p)
      if rozmiar > 1000:
        print '%.2fKb %s' % (rozmiar/1000.0,p)
# Katalog, od ktorego nalezy rozpoczac
```

```
# dzialanie skryptu
katpocz=os.environ['HOME']
# Metoda walk powoduje przejscie
# po calym drzewie katalogowym wywodzacym sie
# z katalogu poczatkowego
os.path.walk(katpocz, rozmiar, None)
```

12. Prosta arytmetyka

```
# funkcja range(i,j) zwraca liczby calkowite
# w zakresie od i+1 do j
for i in range(3,5):
  print i
  j=1
  while j<=3:
    print j
    # Mozna tez tak: j+=1
    j=j+1

if j>3:
  print "wynik %5d %10s\n" % (j," > 3")
else:
  print j, "<=3"
print "koniec skryptu"
```

Wynik skryptu:

```
3
1
2
3
4
1
2
3
wynik     4        > 3

koniec skryptu
```

13. 
```
a=1
i=1
while a <= 10:
  b = 2*a
  print b
  a+=1
  while i <= 2:
   c=2*i
   print c
   i=i+1
```

    Wynik:

```
2
2
4
4
6
8
10
12
14
16
18
20
```

14. Instrukcja `return` zwraca wartości zmiennych do skryptu, z którego wywołano funkcję.

```
a = 1.0
def f(x,y):
  x = 2*x
  w = a*x*y
  return w,x
for j in range(1,10):
  print f(j,5)
  print j
```

Wynik:

```
(10.0, 2)
1
(20.0, 4)
2
(30.0, 6)
3
(40.0, 8)
4
```

15. Proste operacje na listach

```
# Definiowanie listy
l = [1, 3.912, 2.02, 10, -4.5]
# Liczba elementow listy
print len(l)
# Najmniejszy element listy
print min(l)
# Najwiekszy element listy
print max(l)
# Tworzenie nowej listy
wartosci = [ float(z) for z in l ]
print "min ", min(wartosci)
print "max ", max(wartosci)
# Dolaczenie elementu
l.append(23.15)
print l
```

Wynik:

```
5
-4.5
10
min  -4.5
max  10.0
[1, 3.9119999999999999, 2.02, 10,
-4.5, 23.149999999999999]
```

16. Listy mogą być zagnieżdżone – elementami list mogą być inne listy

```
# lista1.py
janek = ["601999000", "student"]
franek = ["602xxxyyy", "student"]
grzes = ["?", "?"]
l = [janek, franek, grzes]
print l
# Odwolanie do elementow list zagniezdzonych
s = l[0][0] + " " + l[0][1]
print s
```

Wynik:

```
[['601999000', 'student'], ['602xxxyyy', 'student'],
['?', '?']]
601999000 student
```

17. Metoda `seek` powoduje przesunięcie bieżącego wskaźnika w pliku o 10 miejsc. Jeżeli drugi argument ma wartość

`0` (domyślnie) – przesunięcie obliczone jest względem początku pliku

`1` – przesunięcie jest obliczone względem bieżącego położenia

`2` – przesunięcie jest obliczone względem końca pliku

```
# Otwarcie pliku do odczytu
p1 = open("dane.d","r")
# Otwarcie pliku do zapisu
p2 = open("danewyj.d","w")
i=0
# Metoda readlines() wczytuje wszystkie wiersze
# i zwraca je w postaci listy
for w in p1.readlines():
  i = i+1
```

```
  print i, w
# zapis do pliku
  p2.write(w)
# zamkniecie pliku
p2.close()
p1.seek(5,0)
b = p1.read(8)
print b
p1.seek(5,1)
b = p1.read(6)
print b
p1.seek(-10,2)
b = p1.read(8)
print b
p1.close()
```

Niech plik `dane.d` ma postać:

```
Ala ma kota.
Ola ma psa.
Piotr ma i kota i psa.
```

Otrzymany wynik:

```
1 Ala ma kota.
2 Ola ma psa.
3 Piotr ma i kota i psa.
a kota.

a psa.
ta i psa
```

18.
```
p1 = open("dane.d","r")
# lista ktorej elementami sa wiersze pliku dane.d
wiersze = p1.readlines()
l = len(wiersze[1])
w1 = wiersze[1][0:l-1]
l = len(wiersze[2])
w2 = wiersze[2][0:l-1]
```

```
suma12 = w1 + " " + w2
print suma12
s = 3*(w1 + " ")
print s
p1.close()
```

Oto wynik dla pliku wejściowego z poprzedniego przykładu:

```
Ola ma psa. Piotr ma i kota i psa.
Ola ma psa. Ola ma psa. Ola ma psa.
```

19. Zapis wszystkich elementów listy do pliku

```
p1 = open("dane.d","r")
p2 = open("danewyj.d","w")
w = p1.readlines()
p2.writelines(w)
p2.close()
p1.close()
```

20. Przekazywanie parametrów do skryptu

```
import sys, os
print sys.argv[0]
print sys.argv[1]
for i in range(len(sys.argv)):
  print i, sys.argv[i]
sciezka = os.environ["PATH"]
print sciezka
# z - slownik (tablica asocjacyjna)
# zestaw: klucz - wartosc
z = os.environ
print z
```

Element tablicy `sys.argv[0]` jest nazwą pliku, z którego uruchomiono skrypt. Kolejne elementy tej tablicy są parametrami przekazanymi do skryptu. Słownik (tablica asocjacyjna) `os.environ` zawiera pary nazwa – wartość dla wszystkich zmiennych środowiskowych.

21. 
```
import sys, os
sciezka = os.environ["PATH"]
print sciezka
# z - slownik (tablica asocjacyjna)
# zestaw: klucz - wartosc
# petla po kluczach:
for z in os.environ.keys():
  print z, "--->", os.environ[z]
# items to pary (klucz,wartosc)
for w in os.environ.items():
  print w
# petla po wartosciach slownika
for w in os.environ.values():
  print w
```

22. 
```
import sys
f=open(sys.argv[1])
# tresc to lista zawierajaca cala zawartosc pliku,
# ktorego nazwa jest przekazana do skryptu jako
# argument: ./skrypt.py plik_wej
tresc=f.readlines()
f.close()
# Petla po wszystkich elementach listy
for i in tresc:
  print i
```

23. 
```
# lista
klucze = ["tel", "miasto", "ulica",
          "nr domu", "zawod"]

# tupla
klucze1 = ("tel", "miasto", "ulica",
           "nr domu", "zawod")

# slownik
adam_kowalski = {
     "tel" : "601000111",
     "miasto" : "Poznan",
     "ulica" : "Akademicka",
     "nr domu" : "100",
}
```

```
ewa_nowak = {
  "tel" : "nieznany",
  "miasto" : "Poznan",
  "ulica" : "brak",
  "nr domu" : "brak" ,
  "zawod" : "konsultant"
}

print adam_kowalski["tel"]

# petla po elementach tupli
for z in klucze1:
# print "%20s %4s %-30s" % \
#        (z, " : ", adam_kowalski[z])
# Drukowanie elementow slownika
  if adam_kowalski.has_key(z):
    print
  else:
    print "brak klucza ", z

# Drukowanie wszystkich par danych
for z in adam_kowalski.items():
  print z
```

## 7.11. Wyrażenia regularne

- `tekst`
  Ciąg znaków `tekst`
- `.`
  Dowolny znak
- `^`
  Początek ciągu znaków
- `$`
  Koniec ciągu znaków

- `*`
  Zero lub więcej powtórzeń poprzedniego wyrażenia
- `+`
  Jedno lub więcej powtórzeń poprzedniego wyrażenia
- `?`
  Zero lub jedno powtórzenie poprzedniego wyrażenia
- `*?`
  Zero lub więcej powtórzeń poprzedniego wyrażenia, możliwie najmniej
- `+?`
  Jedno lub więcej powtórzeń poprzedniego wyrażenia, możliwie najmniej
- `??`
  Zero lub jedno powtórzenie poprzedniego wyrażenia, możliwie najmniej
- `{m,n}`
  Od `m` do `n` powtórzeń ostatniego wyrażenia, jak najwięcej
- `{m,n}?`
  Od `m` do `n` powtórzeń poprzedniego wyrażenia, jak najmniej
- `w1|w2`
  Alternatywa, wyrażenie `w1` lub `w2`
- `(?#wyrazenie)`
  Komentarz
- `(?=wyrazenie)`
  Poprzednie wyrażenie, jeżeli następuje po nim wzorzec podany w nawiasie, `r'Dzien (?=dobry)'` oznacza `'Dzien '` jedynie wtedy, kiedy w dalszej części ciągu znaków pojawia się `dobry`
- `?!=wyrazenie`
  Poprzednie wyrażenie, jeżeli nie następuje po nim wzorzec podany w nawiasie, `r'Dzien (?!=dobry)'` oznacza `'Dzien '` wtedy, kiedy w dalszej części ciągu znaków nie pojawia się `dobry`
- `?<=wyrazenie`
  Podane wyrażenie, jeżeli pojawia się przed nim ciąg podany w nawiasie, `r'(?<=xyz.)com` oznacza `E.com`, o ile poprzedza go ciąg `xyz.`

- `?<!=wyrazenie`
  Podane wyrażenie, jeżeli nie pojawia się przed nim ciąg podany w nawiasie, `r'(?<!=xyz.)com` oznacza `E.com`, o ile nie poprzedza go ciąg `xyz.`

- `\numer`
  Tekst dopasowany przez grupę o tym numerze

- `\A`
  Początek ciągu znaków

- `\B`
  Puste znaki nie na początku ani nie na końcu ciągu znaków

- `\b`
  Puste znaki na początku lub na końcu ciągu znaków

- `\D`
  Znak inny niż cyfra, `'r\[0-9]\`

- `\d`
  Cyfra, `r\[0-9]\`

- `\S`
  Znak inny niż biały znak; równoważny zapis:
  `^r'[\t\n \r\f \v]\`

- `\s`
  Biały znak; równoważny zapis: `r'[\t\n\r\f \v]\`

- `\W`
  Znak inny niż alfanumeryczny

- `\w`
  Znak alfanumeryczny

- `\Z`
  Koniec ciągu znaków

- `\\`
  Ukośnik odwrotny

**Przykłady**

1. Wyszukiwanie wierszy pliku, którego nazwa, składająca się wyłącznie z wielkich liter, jest przekazana do skryptu jako argument.

```
import re, sys
plik = open(sys.argv[1],"r")
wzorzec = r'^[A-Z]+$'
i = 0
for wiersz in plik.readlines():
  i = i+1
  x = re.search(wzorzec,wiersz)
  if x:
    print i, wiersz
```

2. Przeszukiwanie całej gałęzi katalogów w poszukiwaniu wierszy w plikach zawierających przekazany wzorzec. Polecenie uruchomienia skryptu ma postać `./skrypt.py katalog wzorzec`, gdzie `katalog` jest ścieżką do katalogu, od którego należy rozpocząć przeszukiwanie, a `wzorzec` jest poszukiwanym wzorcem.

```
import os, re, sys
def funkcja(arg,katalog,pliki):
  for plik in pliki:
    os.path.join(katalog,plik)
    p = os.path.join(katalog,plik)
    if os.path.isfile(p):
      p1 = open(p,"r")
      i = 0
      for wiersz in p1.readlines():
        i = i+1
        x = re.search(wzorzec,wiersz)
        if x:
          print p, i, wiersz
wzorzec = sys.argv[2]
os.path.walk(sys.argv[1], funkcja, wzorzec)
```

## 7.12. Moduł interfejsu graficznego Tkinter

Jedną z metod tworzenia graficznego interfejsu użytkownika jest wykorzystanie modułu Tk, który powstał w celu obsługi interfejsu graficznego języka Tcl. Odpowiedni moduł języka Python nosi nazwę `Tkinter`. Dokładna dokumentacja tego modułu jest dostępna w Internecie.

1. Najprostszy przykład utworzenia okna

```
# tk1.py
from Tkinter import *
# Tutaj root jest glownym oknem,
# w ktorym powstaje caly interfejs
root=Tk()
w=Label(root, text="Pierwsze okno")
w.pack()
root.mainloop()
```


Rys. 7.1 Wynik skryptu `tk1.py`

Jeżeli zamiast instrukcji `from Tkinter import *` użyć `import Tkinter`, ten sam skrypt będzie miał następującą postać:

```
import Tkinter
root=Tkinter.Tk()
w=Tkinter.Label(root, text="Pierwsze okno")
w.pack()
root.mainloop()
```

2. Definicja klasy

```
from Tkinter import *

class App:
  def __init__(self,master):
    frame=Frame(master)
    frame.pack()
    self.button=Button(frame, text="Zakoncz",\
    fg="red", command=frame.quit)
    self.button.pack(side=LEFT)
    self.hej=Button(frame, text="Hej",\
    command=self.f_hej)
    self.hej.pack(side=LEFT)

# Metoda f_hej obsluguje klikniecie przycisku
  def f_hej(self):
    print "Dzien dobry"

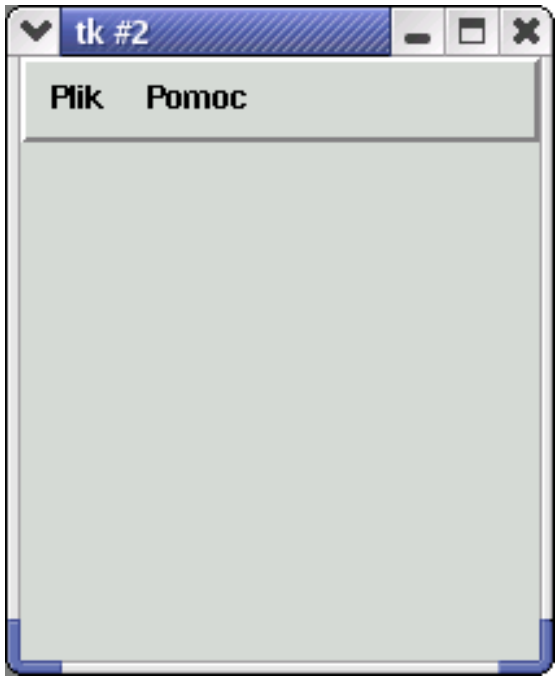
root=Tk()
app=App(root)
root.mainloop()
```

3. 
```
# tk2.py
from Tkinter import *

def polecenie():
  print "wykonuje polecenie"
root = Tk()
menu = Menu(root)
root.config(menu=menu)
# Menu rozwijalne
filemenu = Menu(menu)
# Polecenia menu
menu.add_cascade(label="Plik", menu=filemenu)
filemenu.add_command(label="Nowy",\
command=polecenie)
filemenu.add_command(label="Otworz...",\
command=polecenie)
filemenu.add_separator()
```

```
filemenu.add_command(label="Zakoncz",\
command=polecenie)

helpmenu = Menu(menu)
menu.add_cascade(label="Pomoc", menu=helpmenu)
helpmenu.add_command(label="Temat",\
command=polecenie)
mainloop()
```




Rys. 7.2 Wynik skryptu `tk2.py`

4. 
```
# tk3.py
from Tkinter import *

def polecenie():
  print "wykonuje polecenie"

root = Tk()
pasek = Frame(root)
```

```
b = Button(pasek, text="nowy", width=6,\
command=polecenie)
b.pack(side=LEFT, padx=2, pady=2)

b = Button(pasek, text="open", width=6,\
command=polecenie)
b.pack(side=LEFT, padx=2, pady=2)

pasek.pack(side=TOP, fill=X)
mainloop()
```

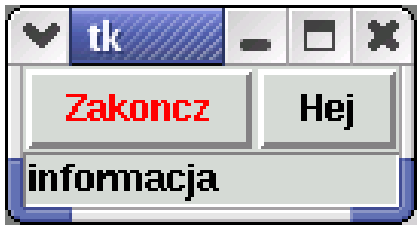

Rys. 7.3 Wynik skryptu `tk3.py`

5. 
```
# tk4.py
from Tkinter import *

class App:
  def __init__(self,master):
    frame=Frame(master)
    frame.pack()
    self.button=Button(frame, text="Zakoncz",\
    fg="red", command=frame.quit)
    self.button.pack(side=LEFT)
    self.hej=Button(frame, text="Hej",\
    command=self.f_hej)
    self.hej.pack(side=LEFT)
    status = Label(master, text="informacja",\
    bd=1, relief=SUNKEN, anchor=W)
    status.pack(side=BOTTOM, fill=X)

  def f_hej(self):
    print "Dzien dobry"

root=Tk()
app=App(root)
root.mainloop()
```




Rys. 7.4 Wynik skryptu `tk4.py`

6. Tworzenie widgetu o własnościach innych niż domyślne:

```
class MojTekst(Text):
  def __init__(self, master, **kw):
    apply(Text.__init__, (self, master), kw)
    self.bind("<Return>", lambda e: "break")
```

7. Przykład, w którym śledzone jest położenie kursora myszy. Program reaguje na zmianę tego położenia.

```
# tk5.py
from Tkinter import *

root = Tk()

def polecenie(event):
  print "klikniecie punktu o wspolrzednych",\
  event.x, event.y
def rozmiar(event):
  print "rozmiar okna", event.width, event.height
def a(event):
  print "a!", event.char
def enter(event):
  print "kursor wszedl w obszar ramki"
def leave(event):
  print "kursor opuscil obszar ramki"
def z(event):
  print "z"
```

```
frame = Frame(root, width=100, height=100)
frame.bind("<Button-1>", polecenie)
frame.bind("<Configure>", rozmiar)
frame.bind("<Key>", a)
frame.bind("<Enter>", enter)
frame.bind("<Leave>", leave)
frame.bind("z", z)
frame.pack()

root.mainloop()
```

8. Utworzenie dwóch okien.

```
# tk6.py
from Tkinter import *

root = Tk()
w1 = Label(root, text="Okno root")
w1.pack()

top = Toplevel()
w2 = Label(top, text="Okno top")

root.mainloop()
```

9. Przykład realizacji prostych operacji graficznych za pomocą metod klasy `Canvas` (rysunek 7.5). Początek układu współrzędnych obrazu na ekranie znajduje się w lewym górnym rogu ekranu.

```
# tk11.py
from Tkinter import *
from math import *

# wspolrzedne punktu w lewym dolnym rogu wykresu
ox = 50; oy = 350

xlen = 200
ylen=100
```

```
# para wspolrzednych oznaczajaca lewy dolny
# i prawy gorny wierzcholek obrazu
p = ox, oy, ox+xlen, oy-ylen

t1 = ox, oy-ylen
t2 = ox+xlen, oy-ylen
t3 = ox+0.5*xlen, oy-2*ylen
trojkat = t1, t2, t3

# srodek owalu (tutaj kolo)
ovx = 280; ovy = 20
srednica = 30
koloxy = ovx, ovy, ovx+srednica, ovy+srednica
srodekx = ovx+srednica/2
srodeky = ovy+srednica/2
dlug_prom = 30
max = 12

root = Tk()
# Miejsce zarezerwowane na obraz ma szerokosc
# 400 pikseli i wysokosc 400 pikseli
c = Canvas(root, height=400, width=400)
l1 = c.create_rectangle(p)
l11 = c.create_rectangle(p, fill="yellow",
      width=2)
l2 = c.create_polygon(trojkat, fill="red",
     outline="black", width=2)
slonce = c.create_oval(koloxy,
         fill="orange", outline="")

for i in range(1,max+1):
  x = srodekx + dlug_prom*cos(2*pi*i/max)
  y = srodeky + dlug_prom*sin(2*pi*i/max)
  promien = c.create_line(srodekx, srodeky,
                          x, y, fill="orange")

chmurka = c.create_oval(20, 20, 180, 70,
  fill="blue", outline="")

c.pack()
```

```
def sciana_zolty():
  c.create_rectangle(p, fill="yellow", width=2)

def sciana_rozowy():
  c.create_rectangle(p, fill="pink", width=2)

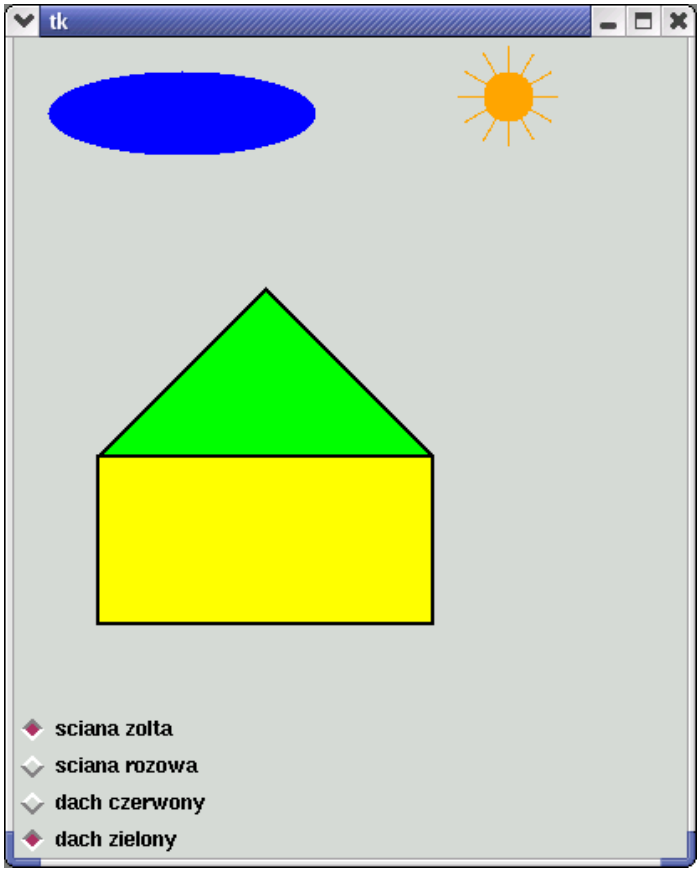
v1 = StringVar()
v1.set("yellow")
b1 = Radiobutton(root, command=sciana_zolty,
     text="sciana zolta", variable=v1,
     value="yellow")
b2 = Radiobutton(root, command=sciana_rozowy,
     text="sciana rozowa", variable=v1,
     value="pink")
b1.pack(anchor=W)
b2.pack(anchor=W)

def dach_czerwony():
  c.create_polygon(trojkat, fill="red",
                   outline="black", width=2)

def dach_zielony():
  c.create_polygon(trojkat, fill="green",
                   outline="black", width=2)

v2 = StringVar()
v2.set("red")
b10 = Radiobutton(root, command=dach_czerwony,
      text="dach czerwony", variable=v2,
      value="red")
b20 = Radiobutton(root, command=dach_zielony,
      text="dach zielony", variable=v2,
      value="green")
b10.pack(anchor=W)
b20.pack(anchor=W)

root.mainloop()
```




Rys. 7.5 Wynik skryptu `tk11.py`

10. Skrypt wyświetla zawartość pliku w przewijalnym polu tekstowym, rysunek 10.

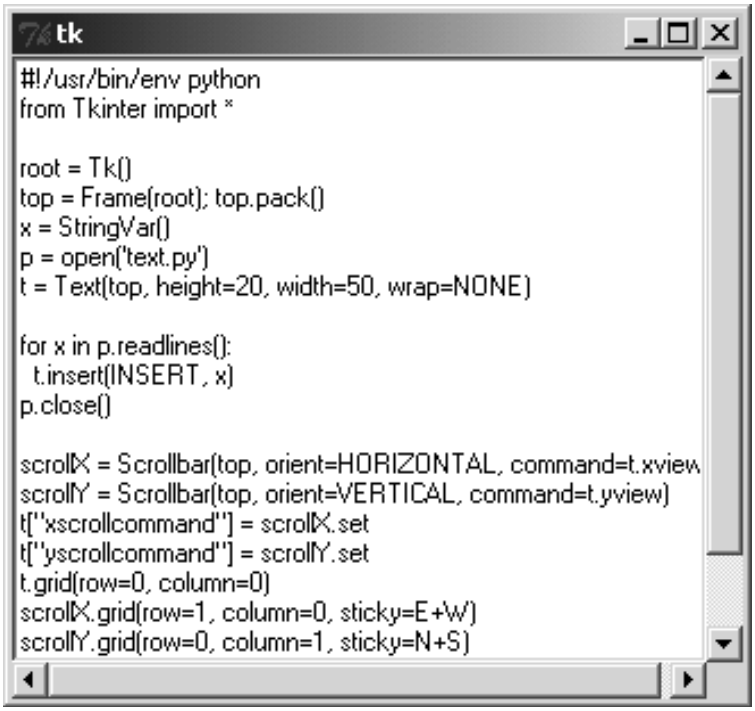
```
# tekst.py
from Tkinter import *

root = Tk()
top = Frame(root); top.pack()
x = StringVar()
p = open('tekst.py')
t = Text(top, height=20, width=50, wrap=NONE)
```

```
for x in p.readlines():
  t.insert(INSERT, x)
p.close()

scrollX = Scrollbar(top, orient=HORIZONTAL,
          command=t.xview)
scrollY = Scrollbar(top, orient=VERTICAL,
          command=t.yview)
t["xscrollcommand"] = scrollX.set
t["yscrollcommand"] = scrollY.set
t.grid(row=0, column=0)
scrollX.grid(row=1, column=0, sticky=E+W)
scrollY.grid(row=0, column=1, sticky=N+S)

root.mainloop()
```




Rys. 7.6 Wynik skryptu `tekst.py`

11. Prosty kalkulator funkcyjny.

```
from Tkinter import *
from math import *

root = Tk()
top = Frame(root); top.pack()
Label(top, text='Define f(x):').pack(side='left')

# Pole tekstowe szerokosci 16 znakow,
# w ktorym wpisywane sa wartosci funkcji
f_entry = Entry(top, width=16)
f_entry.pack(side='left')
f_entry.insert('end', 'x')

Label(top, text='  x =').pack(side='left')

# Pole tekstowe do wpisywania wartosci x
x_entry = Entry(top, width=10)
x_entry.pack(side='left')
x_entry.insert('end', '0')

# lancuch przedstawiajacy wartosc funkcji
s_label = Label(top, width=12)

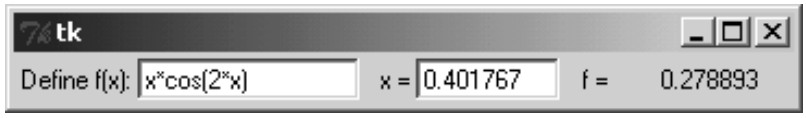
def oblicz(event=None):
    # odczyt lancucha znakow
    # przedstawiajacego funkcje
    f_txt = f_entry.get()
    x = float(x_entry.get())

    # obliczenie wartosci funkcji
    wynik = eval(f_txt)
    global s_label
    s_label.configure(text='%g' % wynik)

x_entry.bind('<Return>', oblicz)
Button(top, text='  f = ', relief='flat',
```

```
       command=oblicz).pack(side='left')
s_label.pack(side='left')

def quit(event=None): root.quit()
root.bind('<q>', quit)
root.mainloop()
```




Rys. 7.7 Kalkulator funkcyjny. Po wpisaniu postaci funkcji oraz wartości zmiennej x należy nacisnąć klawisz `Enter`, aby wyświetlić wynik

## 7.13. Wykres funkcji

1. Prosty przykład

```
from Tkinter import *
from math import *

ox = 50; oy = 450; xlen = 400; ylen = 400

root = Tk()
top = Frame(root); top.pack()
Label(top, text=' f(x) = ').grid(row=1,
      column=0, sticky = E)
ftxt = StringVar()
f_entry = Entry(top, width=30, textvariable=ftxt)
f_entry.grid(row=1, column=1)
f_entry.insert('end', 'x')

Label(top, text=' x_min = ').grid(row=2,column=0)
xmin = StringVar()
x_min = Entry(top, width=12, textvariable=xmin)
x_min.grid(row=2, column=1, sticky = W)
x_min.insert('end', '-10')
Label(top, text=' x_max = ').grid(row=3,column=0)
xmax = StringVar()
```

```
x_max = Entry(top, width=12, textvariable=xmax)
x_max.grid(row=3, column=1, sticky = W)
x_max.insert('end', '10')

Label(top, text=' y_min = ').grid(row=4,column=0)
ymin = StringVar()
y_min = Entry(top, width=12, textvariable=ymin)
y_min.grid(row=4, column=1, sticky = W)
y_min.insert('end', '-10')
Label(top, text=' y_max = ').grid(row=5,column=0)
ymax = StringVar()
y_max = Entry(top, width=12, textvariable=ymax)
y_max.grid(row=5, column=1, sticky = W)
y_max.insert('end', '10')

punkty = IntVar()
pkt = Scale(top, from_ = 1, to = 1000, length=180,
    orient=HORIZONTAL, variable=punkty)
pkt.set(200)
pkt.grid(row=6, column=1)

def linia_ciagla(xx0,yy0,xx,yy):
  c.create_line(xx0,yy0,xx,yy)

def punkty(xx,yy):
  c.create_rectangle(xx-1,yy+1,xx+1,yy-1,
                     fill="black")

p = StringVar()
p.set("linia ciagla")
b10 = Radiobutton(top, command=linia_ciagla,
    text="linia ciagla", variable=p,
    value="linia ciagla")
b20 = Radiobutton(top, command=punkty,
    text="punkty", variable=p, value="punkty")
b10.grid(row=7, column=1, sticky=W)
b20.grid(row=8, column=1, sticky=W)

c = Canvas(top, height=500, width=500,
    relief=SUNKEN, bg="white", bd=2)
```

```
def calc(event=None):
    global x_min, x_max, dx
    f = ftxt.get()
    x1 = float(xmin.get())
    x2 = float(xmax.get())
    dx=( x2 - x1 )/pkt.get()
    xscale=xlen/(x2-x1)
    x=x1
    xx0=ox+xscale*(x-x1)
    y1 = float(ymin.get())
    y2 = float(ymax.get())

#   osie ukladu wspolrzednych:
    c.create_line(ox,oy,ox+xlen,oy)
    c.create_line(ox,oy,ox,oy-ylen)

    print x1, x2, y1, y2
    yscale = ylen/(y2-y1)
    print xscale, yscale
    y = eval(f)
    yy0=oy-yscale*(y-y1)
    while x<=x2:
      x=x+dx
      y = eval(f)
      xx=ox+xscale*(x-x1)
      yy=oy-yscale*(y-y1)
      if p.get()=="linia ciagla":
        linia_ciagla(xx0,yy0,xx,yy)
      if p.get()=="punkty":
        punkty(xx,yy)
      xx0=xx
      yy0=yy

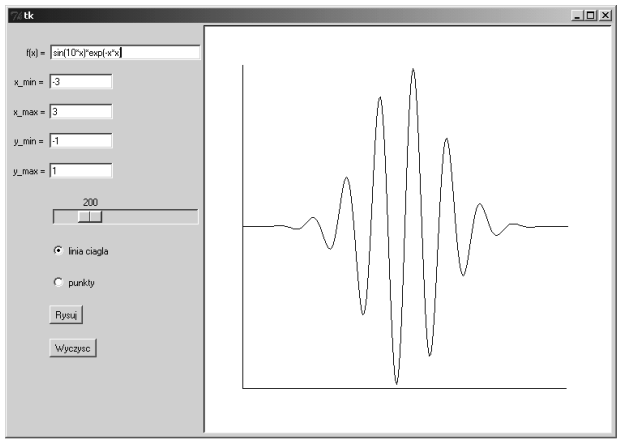
def wyczysc(event=None):
  c.create_rectangle(0,0,500,500,
      fill="white",width=0)

Button(top, text=' Rysuj ', relief='raised',
       command=calc).grid(row=9,
       column=1, sticky=W)
```

```
Button(top, text=' Wyczysc ', relief='raised',
       command=wyczysc).grid(row=10,
       column=1, sticky=W)
#s_label.pack()

c.grid(row=0, rowspan=16, column=2)

def quit(event=None): root.quit()
root.bind('<q>', quit)
root.mainloop()
```




Rys. 7.8 Prosty interfejs do rysowania wykresu funkcji

2. Nieco bardziej rozbudowany przykład

```
#!/usr/bin/env python
from Tkinter import *
import tkFont
from math import *
import string

# szerokosc i wysokosc obrazu, w pikselach
oknox = 500; oknoy = 500
# poczatek ukladu wspolrzednych na wykresie
# - we wspolrzednych ekranu
```

```
ox = 70; oy = 450
# dlugosc osi x i osi y na ekranie, w pikselach
xlen = 400; ylen = 400

# dlugosc kreski na osi wspolrzednych
lkr = 10

# odleglosc napisu od osi poziomej
yod = 16

# odleglosc napisu od osi pionowej
xod = 30

# kolor tla
kolortla = 'white'
# kolor siatki
kolorsiatki = 'grey'

root = Tk()
top = Frame(root); top.pack()

root.title('Wykres')

w = Frame(top)

pf = StringVar()
pf.set("funkcja")
Radiobutton(w, text="", variable=pf,
  value="plik").grid(row=1, column=0,
  sticky=W)
Radiobutton(w, text="", variable=pf,
  value="funkcja").grid(row=2,
  column=0, sticky=W)

# Dane mozna wczytac z pliku;
# zakladamy, w pierwszej kolumnie
# sa wartosci x, w drugiej y
Label(w, text=' Dane z pliku: ').grid(row=1,
  column=1, sticky=E)
plikdane = StringVar()
```

```
plik_dane = Entry(w, width=20,
  textvariable=plikdane)
plik_dane.grid(row=1, column=2)
plik_dane.insert('end', './plot.dane')

Label(w, text=' f(x) = ').grid(row=2,
  column=1, sticky=E)
ftxt = StringVar()
f_entry = Entry(w, width=20, textvariable=ftxt)
f_entry.grid(row=2, column=2)
f_entry.insert('end', 'x')

w.grid(row=1, column=0, columnspan=2, sticky=W)

wxy = Frame(top, borderwidth=2, relief="groove")

Label(wxy, text=' x_min = ').grid(row=1,
  column=0, sticky = E)
xmin = StringVar()
x_min = Entry(wxy, width=6, textvariable=xmin)
x_min.grid(row=1, column=1, sticky = W)
x_min.insert('end', '-10')
Label(wxy, text=' x_max = ').grid(row=2,
  column=0, sticky = E)
xmax = StringVar()
x_max = Entry(wxy, width=6, textvariable=xmax)
x_max.grid(row=2, column=1, sticky = W)
x_max.insert('end', '10')

Label(wxy, text=' y_min = ').grid(row=1,
  column=2, sticky = E)
ymin = StringVar()
y_min = Entry(wxy, width=6, textvariable=ymin)
y_min.grid(row=1, column=3, sticky = W)
y_min.insert('end', '-10')
Label(wxy, text=' y_max = ').grid(row=2,
  column=2, sticky = E)
ymax = StringVar()
y_max = Entry(wxy, width=6, textvariable=ymax)
y_max.grid(row=2, column=3, sticky = W)
```

```
y_max.insert('end', '10')

wxy.grid(row=3, rowspan=2, column=0, sticky=W)

wp = Frame(top, borderwidth=2, relief="groove")

Label(wp, text=' Punkty na wykresie ').grid(row=1,
      column=0, sticky=W)
punkty = IntVar()
pkt = Scale(wp, from_ = 1, to = 1000, length=100,
    orient=HORIZONTAL, variable=punkty)
pkt.set(200)
pkt.grid(row=1, column=1, columnspan=1, sticky = W)

wp.grid(row=5, rowspan=2, column=0, sticky=W)

def rysuj_linia(xx0,yy0,xx,yy):
  c.create_line(xx0,yy0,xx,yy, fill=kolor.get())

def rysuj_punkty(xx,yy):
  c.create_rectangle(xx,yy,xx,yy, fill=kolor.get())

def rysuj_3x3(xx,yy):
  c.create_rectangle(xx-1,yy+1,xx+1,yy-1,
    outline=kolor.get(), fill=kolor.get())

wwpk = Frame(top, borderwidth=2, relief="groove")
wp = Frame(wwpk, borderwidth=2, relief="groove")

p = StringVar()
p.set("linia ciagla")
Radiobutton(wp, text="linia ciagla", variable=p,
  value="linia ciagla").grid(row=1, column=0, sticky=W)
Radiobutton(wp, text="punkty", variable=p,
  value="punkty").grid(row=2, column=0, sticky=W)
Radiobutton(wp, text="3x3", variable=p,
  value="3x3").grid(row=3, column=0, sticky=W)

wp.grid(row=1, rowspan=3, column=0, sticky=W)
```

```
wkolor=Frame(wwpk, borderwidth=2, relief="groove")

kolor = StringVar()
kolor.set("black")
czarny = Radiobutton(wkolor, text="czarny",
  variable=kolor, value="black").grid(row=1,
  column=0, sticky=W)
niebieski = Radiobutton(wkolor,
  text="niebieski", variable=kolor,
  value="blue").grid(row=2, column=0, sticky=W)
czerwony = Radiobutton(wkolor, text="czerwony",
  variable=kolor, value="red").grid(row=3,
  column=0, sticky=W)

wkolor.grid(row=1, rowspan=3, column=1, sticky=W)
wwpk.grid(row=7, column=0, sticky=W)

wk = Frame(top, borderwidth=2, relief="groove")

Label(wk, text="Kreski na osiach: ").grid(row=1,
      column=0, sticky=E)
Label(wk, text="x ").grid(row=1,column=1,sticky=W)
kreskix = StringVar()
k_x = Entry(wk, width=2, textvariable=kreskix)
k_x.grid(row=1, column=2, sticky = W)
k_x.insert('end', '5')

Label(wk, text="  y ").grid(row=1,
  column=3, sticky=E)
kreskiy = StringVar()
k_y = Entry(wk, width=2, textvariable=kreskiy)
k_y.grid(row=1, column=4, sticky=W)
k_y.insert('end', '5')

wk.grid(row=8, column=0, columnspan=2, sticky=W)

c = Canvas(top, height=oknox, width=oknoy,
           relief=SUNKEN, bg="white", bd=2)
def calc(event=None):
    global x_min, x_max, dx
```

```
# wczytuje ciag znakow z pola tekstowego
# oznaczajacy postac funkcji:
    f = ftxt.get()
    x1 = float(xmin.get())
    x2 = float(xmax.get())
    dx = (x2 - x1)/pkt.get()
    xscale = xlen/(x2-x1)
    x = x1
    xx0 = ox+xscale*(x-x1)
    y1 = float(ymin.get())
    y2 = float(ymax.get())

    if pf.get()=="funkcja":
      yscale = ylen/(y2-y1)
      y = eval(f)
      yy0=oy-yscale*(y-y1)
      while x<=x2:
        x=x+dx
        # oblicza wartosc funkcji f(x)
        y = eval(f)
        xx=ox+xscale*(x-x1)
        yy=oy-yscale*(y-y1)
        if p.get()=="linia ciagla":
          rysuj_linia(xx0,yy0,xx,yy)
        if p.get()=="punkty":
          if yy <= oy and yy >= oy-yscale*(y2-y1):
            rysuj_punkty(xx,yy)
        if p.get()=="3x3":
          if yy <= oy and yy >= oy-yscale*(y2-y1):
            rysuj_3x3(xx,yy)
        xx0=xx
        yy0=yy
    if pf.get()=="plik":
      yscale = ylen/(y2-y1)
      pp = open(plik_dane.get())
      if p.get()=="linia ciagla":
        wiersz = pp.readline()
        wiersz_str = wiersz.split()
        x = float(wiersz_str[0])
        y = float(wiersz_str[1])
```

```
          xx0 = ox+xscale*(x-x1)
          yy0 = oy-yscale*(y-y1)
          wiersze = pp.readlines()
          for i in range(1,len(wiersze)+1):
            print i
            wiersz_str = wiersze[i-1].split()
            x = float(wiersz_str[0])
            y = float(wiersz_str[1])
            xx = ox + xscale*(x-x1)
            yy = oy - yscale*(y-y1)
            rysuj_linia(xx0,yy0,xx,yy)
            xx0 = xx
            yy0 = yy
        if p.get()=="punkty" or p.get()=="3x3":
          for wiersz in pp.readlines():
            wiersz_str = wiersz.split()
            x = float(wiersz_str[0])
            y = float(wiersz_str[1])
            xx = ox + xscale*(x-x1)
            yy = oy - yscale*(y-y1)
            if yy <= oy and yy >= oy-yscale*(y2-y1):
              if p.get()=="punkty":
                rysuj_punkty(xx,yy)
              else:
                rysuj_3x3(xx,yy)

# jesli wykres przekroczy gorna lub dolna os x,
# prostokat w kolorze tla usunie czesc
# wykresu nad gorna osia x
    c.create_rectangle(ox,oy-ylen,ox+xlen,0,
      fill=kolortla, width=0)
# lub dolna
    c.create_rectangle(ox,oknoy+10,ox+xlen,oy+1,
      fill=kolortla, width=0)

#   prostokatna ramka wykresu
    c.create_rectangle(ox,oy,ox+xlen,oy-ylen)

    h16 = tkFont.Font(family="Helvetica", size=16)
#   kreski na osiach
```

```
    mkx = int(kreskix.get())
    for i in range(0, mkx+1):
      xk = ox + i*xlen/mkx
# krawedz dolna
      c.create_line(xk, oy, xk, oy-lkr)
    for i in range(0, mkx+1):
      xk = ox + i*xlen/mkx
      c.create_text(xk, oy+yod,
        text=str(x1+i*(x2-x1)/mkx), font=h16)
# krawedz gorna
      c.create_line(xk, oy-ylen, xk, oy-ylen+lkr)

    mky = int(kreskiy.get())
    for i in range(0, mky+1):
      yk = oy - i*ylen/mky
# krawedz lewa
      c.create_line(ox, yk, ox+lkr, yk)
# liczby przy osi y
      c.create_text(ox-xod, yk,
        text=str(y1+i*(y2-y1)/mky), font=h16)
# krawedz prawa
      c.create_line(ox+xlen-lkr, yk, ox+xlen, yk)

def siatka(event=None):
  mkx = int(kreskix.get())
  for i in range(1,mkx):
    xk = ox + i*xlen/mkx
    c.create_line(xk, oy, xk, oy-ylen,
      fill=kolorsiatki)
  mky = int(kreskiy.get())
  for i in range(1,mky):
    yk = oy - i*ylen/mky
    c.create_line(ox, yk, ox+xlen, yk,
      fill=kolorsiatki)

# siatka (grid)
Checkbutton(top, text='Rysuj siatke',
  command=siatka).grid(row=9, column=0, sticky=W)

# zapis do pliku
```

```
def zapisz_plik_eps():
  c.postscript(file=nazwapliku.get())

wsp = Frame(top, borderwidth=2, relief="groove")

nazwapliku = StringVar()
Entry(wsp, width=20,
  textvariable=nazwapliku).grid(row=2, column=1,
  sticky=W)

Checkbutton(wsp, text='Zapisz obraz do pliku',
  command=zapisz_plik_eps).grid(row=1, column=1,
  sticky=W)

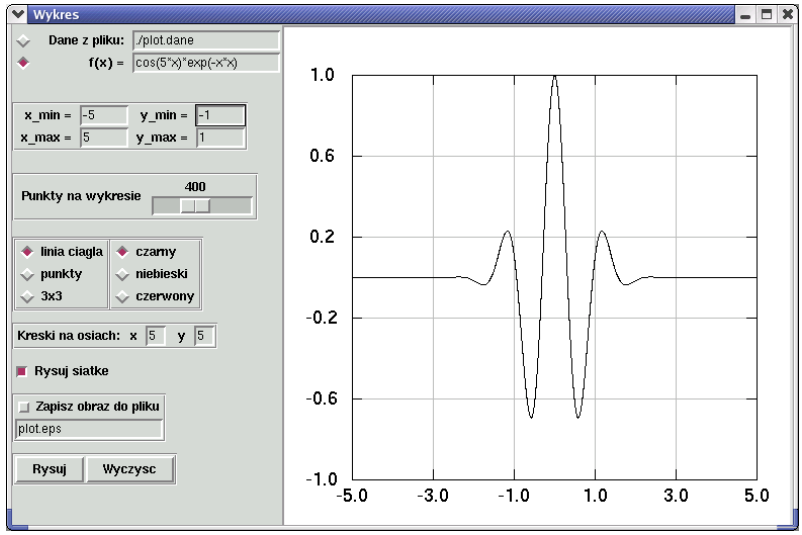
wsp.grid(row=10, column=0, sticky=W)

def wyczysc(event=None):
  c.create_rectangle(0,0,oknox,oknoy,
  fill="white",width=0)

wb = Frame(top, borderwidth=2, relief="groove")
Button(wb, text=' Rysuj ', relief='raised',
       command=calc).grid(row=1, column=0,
       sticky=W)
Button(wb, text=' Wyczysc ', relief='raised',
       command=wyczysc).grid(row=1, column=1,
       sticky=W)
wb.grid(row=11, column=0, sticky=W)

c.grid(row=1, rowspan=16, column=2,
       columnspan=10)

def quit(event=None): root.quit()
root.bind('<q>', quit)
root.mainloop()
```




Rys. 7.9 Nieco bardziej złożony interfejs do rysowania funkcji

## 7.14. Ilustracja rozwiązania równania różniczkowego zwyczajnego

Opierając się na skrypcie przedstawionym w poprzednim podrozdziale, łatwo jest tworzyć programy do obliczeń numerycznych z jednoczesną graficzną prezentacją wyników.

```
from Tkinter import *
from math import *
from string import *

ox = 50
oy = 450
xlen = 400
ylen = 400

root = Tk()
top = Frame(root); top.pack()
Label(top, text=' f(x) = ').grid(row=1,
  column=0, sticky = E)
```

```
ftxt = StringVar()
f_entry = Entry(top, width=20, textvariable=ftxt)
f_entry.grid(row=1, column=1, columnspan=1)
f_entry.insert('end', '-y')

Label(top, text= 't_pocz = ').grid(row=2,
  column=0, sticky=E)
tpocz = StringVar()
t_pocz = Entry(top, width=8, textvariable=tpocz)
t_pocz.grid(row=2, column=1, sticky = W)

Label(top, text= 't_konc = ').grid(row=3,
  column=0, sticky=E)
tkonc = StringVar()
t_konc = Entry(top, width=8, textvariable=tkonc)
t_konc.grid(row=3, column=1, sticky = W)

#Label(top, text= 'dt = ').grid(row=4,
#  column=0, sticky=E)
#dt0 = StringVar()
#dt_0 = Entry(top, width=8, textvariable=dt0)
#dt_0.grid(row=3, column=1, sticky = W)

Label(top, text= 'y_pocz = ').grid(row=5,
  column=0, sticky=E)
y0 = StringVar()
y_0 = Entry(top, width=8, textvariable=y0)
y_0.grid(row=5, column=1, sticky = W)

Label(top, text=' y_min = ').grid(row=6,
  column=0, sticky = E)
ymin = StringVar()
y_min = Entry(top, width=8, textvariable=ymin)
y_min.grid(row=6, column=1, sticky = W)
y_min.insert('end', '-10')

Label(top, text=' y_max = ').grid(row=7,
  column=0, sticky = E)
ymax = StringVar()
y_max = Entry(top, width=7, textvariable=ymax)
```

```
y_max.grid(row=7, column=1, sticky = W)
y_max.insert('end', '10')

Label(top, text=' Punkty na wykresie ').grid(row=8,
      column=0, sticky=E)
punkty = IntVar()
pkt = Scale(top, from_ = 1, to = 1000, length=100,
    orient=HORIZONTAL, variable=punkty)
pkt.set(200)
pkt.grid(row=8, column=1, columnspan=1, sticky = W)

def linia_ciagla(xx0,yy0,xx,yy):
    c.create_line(xx0,yy0,xx,yy, fill=kolor.get())

def punkty(xx,yy):
  c.create_rectangle(xx-1,yy+1,xx+1,yy-1,
  fill="black")

p = StringVar()
p.set("linia ciagla")
linia = Radiobutton(top, command=linia_ciagla,
    text="linia ciagla", variable=p,
    value="linia ciagla")
punkty = Radiobutton(top, command=punkty,
    text="punkty", variable=p, value="punkty")
linia.grid(row=9, column=0, sticky=W)
punkty.grid(row=10, column=0, sticky=W)

kolor = StringVar()
kolor.set("black")
czarny = Radiobutton(top, text="czarna",
  variable=kolor, value="black")
niebieski = Radiobutton(top, text="niebieska",
  variable=kolor, value="blue")
czerwony = Radiobutton(top, text="czerwona",
  variable=kolor, value="red")
czarny.grid(row=9, column=1, sticky=W)
niebieski.grid(row=10, column=1, sticky=W)
czerwony.grid(row=11, column=1, sticky=W)
```

```
Label(top, text="Kreski na osiach").grid(row=12,
      column=0, sticky=E)
kreski = StringVar()
kreski_t = Entry(top, width=2, textvariable=kreski)
kreski_t.grid(row=12, column=1, sticky = W)
kreski_t.insert('end', '5')

c = Canvas(top, height=500, width=500,
           relief=SUNKEN, bg="white", bd=2)

def calc(event=None):
    global x_min, x_max, dx
# wczytuje ciag znakow z pola tekstowego
# oznaczajacy postac funkcji:
    f = ftxt.get()
#    x1 = float(xmin.get())
#    x2 = float(xmax.get())
    x1 = float(tpocz.get())
    x2 = float(tkonc.get())
    dx=(x2 - x1)/pkt.get()
    xscale = xlen/(x2-x1)
    x = x1
    xx0 = ox + xscale*(x-x1)
    y1 = float(ymin.get())
    y2 = float(ymax.get())

#   osie ukladu wspolrzednych
    c.create_line(ox,oy,ox+xlen,oy)
    c.create_line(ox,oy,ox,oy-ylen)

#   kreski na osiach
    mk = int(kreski_t.get())
    for i in range(1,mk+1):
      xk = ox + i*xlen/mk
      c.create_line(xk, oy, xk, oy-10)
      yk = oy - i*ylen/mk
      c.create_line(ox, yk, ox+10, yk)

    yscale = ylen/(y2-y1)
#   y = eval(f)
```

```
    y = float(y0.get())
    yy0=oy - yscale*(y-y1)

    while x<=x2:
      x=x + dx
#      y = eval(f)
      xx=ox+xscale*(x-x1)
#      yy=oy-yscale*(y-y1)
      y = y + eval(f) * dx
      yy = oy - yscale*(y-y1)

      if p.get()=="linia ciagla":
        linia_ciagla(xx0,yy0,xx,yy)

      if p.get()=="punkty":
        punkty(xx,yy)
      xx0=xx
      yy0=yy

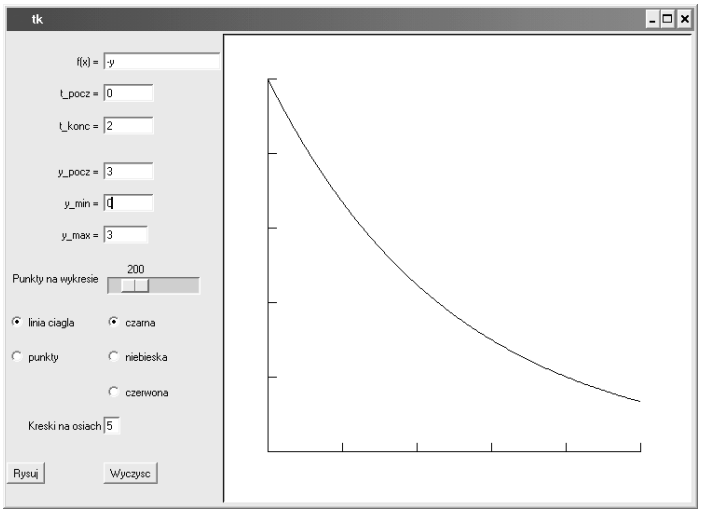
def wyczysc(event=None):
  c.create_rectangle(0,0,500,500,
  fill="white",width=0)

Button(top, text=' Rysuj ', relief='raised',
       command=calc).grid(row=14,
       column=0, sticky=W)

Button(top, text=' Wyczysc ', relief='raised',
       command=wyczysc).grid(row=14,
       column=1, sticky=W)
#s_label.pack()

c.grid(row=0, rowspan=16, column=2, columnspan=20)

def quit(event=None): root.quit()
root.bind('<q>', quit)
root.mainloop()
```




Rys. 7.10 Rozwiązanie równania różniczkowego zwyczajnego

## 7.15. Kody pocztowe i położenie geograficzne

Na zakończenie tego rozdziału przedstawiam program rysujący mapę Polski z wybranymi miejscowościami oraz interfejs graficzny z mapą Polski, na której można zaznaczyć miejscowości, których nazwy lub kody pocztowe pasują do określonych wyrażeń regularnych.

```
57-100 Strzelin Do 50 47 N 17 4 E 50.7833 17.0667 174
58-100 Swidnica Do 50 51 N 16 30 E 50.8500 16.5000 209
58-300 Walbrzych Do 50 46 N 16 17 E 50.7667 16.2833 509
```

Plik danych przekształcamy do takiej postaci, w której poszczególne pola są oddzielone przecinkiem. Dzięki temu łatwiejsze staje się przetwarzanie nazw miast, które mogą się składać z kilku członów rozdzielonych spacjami.

```
import re
plik = open('miasta_pow.d')
plikwyj = open('miasta_pow.csv', 'w')
for wiersz in plik.readlines():
  wiersz1 = re.sub('\t', ' ', wiersz)
```

```
  w1 = wiersz1.split(' ')
  liczba_pol = len(w1)
  n = w1[0] + ','
  for i in range(1, liczba_pol-10):
    if i<liczba_pol-11:
      n = n + w1[i] + ' '
    else:
      n = n + w1[i]
  for i in range(liczba_pol-10, liczba_pol):
    n = n + ',' + w1[i]
  plikwyj.write(str(n))
plik.close()
plikwyj.close()
```

Nowa postać pliku danych:

```
57-100,Strzelin,Do,50,47,N,17,4,E,50.7833,17.0667,174
58-100,Swidnica,Do,50,51,N,16,30,E,50.8500,16.5000,209
58-300,Walbrzych,Do,50,46,N,16,17,E,50.7667,16.2833,509
...
06-400,Ciechanow,Mz,52,53,N,20,37,E,52.8833,20.6167,119
09-500,Gostynin,Mz,52,26,N,19,29,E,52.4333,19.4833,73
...
70-000,Szczecin,Za,53,25,N,14,35,E,53.4167,14.5833,1
71-000,Szczecin,Za,53,25,N,14,35,E,53.4167,14.5833,1
```

```
# -*- coding: cp1250 -*-

from Tkinter import *
#from math import *
import re
import tkFont

dane = ''
ox = 50
oy = 450
xlen = 500
ylen = 400

# dlugosc geograficzna
x_za = 14.0
y_pd = 49.0
```

```
x_ws = 24.5
y_pn = 55.0

x_za_mapa = ox
y_pd_mapa = oy
x_ws_mapa = ox + xlen
y_pn_mapa = oy - ylen

x_scale = xlen/(x_ws - x_za)
y_scale = ylen/(y_pn - y_pd)

print x_scale, y_scale

root = Tk()
root.title("Mapa Polski")
root.resizable()
ramka = Frame(root)

helv18 = tkFont.Font (family="Helvetica",size=18, weight="bold")
#tytul = Label(root, font=helv18,text="Mapa Polski", height=2)
c = Canvas(root, height=500, width=600,
           relief=SUNKEN, bg="white", bd=2)

plik = open('granica-PL.d')
w = plik.readline()
s = re.split(' ', w)
s[3] = re.sub('^0', '', s[3])
s[5] = re.sub('\n$', '', s[5])
y1 = oy - y_scale*(float(s[0])
   + float(s[1])/60 + float(s[2])/(60*60) - y_pd)
x1 = ox + x_scale*(float(s[3])
   + float(s[4])/60 + float(s[5])/(60*60) - x_za)
for wiersz in plik.readlines():
  s = re.split(' ', wiersz)
  s[3] = re.sub('^0', '', s[3])
  y2 = oy - y_scale*(float(s[0])
     + float(s[1])/60 + float(s[2])/(60*60) - y_pd)
  x2 = ox + x_scale*(float(s[3])
     + float(s[4])/60 + float(s[5])/(60*60) - x_za)
  c.create_line(x1, y1, x2, y2)
  x1 = x2
  y1 = y2
plik.close()
```

```
m = StringVar()
miasto = StringVar()
l1 = Label(root, text="Nazwa miasta", anchor=NE)
t = Entry(root, width=20, textvariable=m)

woj = StringVar()
l2 = Label(root, text=" Województwo", anchor=NE)
#w = Entry(root, width=30, textvariable=woj)
#w = Listbox(root, width=30, textvariable=woj)
wartosci = ["Dolnoslaskie", "Kujawsko-Pomorskie",
            "Lubelskie", "Lubuskie", "Lodzkie",
            "Malopolskie", "Mazowieckie", "Opolskie",
            "Podkarpackie", "Podlaskie", "Pomorskie",
            "Slaskie", "Swietokrzyskie",
            "Warminsko-Mazurskie", "Wielkopolskie",
            "Zachodniopomorskie"]
woj_nazwy = {
             "Dolnoslaskie" : "Do",
             "Kujawsko-Pomorskie" : "Ku",
             "Lubelskie" : "Le",
             "Lubuskie" : "Lu",
             "Lodzkie" : "Lo",
             "Malopolskie" : "Ma",
             "Mazowieckie" : "Mz",
             "Opolskie" : "Op",
             "Podkarpackie" : "Po",
             "Podlaskie" : "Pl",
             "Pomorskie" : "Pm",
             "Slaskie" : "Sl",
             "Swietokrzyskie" : "Sw",
             "Warminsko-Mazurskie" : "Wm",
             "Wielkopolskie" : "Wi",
             "Zachodniopomorskie" : "Za",
            }
wojlista = Listbox(root, height=1)
for i in wartosci:
    wojlista.insert(END, i)
przewin = Scrollbar(root, command = wojlista.yview)
wojlista.configure(yscrollcommand = przewin.set)

kod = StringVar()
l3 = Label(root, text=" Kod pocztowy", anchor=NE)
```

```
k = Entry(root, width=6, textvariable=kod)

def miasto_szukaj():
  global dane
  miasto = m.get()
  plik = open('miasta_pow.csv')
  for wiersz in plik.readlines():
    w1 = wiersz.split(',')
    if w1[1].find(miasto) >= 0:
      dane = dane + w1[0] + ' ' \
      + w1[1] + ' ' + w1[2] + "\n"
      x = ox + x_scale*(float(w1[10]) - x_za)
      y = oy - y_scale*(float(w1[9]) - y_pd)
      c.create_rectangle(x-1,y-1,x+1,y+1,
        fill="blue", width=1, tags="miasto")
  plik.close()
  print dane

def miasta_usun():
  c.delete("miasto")

def woj_zaznacz():
  global dane
#  w = woj.get()
  nr = wojlista.curselection()
  x = wojlista.get(nr)
  w = woj_nazwy[x]
  plik = open('miasta_pow.csv')
  for wiersz in plik.readlines():
    w1 = wiersz.split(',')
    if w1[2].find(w) >= 0:
      dane = dane + w1[0] + ' ' \
      + w1[1] + ' ' + w1[2] + "\n"
      x = ox + x_scale*(float(w1[10]) - x_za)
      y = oy - y_scale*(float(w1[9]) - y_pd)
      c.create_rectangle(x-1,y-1,x+1,y+1,
        fill="blue", width=1, tags="miasto")
  plik.close()
  print dane

def kod_szukaj():
  global dane
  k = kod.get()
```

```
  plik = open('miasta_pow.csv')
  for wiersz in plik.readlines():
    w1 = wiersz.split(',')
    if w1[0].find(k) >= 0:
      dane = dane + w1[0] + ' ' \
      + w1[1] + ' ' + w1[2] + "\n"
      x = ox + x_scale*(float(w1[10]) - x_za)
      y = oy - y_scale*(float(w1[9]) - y_pd)
      c.create_rectangle(x-1,y-1,x+1,y+1,
        fill="blue", width=1, tags="miasto")
  plik.close()
  print dane

def zapisz_plik_eps():
  c.postscript(file="PLmapa.eps")

pasekmenu = Menu(root)

menuplik = Menu(pasekmenu, tearoff=0)
menuplik.add_command(label="Wyczysc", command=miasta_usun)
menuplik.add_command(label="Zapisz jako eps",
                     command=zapisz_plik_eps)
menuplik.add_separator()
menuplik.add_command(label="Zamknij", command=root.quit)
pasekmenu.add_cascade(label="Plik", menu=menuplik)

menuopcje = Menu(pasekmenu, tearoff=0)
menuopcje.add_command(label="Miasto", command=miasto_szukaj)
menuopcje.add_command(label="Województwo", command=woj_zaznacz)
menuopcje.add_command(label="Kod", command=kod_szukaj)
pasekmenu.add_cascade(label="Opcje", menu=menuopcje)

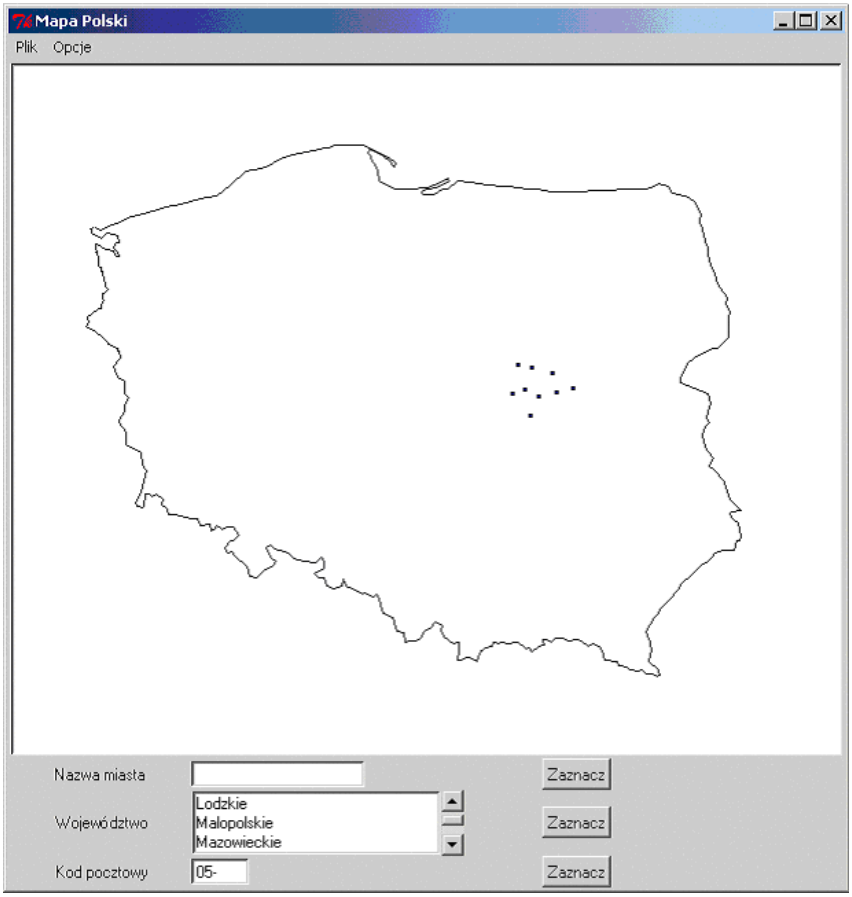
root.config(menu=pasekmenu)

b1 = Button(root, text='Zaznacz', command=miasto_szukaj)
b3 = Button(root, text='Zaznacz', command=woj_zaznacz)
b5 = Button(root, text='Zaznacz', command=kod_szukaj)

komunikat = Text(root)
# tytul.grid(row=1, column=0, columnspan=3)
c.grid(row=1, column=0, columnspan=6)
l1.grid(row=2, column=0)
t.grid(row=2, column=1, sticky=W)
```

```
b1.grid(row=2, column=3, sticky=W)
l2.grid(row=3, column=0)
wojlista.grid(row=3, column=1, sticky=N+S+E+W)
przewin.grid(row = 3, column = 2, sticky = W+N+S)
b3.grid(row=3, column=3, sticky=W)
l3.grid(row=4, column=0)
k.grid(row=4, column=1, sticky=W)
b5.grid(row=4, column=3, sticky=W)
root.mainloop()
```

Rys. 7.11 Wynik skryptu `PLmapa2.py`




Rys. 7.12 Okno programu rysującego mapę Polski z wybranymi miejscowościami

# 8. Zadania

1. Napisz polecenie drukujące te pliki i katalogi z bieżącego katalogu, które mogą być uruchamiane przez wszystkich użytkowników.

   ```
   ls -l | grep '^.........x'
   ```

2. Dany jest plik tekstowy. Podaj polecenie, które wydrukuje wszystkie wiersze zawierające wielką literę A co najmniej 3 razy.

   ```
   grep 'A.*A.*A' plik
   ```

   To samo polecenie można wykonać na wielu plikach, na przykład na wszystkich plikach bieżącego katalogu:

   ```
   grep 'A.*A.*A' *
   ```

   lub

   ```
   for i in *
   do
     grep 'A.*A.*A' $i
   done
   ```

3. Dany jest plik tekstowy zawierający po jednym słowie w wierszu. Napisz polecenie, które wydrukuje liczbę słów kończących się literą `o`.

   ```
   grep -c 'o$' plik
   ```

4. Wydrukuj wiersze rozpoczynające się literą `b` lub `B`.

   ```
   grep '^[Bb]' plik
   ```

5. Zamień w pliku tekstowym wszystkie wielkie litery na małe.

```
sed 'y/ABCDEFGHIJKLMNOPQRSTUVWXYZ/\
       abcdefghijklmnopqrstuvwxyz/' plik
```

   Inny sposób:

```
cat plik | tr '[a-z]' '[A-Z]'
```

6. Dany jest plik z nazwami geograficznymi, po jednej nazwie w wierszu. Niektóre nazwy składają się z dwóch lub większej liczby członów, oddzielonych znakiem `-`, spacją lub wieloma spacjami (np. Grodzisk Wielkopolski, Saint-Etienne). Wydrukuj wiersze składające się z dwóch lub więcej członów. Uwaga: zrób to tak, by uzyskać poprawny wynik również wtedy, kiedy na początku lub na końcu wiersza występują spacje.

```
egrep '[A-Z][a-z]*([ ][ ]*|[-])[A-Za-z]*' plik
```

7. Drukuj wiersze pliku tekstowego zawierające co najmniej 5 liter `x` (niekoniecznie następujących po sobie).

```
grep 'x.*x.*x.*x.*x' plik
```

8. Plik słownika zawiera po jednym słowie w każdym wierszu. Jeśli słowo jest nazwą własną, rozpoczyna się wielką literą. Drukuj słowa zawierające

   (a) dwie litery a (uwzględnić możliwość występowania wielkiej litery)

```
egrep '[Aa].*a' plik
```

   (b) dwie samogłoski

```
egrep '[AEIOUYaeiouy].*[aeiouy]' plik
```

9. Napisz polecenia edytora (potrzebne są dwa oddzielne polecenia), które przekształcą wiersz

   *W domu były kot i pies*

   na wiersz

   *W domu były pies i ptak.*

```
sed 's/pies/ptak'
sed 's/kot/pies'
```

Zadania 10-25 dotyczą następującego pliku tekstowego:

```
www.interia.pl http://www.interia.pl PORTAL INTERNETOWY
www.ONET.pl http://www.ONET.pl
http://www.interia.pl, Portal 1998 2004
www.yahoo.com, www.GOOGLE.pl, www.GOOGLE.com
http://www.google.com Wyszukiwarka http://www.google.pl
```

10. Napisz polecenie usuwające wiersze zawierające adresy rozpoczynające się ciągiem znaków `http`.

```
sed '/^http/d'
```

11. Napisz polecenie drukujące tylko wiersze kończące się ciągiem znaków `com`.

```
sed -n '/com$/p' plik
```

12. Napisz polecenie drukujące tylko wiersze zawierające liczby.

```
sed -n '/[0-9]/p' plik
```

13. Napisz polecenie usuwające wszystkie słowa (ale nie adresy internetowe) składające się wyłącznie z wielkich liter.

```
sed 's/ [A-Z][A-Z]*[ $]//g' plik
```

14. Napisz polecenie edytora zastępujące wszystkie wielkie litery małymi wszędzie oprócz wierszy zawierających ciąg `http`.

```
sed '/http/!y/QWERTYUIOPASDFGHJKLZXCVBNM/\
              qwertyuiopasdfghjklzxcvbnm/' plik
```

15. Wykonaj zadanie 1 tak, by polecenia edytora były wczytywane z pliku. Podaj polecenie, które należy wykonać i treść pliku, w którym są zapisane polecenia edytora.

```
sed -f skrypt.sed
```

Zawartość pliku `skrypt.sed`:

```
s/pies/ptak/
s/kot/pies/
```

16. Napisz polecenie usuwające ciąg znaków `http://`

```
sed 's/http:\/\///g' plik
```

17. Napisz polecenie drukujące tylko wiersze od 2 do 4.

```
sed -n '2,4p' plik
```

18. Napisz polecenie drukujące tylko wiersze zawierające wielkie litery.

```
sed -n '/[A-Z]/p' plik
```

19. Napisz polecenie usuwające wszystkie liczby.

```
sed 's/[0-9]//g' plik
```

20. Napisz polecenie edytora zastępujące wszystkie spacje dwoma przecinkami wszędzie, oprócz wierszy zawierających pojedyncze przecinki.

```
sed '/,/!s/ /,,/g' plik
```

21. Napisz polecenie usuwające wiersze kończące się ciągiem znaków `com`.

```
sed '/com$/d' plik
```

22. Napisz polecenie drukujące tylko wiersze zawierające adresy polskiej części Internetu.

```
sed -n '/\.pl /p' plik
```

23. Napisz polecenie drukujące wiersze od 1 do pierwszego, w którym pojawiają się cyfry.

```
sed -n '1,/[0-9]/p' plik
```

24. Napisz polecenie usuwające wszystkie przecinki.

```
sed 's/,//g' plik
```

25. Napisz polecenie edytora zastępujące wielkie litery małymi w wierszach zawierających co najmniej dwie kolejne wielkie litery.

```
sed '/[A-Z][A-Z]/y/ABCDEFGHIJKLMNOPQRSTWVXYZ/\
                   abcdefghijklmnopqrstwvxyz/' plik
```

26. Napisz skrypt zamieniający angielskie nazwy dni i miesięcy na polskie w wyniku polecenia `date` systemu Unix.

```
date | sed 's/Mon/Pon/; s/Tue/Wto/; s/Wed/Sro/;
s/Thu/Czw/; s/Fri/Pia/; s/Sat/Sob/; s/Sun/Nie/
s/Jan/Sty/; s/Feb/Lut/; #s/Mar/Mar/
s/Apr/Kwi/; s/May/Maj/; s/Jun/Cze/
s/Jul/Lip/; s/Aug/Sie/; s/Sep/Wrz/
s/Oct/Paz/; s/Nov/Lis/; s/Dec/Gru/'
```

27. Napisz polecenie, które usunie wszystkie spacje na początku i na końcu wiersza.

```
sed 's/^[ ]*//; s/[ ]*$//' plik
```

28. Usuń cały tekst, oprócz wierszy w zakresie od wyrażenia `w1` do wyrażenia `w2`. Wszyskie spacje zastąp przecinkami.

```
sed -e '/w1/,/w2/!d' -e 's/ /,/g' plik
```

Jeżeli dowolny ciąg spacji ma być zastąpiony jednym przecinkiem:

```
sed -e '/w1/,/w2/!d' -e 's/[ ][ ]*/,/g' plik
```

29. Dany jest plik tekstowy. Zamień wszystkie ciągi spacji znakami nowego wiersza. Poszczególne słowa mają pojawiać się w osobnych wierszach.

```
sed 's/[ ][ ]/\
>/g' plik
```

Znak > pojawiający się na początku drugiego wiersza jest znakiem kontynuacji polecenia, drukowanym przez powłokę. To samo polecenie edytora sed zapisane w skrypcie miałoby następującą postać:

```
s/[ ][ ]/\
/g
```

30. Usuń z pliku zawierającego kod źródłowy w języku C wszystkie wiersze zawierające komentarze.

(a) Zrób to najpierw w przypadku, kiedy istnieją tylko komentarze rozpoczynające się znakami //. Dla uproszczenia przyjmijmy, że w pliku kod i komentarze zawsze pojawiają się w osobnych wierszach

```
sed 's/^.*\/\/.*$//' plik.c
```

(b) Zrób to w przypadku, kiedy istnieją także komentarze zaznaczone znakami /* */, obejmujące kilka wierszy

```
sed '/\/\*/,/\*\//d' plik.c
```

(c) Zrób to w przypadku ogólnym: kod i komentarz mogą pojawiać się w tym samym wierszu

```
sed 's/\/\*/\
\/\*\
/; s/\*\//\
\*\/\
/' plik.c
```

Testuj swoje rozwiązania na odpowiednio przygotowanym pliku tekstowym.

31. Zastąp kropkę spacją.

```
sed 's/\./ /g'
```

32. Wczytaj plik tekstowy. Usuń spacje na początku i na końcu każdego wiersza. Jeżeli wiersz zawiera cyfry, dopisz go do pliku `A`. Pozostałe wiersze dopisz do pliku `B`.

```
sed 's/^[ ][ ]*//; s/[ ][ ]*$//' plik |\
sed -n '/[0-9]/w A
        /[0-9]/!w B'
```

33. Napisz skrypt wykonujący proste szyfrowanie tekstu. Szyfr powinien być cykliczny z okresem 2. Pierwsze uruchomienie skryptu szyfruje tekst, a drugie przywraca pierwotną postać tekstu. Załóżmy, że w tekście nie ma znaków diakrytycznych. Szyfrowaniu mają podlegać tylko litery i cyfry. Inne znaki, np. znaki przestankowe, nie są szyfrowane.

    Oto przykładowe rozwiązanie:

```
y/ABCDEFGHIJKLM/NOPQRSTUWVXYZ/
y/NOPQRSTUWVXYZ/ABCDEFGHIJKLM/
y/abcdefghijklm/nopqrstuwvxyz/
y/nopqrstuwvxyz/abcdefghijklm/
y/0123456789/5678901234/
```

    Spróbujmy zaszyfrować następujący plik:

```
Ala ma 2 kotki.
Ola ma 3 pieski.
```

```
sed -f ./skrypt.sed plik
Nyn zn 7 xbgxw.
Byn zn 8 cwrfxw.
```

34. Dodaj do nazwy każdego pliku rozmiar

    (a) w bajtach

```
for i in *
do
  rozmiar=`ls -l $i | awk '{ print $5 }'`
  mv $i ${i}$rozmiar
done
```

    (b) w kilobajtach

```
for i in *
do
  rozmiar=`ls -l $i | awk '{ print int(($5-1)/1000)+1 }'`
  mv $i ${i}$rozmiar
done
```

35. Wykonaj poprzednie zadanie, przeznaczając na rozmiar pliku dokładnie 9 znaków. W miejscu, w którym brak cyfry, drukuj znak podkreślenia _.

```
n=9
for i in *
do
  rozmiar=`ls -l $i |\
  awk '{ print int(($5-1)/1000)+1 }'`
  ncyfr=`awk 'BEGIN{ print length("'"$rozmiar"'") }'`
  k=0
  nowa_nazwa=$i
  while [ $k -lt $(($n-$ncyfr)) ]
  do
    k=$(($k+1))
    nowa_nazwa=${nowa_nazwa}_
  done
  nowa_nazwa=${nowa_nazwa}$rozmiar
  mv $i $nowa_nazwa
done
```

36. Zmień nazwy wszystkich plików w bieżącym katalogu tak, by nowa nazwa pliku składała się z pierwszych 6 znaków starej nazwy. Należy przyjąć, że nowe nazwy są jednoznaczne, tj. nie trzeba obawiać się powtórzeń wśród nowych nazw.

```
for i in *
do
  nowa_nazwa=`ls $i | awk 'BEGIN{ FS=""}\
    { print $1$2$3$4$5$6 }'`
  mv $i $nowa_nazwa
done
```

37. Uwzględnij w poprzednim zadaniu przypadek jednakowych nazw 6-znakowych. Jeżeli skrócona nazwa jest identyczna z inną, dodaj do obu datę utworzenia pliku w formacie `dd-mm-rrrr`.

```
ls -l $i | awk '
{ nazwa6[NR]=substr($9, 1, 6)
  nazwa[NR]=$9
  if ($8 ~ /[012][0-9]:[0-5][0-9]/)
    rok=strftime("%Y")
  else
    rok=$8
  data[NR]=$7"-"$6"-"rok
}
END{
  for (i=1; i<=NR; i++){
    print i
    for (k=1; k<=NR; k++){
      if (i!=k && nazwa6[i]==nazwa6[k]){
        system("mv " nazwa[i]" "nazwa6[i]"_"data[i]"")
        break; break
      }
    }
  }
}'
```

38. Do nazw wszystkich plików dodaj datę jego ostatniej modyfikacji w formacie `dd-mm-rrrr` (dla pliku z lat poprzednich) lub `dd-mm-gg:mm` (jeśli data dotyczy roku bieżącego). Jeżeli data była już wcześniej częścią nazwy, sprawdź, czy data ostatniej modyfikacji jest zgodna z datą w nazwie. Jeżeli są różne, umieść datę modyfikacji w miejsce daty poprzedniej.

```
ls -l $i | awk '{
nazwa=$9
data=$7"-"$6"-"$8
dlugnazwy=length($9)
if ((nazwa ~ /[0-9][0-9]-[ADFJMNOS][aceopu][bcglnprtvy]\
-[12][09][089][0-9]/)\
|| (nazwa ~ /[0-9][0-9]-[ADFJMNOS][aceopu][bcglnprtvy]\
-[012][0-9]:[0-5][0-9]/))
{
    datawnazwie=substr($9,dlugnazwy-10)
    if (data != datawnazwie)
    {
      nowanazwa=substr($9, 1, dlugnazwy-10)""data
      print nazwa, nowanazwa
    }
}
else
{ nowanazwa=$9""data
  print $9, nowanazwa }
}'
```

39. Plik tekstowy ma następującą postać:

    a1 b1 c1 d1

    a2 b2 c2 d2

    ...

    Drukuj wiersze, w których wartość liczbowa w kolumnie 3 jest większa niż w kolumnie 2.

    ```
    $3 > $2
    ```

40. Dany jest plik tekstowy. Zapisz wiersze o numerach od 200 do 300 w nowym pliku.

    ```
    awk 'NR == 200, NR == 300 { print }' plik > nowyplik
    ```

    lub

    ```
    sed -n '200,300p' plik > nowyplik
    ```

lub

```
head -300 plik | tail -100 > nowyplik
```

41. Dane w pliku tekstowym są ułożone w kolumnach. Drukuj numer wiersza, zawartość pierwszego pola i sumę pól, od drugiego do ostatniego. Liczba pól w wierszach jest zmienna.

    Skrypt awk:

```
{ s=0
  for (i=2, i<=NF, i++)
  { s=s+$i }
  print NR, $1, s
}
```

42. Napisz skrypt, który drukuje zawartość pliku od początku aż do wiersza, w którym po raz $n$-ty pojawia się poszukiwany ciąg znaków. Nazwa pliku, poszukiwany ciąg oraz liczba $n$ mają być przekazywane do skryptu jako parametry.

```
#!/usr/bin/bash
plik=$1
lancuch=$2
liczba=$3
IFS=$'\n'
j=0
for i in `cat $plik`
do
  s=`echo "$i" | grep "$lancuch"`
  if [ -n "$s" ]
  then
    j=$(($j+1))
    if [ $j -eq $3 ]
    then
      echo "$i"
      exit 0
    fi
  fi
  echo $i
done
```

43. Oblicz liczbę spacji w pliku tekstowym

```
BEGIN { FS = "" }
{ for (i=1; i<=NF; i++)
  {
    if ($i == " ")
    { suma++ }
  }
}
END { print suma }
```

44. Oto wynik polecenia `netstat -a`:

```
$ netstat -a

Aktywne polaczenia

 Protokol Adres lokalny       Obcy adres  Stan
 TCP     komp1:epmap          komp1:0     NASLUCHIWANIE
 TCP     komp1:microsoft-ds   komp1:0     NASLUCHIWANIE
 TCP     komp1:1028           komp1:0     NASLUCHIWANIE
 TCP     komp1:44334          komp1:0     NASLUCHIWANIE
 TCP     komp1:netbios-ssn    komp1:0     NASLUCHIWANIE
 UDP     komp1:microsoft-ds   *:*
 UDP     komp1:44334          *:*
 UDP     komp1:netbios-ns     *:*
 UDP     komp1:netbios-dgm    *:*
 UDP     komp1:isakmp         *:*
```

Napisz skrypt, który drukuje wszystkie wiersze informujące o połączeniach z portem o numerze 44334.

```
#!/usr/bin/awk -f
BEGIN{ FS = "[: \t]+"
       port = ARGV[2]; ARGV[2]=""  }
NR > 1 && $3 == port
```

Numer portu jest przekazywany do skryptu jako argument. Uruchomienie skryptu, jeśli dane znajdują się w pliku dane.netstat:

```
$ ./skrypt.awk dane.netstat 44334
```

45. Wydrukuj najdłuższy wiersz pliku tekstowego i jego numer.

    Skrypt awk dla przypadku, kiedy jest tylko jeden wiersz o maksymalnej długości:

```
{ if (length($0) > max) { nrmax=NR; max=length($0) } }
END{ print nrmax, max }
```

    Jeżeli wierszy o maksymalnej długości może być wiele, należałoby użyć następującego skryptu:

```
{ wiersze[NR] = $0
  if (length($0) > max)
  {
    max = length($0)
  }
}
END{
  print "Maksymalna dlugosc wiersza:", max
  for (i=1; i<=NR; i++)
  {
    if (length(wiersze[i]) == max)
    {
      print i, wiersze[i]
    }
  }
}
```

46. Wydrukuj najdłuższy wiersz z kilku plików tekstowych. Drukuj także nazwę pliku i numer wiersza w pliku.

    Pierwszy sposób

```
{
  if (FILENAME != plik)
  { nrpliku++
    plik = FILENAME
  }
  dlugosc[FILENAME]++
  if (length($0) > max)
  { max = length($0)
    plik = FILENAME
```

```
    nrwiersza = dlugosc[FILENAME]
    wiersz = $0
  }
}

END{
  print "Plik:", plik
  print "Numer wiersza w pliku:", nrwiersza
  print "Liczba znakow w najdluzszym wierszu:", max
  print "Zawartosc wiersza:"
  print wiersz
}
```

Drugi sposób

```
{ if (FILENAME != plik)
  {
    nrpliku++
    plik = FILENAME
    nazwyplikow[nrpliku] = FILENAME
  }
  dlugosc[FILENAME]++
  if (length($0) > max)
  {
    max = length($0)
    wiersz[numer,FILENAME] = $0
  }
}
END{
  print "Maksymalna dlugosc wiersza:", max
  for (i=1; i<=nrpliku; i++)
  {
    plik = nazwyplikow[i]
    for (j=1; j<=dlugosc[plik]
    {
      if (legth(wiersz[j,plik]) == max)
      {
        print plik, j, wiersz[j,plik]
      }
    }
  }
}
```

47. Plik zawiera dane o miastach należących do różnych krajów:

```
miasto
kraj
region
liczba mieszkańcow

miasto
kraj
region
liczba mieszkańcow
```

Pomiędzy danymi o kolejnych miastach pojawia się pusty wiersz. Napisz skrypt, który wydrukuje dane o tych miastach w PL i DE, które mają więcej niż 100 000 mieszkańców.

```
BEGIN{ RS="\n\n"; FS="\n" }
$2 == "PL" || $2 == "DE" { if ($4 >= 100000)
                           { print $0 } }
```

48. Plik tekstowy zawiera dane o przeprowadzonych rozmowach telefonicznych. Wiersze mają następującą postać:

nr_kier_kraju nr_tel rok miesiac dzien godz min sek min sek koszt

Na przykład:

```
+48 601111222 2004 1 12 22 48 32 2 16 3.45
+49 202233456 2004 4 28  7 13 50 5 12 6.18
```

itd.

(a) Wydrukuj liczbę połączeń z numerem `+48 601111222` w pierwszych 6 miesiącach 2004 roku

```
$4 <= 6 { if ($1 == "+48" && $2 == "601111222")
          { suma++ }
}
END { print suma }
```

(b) Wydrukuj łączny czas połączeń z tym numerem w minutach i sekundach

```
$4 <= 6 { if ($1 == "+48" && $2 == "601111222")
          { suma = suma + $9*60 + $10 }
}
END {
  print "Laczny czas polaczen"
  print int(suma/60), "minut", suma%60, "sekund"
}
```

(c) Wydrukuj średnią długość rozmowy

```
{ suma = suma + $9*60 + $10 }
END{ print "Srednia dlugosc rozmowy:"
     print int(suma/NR), "minut",
           suma%NR, "sekund"
}
```

(d) Wydrukuj liczbę połączeń trwających ponad 1 minutę

```
$9 >= 1 { suma++ }
END{ print suma }
```

(e) Wydrukuj łączny koszt wszystkich połączeń z numerem +48 601111222 w marcu i kwietniu

```
$1 == "+48" && $2 == 601111222 && $4 == 3\
 { suma = suma + $11 }
$1 == "+48" && $2 == 601111222 && $4 == 4\
 { suma = suma + $11 }
END { print suma }
```

49. Dane w pliku ułożone są następująco:

```
nr telefonu
imię
nazwisko

nr telefonu
imię nazwisko
...
```

Między danymi kolejnych abonentów jest pusty wiersz. Dane jednego abonenta mogą znajdować się zarówno w dwóch wierszach, jak i w trzech (patrz wyżej).

Napisz skrypt, który przekształci ten plik do następującej postaci:

```
nr telefonu, imię, nazwisko
nr telefonu, imię, nazwisko
...
```

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

50. Plik tekstowy zawierający parametry konfiguracji pewnego programu ma następującą strukturę:

```
[grafika]
rozdzielczosc=1024x768
kolory=256
...

[uzytkownik]
userid=adam
typuzytkownika=root
...
```

Dane poszczególnych grup parametrów są oddzielone od siebie pustym wierszem. Napisz skrypt drukujący wartość zmiennej, której nazwa jest przekazywana do skryptu jako parametr

```
#!/usr/bin/awk -f
BEGIN{ RS=""; FS="\n"; nazwa=ARGV[2]; ARGV[2]="" }
{ for (i=2; i<=NF; i++) {
    split($i, a, "=")
    if (nazwa == a[1]) { print a[2] }
  }
}
```

Uruchomienie skryptu: `./skrypt.awk plik nazwa_parametru`

51. Skompresuj zawartość wszystkich podkatalogów katalogu bieżącego. Każdy podkatalog powinien trafić do osobnego pliku.

    Niech plik `nazwy_katalogow` zawiera nazwy podkatalogów, po jednej nazwie w wierszu. Oto dwa skrypty, w bash i awk, które realizują to zadanie:

```
#!/usr/bin/bash
a=`cat nazwy_katalogow`
for i in $a
do
  tar zcf $i.tgz $i/*
done
```

```
#!/usr/bin/awk -f
{ a[NR] = $0 }
END{
  for (i=1; i<=NR; i++){
    system("tar zcf "a[i]".tgz "a[i]"/*")
  }
}
```

    Uruchomienie: `./skrypt.awk nazwy_katalogow`.

52. Kursy akcji spółki akcyjnej. W pliku zapisane są dane o kursie akcji w ciągu ostatnich 365 dni, od najnowszych do najstarszych. W pierwszym są dane z bieżącego dnia, w drugim – z poprzedniego. Wydrukuj minimalny i maksymalny kurs akcji z ostatnich 365 dni. Wydrukować informację o zmianie kursu akcji (wielkość przyrostu lub spadku) cen akcji w ciągu ostatniego roku, miesiąca, tygodnia i dnia.

```
NR == 1 { pierwszy = $2; min = $2 }
{ if (
  if ($2 > max)
  {
    max = $2
  }
}
NR == 334 { miesiac = $2 }
NR == 358 { tydzien = $2 }
NR == 364 { dzien = $2 }
```

```
NR == 365 { ostatni = $2 }
END {
  print "Zmiana kursu w ciagu roku: ",
        ostatni - pierwszy
  print "Zmiana kursu w ciagu miesiaca: ",
        ostatni - miesiac
  print "Zmiana kursu w ciagu tygodnia: ",
        ostatni - tydzien
  print "Zmiana kursu w ciagu dnia: ",
        ostatni - dzien
}
```

53. Książka kodów pocztowych. Dane w pliku mają następującą postać:

```
kod
miejscowość
ulica
nr domu1
nr domu2
```

Napisz skrypt, który obsługuje książkę kodów. Na wejściu skrypt może przyjmować parametry w trzech wariantach:

- tylko kod pocztowy
- tylko nazwa miejscowości
- nazwa miejscowości i ulica

```
BEGIN { RS = "\n\n"; FS = "\n"
        if (ARGC == 3)
        {
          a = ARGV[1]; ARGV[1] = ""
          if (a ~ /[0-9][0-9][-]?[0-9][0-9][0-9]/)
  { kod = a }
          else
          { nazwa = a }
        }
        if (ARGC == 4)
        {
          a=ARGV[1]; b=ARGV[2]
          ARGV[1]=""; ARGV[2]=""
```

```
          nazwa = a
          ulica = b
        }
}
a == $1 { print }
a == $2 || a == $2 && b == $3 { print }
```

54. Firma Najlepsza S.A. wytwarza butelkowaną wodę mineralną. Aby dostosować strategię marketingową do potrzeb rynku, Najlepsza S.A. zamawia w firmie specjalizującej się w badaniach rynku dane o sprzedaży swoich produktów.

    Zamawiane dane dostarczane są w plikach tekstowych. Dotyczą one wielkości sprzedaży i są liczone liczbą butelek. Dane obejmują następujące kategorie: marka (Zimna lub Dobra), wielkość butelki (0.75 lub 1.5 litra), typ wody (gazowana, niegazowana), wielkość sklepu (mały, duży), okres, którego dotyczą dane (numer tygodnia w danym roku).

    Zaprojektuj postać pliku danych i podaj kilka przykładowych wierszy.

    W pliku powinny być zapisane w 6 polach: marka, wielkość, typ, wielkość sklepu, okres, liczba butelek, na przykład:

    ```
    Z 0.75 gaz maly 30 10000
    Z 1.5  niegaz duzy 30 20000
    D 0.75 niegaz maly 30 18000
    ...
    ```

    Napisz skrypt, który:

    (a) wydrukuje wielkość sprzedaży wody Zimnej niegazowanej w małych sklepach w pierwszych 4 tygodniach roku

    ```
    $1 == "Z" && $3 == "niegaz" $4 == "maly"\
       && $5 <= 4 { s = s + $6 }
    END { print s }
    ```

    (b) wydrukuje wielkość sprzedaży 2-litrowych butelek wody Dobrej gazowanej w dużych sklepach w całym roku

    ```
    $1 == "D" && $2 == 2 && $3 == "gaz" &&\
    $4 == "duzy" { s = s + $6 }
    END { print s }
    ```

(c) wydrukuje dane obliczone w punkcie (b) jako odsetek całej sprzedaży butelek 2-litrowych wody Dobrej gazowanej we wszystkich sklepach (dużych i małych razem) w całym roku.

```
$1 == "D" && $2 == 2 && $3 == "gaz" && $4 == "duzy"
  { s1 = s1 + $6 }
$1 == "D" && $2 == 2 && $3 == "gaz" { s2 = s2 + $6 }
END { print s1/s2 }
```

55. Plik tekstowy zawiera liczby przedzielone spacją:

(a) Drukuj numer wiersza, w którym suma liczb jest największa, sumę liczb w tym wierszu i zawartość wiersza

```
{ suma = 0
  for (i=1; i<=NF; i++)
  { suma = suma + $i }
  if (suma > max)
  {
    max = suma
    numer = NR
    wiersz = $0
  }
}
END {
  print "nr wiersza:", numer, "suma:", suma
  print wiersz
}
```

(b) Uwzględnij przypadek występowania kilku wierszy o jednakowej, maksymalnej sumie

```
{ suma[NR] = 0
  for (i=1; i<=NF; i++)
  { suma[NR] = suma[NR] + $i }
  if (suma[NR] > max)
  { max = suma[NR] }
  wiersz[NR] = $0
}
```

```
END { for (i=1; i<=NR; i++)
      {
        if (suma[NR] == max)
        { print wiersz[NR] }
      }
}
```

56. W pliku są zapisane dane o sprzedaży samochodów różnych producentów. W kolejnych polach umieszczono nazwę producenta, numer miesiąca, liczbę sprzedanych samochodów:

```
Fiat Panda 1 100
Fiat Panda 2 120
Fiat Panda 3 150
...
Fiat Punto 1 200
Fiat Punto 2 190
...
Ford Fiesta 1 120
Ford Fiesta 2 130
...
```

(a) Drukuj wielkość sprzedaży wszystkich samochodów Fiata w pierwszych sześciu miesiącach roku

```
$1 == "Fiat" && $3 <= 6 { suma = suma + $4 }
END { print suma }
```

(b) Drukuj liczbę egzemplarzy modelu Fiesta sprzedanych w całym roku

```
$2 == "Fiesta" { suma = suma + $4 }
END { print suma }
```

(c) Drukuj nazwę producenta, który sprzedał najwięcej samochodów w jednym miesiącu

```
{ s[$1,$3] = s[$1,$3] + $4
}
END {
  for (petla po producentach)
  {
    for (i=1; i<=12; i++)
    {
      suma = suma + s[...,i]
      if (suma > max)
      {
        producent = ...
      }
    }
  }
  print producent
}
```

57. Oto lista pociągów odjeżdżających z Poznania do Warszawy. Podane są: stacja początkowa, godzina odjazdu, rodzaj pociągu, stacja końcowa, czas przyjazdu do stacji końcowej i czas podróży.

```
Poznan  2:07    P    Warszawa Centralna   5:34  3:27
Poznan  7:15    IC   Warszawa Centralna   9:55  2:40
Poznan  7:45    EX   Warszawa Zachodnia  10:34  2:49
Poznan  8:45    IC   Warszawa Centralna  11:25  2:40
Poznan  9:45    EC   Warszawa Centralna  12:32  2:47
```

(a) Drukuj dane o pociągach InterCity (IC)

```
$3 == "IC"
```

(b) Drukuj dane o pierwszym pociągu odjeżdżającym po godzinie 6:00

```
egrep -m 1 'Poznan[ ][ ]*([6-9]|1[0-9]|2[0-4])' plik
```

(c) Drukuj dane o pociągu z najkrótszym czasem podróży

```
BEGIN { FS=":|[ ]*"; min=1000 }
{ czas = $9*60+$10
  if (czas < min)
  {
    min = czas
    wiersz = $0
  }
}
END {
  print "Najkrotszy czas podrozy:"
  print int(min/60), "godzin", min%60, "minut"
  print wiersz
}
```

Jeżeli w pliku brak ostatniej kolumny z informacją o czasie przejazdu, skrypt miałby następującą postać:

```
BEGIN{ FS=":|[ ]*"; min = 1000 }
{ print $8, $7, $2, $3
  czas = $8+($7-$2)*60-$3
  if (czas < min)
  {
    min = czas
    wiersz = $0
  }
}
END{ print min
     x = min/60
     g = int(x)
     m = min%60
     print "Najkrotszy czas podrozy: "
     print g, "godzin", m, "minut"
     print wiersz
}
```

(d) Drukuj informację, ile pociągów dojeżdża do stacji Warszawa Centralna (liczba wierszy zawierających tę nazwę stacji)

```
/Warszawa Centralna/ { n++ }
END{ print n}
```

58. Oto dane o sprzedaży książek w księgarni. Dwie ostatnie pozycje w każdym wierszu to cena jednego egzemplarza i liczba sprzedanych egzemplarzy.

```
Programowanie w jezyku C
A. Abacki
Wyd. Enigma
2000
50 zl
7

UNIX
B. Babacki
Wyd. PWN
2000
40 zl
12

Perl
C. Cabacki
Wyd. Enigma
2001
30 zl
4
```

(a) Drukuj liczbę wszystkich sprzedanych książek

```
END{ print NR }
```

(b) Drukuj sumę przychodów

```
BEGIN{ FS=";;" }
{ suma = suma + $5*$6 }
END{ print suma }
```

(c) Drukuj wielkość przychodów ze sprzedaży książek wydawnictwa Enigma

```
BEGIN{ FS=";;" }
$3 ~ /Enigma/ { suma = suma + $5*$6 }
END{ print suma }
```

(d) Drukuj wielkość sprzedaży (liczbę egzemplarzy) książek wydanych w 2000 r.

```
BEGIN{ FS=";;" }
$4 == 2000 { suma = suma + $6 }
END{ print suma }
```

59. Dany jest zwykły plik tekstowy. Drukuj, ile jest słów rozpoczynających się wielką literą.

```
{ for (i=1; i<=NF; i++)
  {
    if ($i ~ /^[A-Z]/) { suma++ }
  }
}
END{ print suma }
```

60. Tygodniowy plan zajęć. Informacje o rozkładzie zajęć zapisane są w następujący sposób:

```
Pon
8
9    Matematyka wyklad 6
10  Informatyka Audytorium Wschodnie
...
18

Pt
8  J. angielski 10
...
18
```

Zakładamy, że zajęcia rozpoczynają się o pełnej godzinie. Jeżeli o danej godzinie nie ma zajęć, nie ma żadnego wpisu obok numeru godziny.

(a) Drukuj łączną liczbę godzin przeznaczoną na zajęcia w każdym dniu i liczbę godzin przypadającą na cały tydzień (sumę)

Dane:

```
Pon
8
9 Matematyka wyklad 6
10 Informatyka wyklad Audytorium Wschodnie
11 Metody numeryczne laboratorium 61
12 Metody numeryczne laboratorium 61
13 Telekomunikacja wyklad Audytorium Maximum
14 Telekomunikacja wyklad Audytorium Maximum
15 Jezyki skryptowe wyklad Audytorium Maximum
16 Jezyki skryptowe wyklad Audytorium Maximum
17
18

Wt
8
9
10 WF Sala gimnastyczna
11 WF Sala gimnastyczna
12
13 Filozofia wyklad Audytorium Zachodnie
14 Filozofia wyklad Audytorium Zachodnie
15 Filozofia cwiczenia 8
16 Filozofia cwiczenia 8
16
17
18

Sr
8
9
10
11
12
13
14
15
16
17
18
```

```
Czw
8
9
10
11
12
13
14
15
16
17
18

Pt
8 Jezyk angielski 10
9 Sieci komputerowe laboratorium 64
10 Sieci komputerowe laboratorium 64
11 Fizyka wyklad Audytorium Maximum
12 Fizyka wyklad Audytorium Maximum
13
14
15
16
17
18
```

Skrypt:

```
BEGIN{ RS="\n\n"; FS="\n"}
{ suma = 0
  for(i=2; i<=NF; i++)
  {
    if (length($i) > 4)
    {
      suma++
    }
  }
  print $1, " Suma godzin zajec: ", suma
  suma_tydzien = suma_tydzien + suma
}
END{
  print "Liczba godzin zajec w calym tygodniu:"
  print suma_tydzien }
```

Wynik:

```
Pon  Suma godzin zajec:  8
Wt  Suma godzin zajec:  6
Sr  Suma godzin zajec:  0
Czw  Suma godzin zajec:  0
Pt  Suma godzin zajec:  5
Liczba godzin zajec w calym tygodniu:  19
```

(b) Drukuj plan dotyczący tego dnia, w którym odbywa się najwięcej zajęć (załóżmy, że maksymalne obciążenie dotyczy tylko jednego dnia)

```
BEGIN{ RS="\n\n"; FS="\n"}
{ suma = 0
  tydzien[NR] = $0
  for (i=2; i<=NF; i++)
  {
    if (length($i) > 4)
    {
      suma++
    }
  }
  if (suma > max)
  {
    max = suma
    dzienmax = NR
  }
}
END{
  print tydzien[dzienmax]
}
```

Wynik:

```
Pon
8
9 Matematyka wyklad 6
10 Informatyka wyklad Audytorium Wschodnie
11 Metody numeryczne laboratorium 61
12 Metody numeryczne laboratorium 61
```

```
13 Telekomunikacja wyklad Audytorium Maximum
14 Telekomunikacja wyklad Audytorium Maximum
15 Jezyki skryptowe wyklad Audytorium Maximum
16 Jezyki skryptowe wyklad Audytorium Maximum
17
18
```

(c) Wykonaj punkt (b), rezygnując z założenia, że maksymalne obciążenie przypada na jeden dzień tygodnia. Drukuj informacje o zajęciach we wszystkich dniach, na które przypada maksymalne obciążenie

```
BEGIN{ RS="\n\n"; FS="\n"}
{ suma = 0
  tydzien[NR] = $0
  for (i=2; i<=NF; i++)
  {
    if (length($i) > 4)
    {
      suma++
    }
  }
  if (suma > max)
  {
    max = suma
    dzienmax = NR
  }
  sumadzien[NR] = suma
}
END{
  for (i=1; i<=NR; i++)
  {
    if (sumadzien[i] == max)
    {
      print tydzien[i]
    }
  }
}
```

Dane:

```
Pon
8
9 Matematyka wyklad 6
10 Informatyka wyklad Audytorium Wschodnie
11
12
13 Telekomunikacja wyklad Audytorium Maximum
14 Telekomunikacja wyklad Audytorium Maximum
15 Jezyki skryptowe wyklad Audytorium Maximum
16 Jezyki skryptowe wyklad Audytorium Maximum
17
18

Wt
8
9
10 WF Sala gimnastyczna
11 WF Sala gimnastyczna
12
13 Filozofia wyklad Audytorium Zachodnie
14 Filozofia wyklad Audytorium Zachodnie
15 Filozofia cwiczenia 8
16 Filozofia cwiczenia 8
16
17
18

Sr
8
9
10
11
12
13
14
15
16
17
18
```

```
Czw
8
9
10
11
12
13
14
15
16
17
18

Pt
8 Jezyk angielski 10
9 Sieci komputerowe laboratorium 64
10 Sieci komputerowe laboratorium 64
11 Fizyka wyklad Audytorium Maximum
12 Fizyka wyklad Audytorium Maximum
13
14
15
16
17
18
```

Wynik:

```
Pon
8
9 Matematyka wyklad 6
10 Informatyka wyklad Audytorium Wschodnie
11
12
13 Telekomunikacja wyklad Audytorium Maximum
14 Telekomunikacja wyklad Audytorium Maximum
15 Jezyki skryptowe wyklad Audytorium Maximum
16 Jezyki skryptowe wyklad Audytorium Maximum
17
18
Wt
8
9
```

```
10 WF Sala gimnastyczna
11 WF Sala gimnastyczna
12
13 Filozofia wyklad Audytorium Zachodnie
14 Filozofia wyklad Audytorium Zachodnie
15 Filozofia cwiczenia 8
16 Filozofia cwiczenia 8
16
17
18
```

(d) Drukuj rozkład zajęć w innej postaci. Pomiń te godziny na początku i na końcu dnia, w których nie odbywają się zajęcia. Drukowane mają być wszystkie informacje od pierwszej do ostatniej godziny zajęć. Jeżeli między zajęciami istnieją godziny wolne, informacja o nich ma być drukowana.

```
BEGIN{ RS="\n\n"; FS="\n"}
{
  pierwsza[NR] = 0
  ostatnia[NR] = 0
  tydzien[NR,1] = $1
  for (i=2; i<=NF; i++)
  {
    tydzien[NR,i] = $i
    if (length($i) > 4 && pierwsza[NR] == 0)
    {
      pierwsza[NR] = i
    }
    if (length($i) > 4)
    {
      ostatnia[NR] = i
    }
  }
}
END{
  for (j=1; j<=NR; j++)
  {
    print tydzien[j,1]
    for (i=pierwsza[j]; i<=ostatnia[j]; i++)
    {
```

```
      print tydzien[j,i]
    }
  }
}
```

Wynik:

```
Pon
9 Matematyka wyklad 6
10 Informatyka wyklad Audytorium Wschodnie
11 Metody numeryczne laboratorium 61
12 Metody numeryczne laboratorium 61
13 Telekomunikacja wyklad Audytorium Maximum
14 Telekomunikacja wyklad Audytorium Maximum
15 Jezyki skryptowe wyklad Audytorium Maximum
16 Jezyki skryptowe wyklad Audytorium Maximum
Wt
10 WF Sala gimnastyczna
11 WF Sala gimnastyczna
12
13 Filozofia wyklad Audytorium Zachodnie
14 Filozofia wyklad Audytorium Zachodnie
15 Filozofia cwiczenia 8
16 Filozofia cwiczenia 8
Sr

Czw

Pt
8 Jezyk angielski 10
9 Sieci komputerowe laboratorium 64
10 Sieci komputerowe laboratorium 64
11 Fizyka wyklad Audytorium Maximum
12 Fizyka wyklad Audytorium Maximum
```

61. Załóżmy, że dane dotyczące poprzedniego zadania są ułożone inaczej. Informacje dotyczące poszczególnych dni tygodnia są umieszczone w oddzielnych plikach.

    (a) Zaprojektuj postać tego pliku, uwzględniając godzinę rozpoczęcia zajęć, nazwę przedmiotu, która może składać się z wielu wyrazów oraz nazwę lub numer sali, w której odbywają się zajęcia.

Oto przykład pliku Pon.

```
8
9;Matematyka wyklad;6
10;Informatyka wyklad;Audytorium Wschodnie
11;Metody numeryczne laboratorium;61
12;Metody numeryczne laboratorium;61
13;Telekomunikacja wyklad;Audytorium Maximum
14;Telekomunikacja wyklad;Audytorium Maximum
15;Jezyki skryptowe wyklad;Audytorium Maximum
16;Jezyki skryptowe wyklad;Audytorium Maximum
17
18
```

(b) Drukuj nazwy zajęć odbywających się jednego, dowolnie wybranego dnia tygodnia – tylko nazwy zajęć, żadnych innych danych

```
BEGIN{ FS=";" }
{ print $2 }
```

(c) Drukuj nazwy zajęć z całego tygodnia tak, by każda nazwa pojawiała się jeden raz

```
awk 'BEGIN{ FS=";"} {print $2}' \
  Pon Wt Sr Czw Pt | sort | uniq
```

62. Napisz skrypt łączący dwa sąsiednie wiersze w jeden, pod warunkiem, że oba są niepuste.

```
# Laczy dwa wiersze tekstu w jeden, w calym pliku.
# Puste wiersze pozostaja bez zmian

{ line[NR] = $0 }

END {
  i=1
  while (i<=NR)
  {
    if (length(line[i]) > 1)
```

```
      if (length(line[i+1]) > 1)
      { print line[i] " " line[i+1]
        i=i+2
      }

    if (length(line[i]) < 2)
      print "" i=i+1

    if(length(line[i]) > 1 )
      if (length(line[i+1]) < 2)
        { print line[i]
          print ""
          i=i+2
        }
  }
}
```

63. Plik zawiera fragmenty oddzielone od reszty tekstu parami łańcuchów, np. "początek"i "koniec". Oblicz średnią długość jednego takiego bloku i średnią liczbę znaków w jednym wierszu bloku.

```
# Liczy srednia dlugosc fragmentow tekstu zawartych
# miedzy wzorcami pocz i koniec
# Na wyjsciu:
#    liczba fragmentow
#    liczba wszystkich wierszy
#    srednia liczba wierszy/fragment
#    liczba wszystkich znakow
#    srednia liczba znakow/wiersz
#    srednia liczba znakow roznych od spacji/wiersz
#
# n - liczba par pocz-koniec w tekscie
# wiersze - suma wierszy we wszystkich fragmentach
# znaki - suma znakow we wszystkich fragmentach
#        (lacznie ze spacjami)
# spacje - suma spacji we wszystkich fragmentach

BEGIN{ FS=""; pocz=ARGV[2]; koniec=ARGV[3]
       ARGV[2]=""; ARGV[3]="" }
```

```
pocz,koniec {
  if ( $0 !~ pocz && $0 !~ koniec )
  { wiersze++
    znaki = znaki + length($0)
    spacje = spacje + policzspacje()
  }
  else n++
}

END{ printf("%8d %s", n/2, "fragmentow\n")
     printf("%8d %s", wiersze, "wierszy\n")
     printf("%8.2f %s", wiersze/(n/2),
            "wierszy na fragment\n")
     printf("%8d %s", znaki, "znakow\n")
     printf("%8.2f %s", znaki/wiersze,
            "znakow na wiersz\n")
     printf("%8.2f %s %s", (znaki - spacje)/wiersze,
            "srednia liczba znakow roznych",
            "od spacji w wierszu\n")
}

function policzspacje()
{
  suma = 0
  for (i=1; i<=NF; i++)
    if ($i == " ") suma++
  return suma
}
```

Uruchomienie skryptu: `./skrypt.awk plik pocz koniec`.

64. Wykonaj poprzednie zadanie, tym razem obliczając średnią liczbę słów w jednym bloku oraz średnią liczbę znaków w jednym słowie.

```
# Liczy srednia dlugosc fragmentow tekstu zawartych
# miedzy wzorcami pocz i koniec
# Na wyjsciu:
#    liczba fragmentow
#    liczba wszystkich wierszy
#    liczba wszystkich slow
#    liczba wszystkich znakow w slowach
```

```
#    srednia liczba wierszy/fragment
#    srednia liczba slow/wiersz
#    srednia liczba znakow/slowo
# n - liczba par pocz-koniec w tekscie
# wiersze - suma wierszy we wszystkich fragmentach
# slowa - suma slow we wszystkich fragmentach
# znaki - suma znakow we wszystkich slowach

BEGIN{ pocz=ARGV[2]; koniec=ARGV[3]
       ARGV[2]=""; ARGV[3]="" }

pocz,koniec {
  if ( $0 !~ pocz && $0 !~ koniec )
  { wiersze++
    slowa = slowa + NF
    for (i=1; i<=NF; i++) znaki = znaki + length($i)
  }
  else n++
}

END{
  printf("%8d %s", n/2, "fragmentow\n")
  printf("%8d %s", wiersze, "wierszy\n")
  printf("%8d %s", slowa, "slow\n")
  printf("%8d %s", znaki, "znakow\n")
  printf("%8.2f %s", wiersze/(n/2), "wierszy na fragment\n")
  printf("%8.2f %s", slowa/wiersze, "slow na wiersz\n")
  printf("%8.2f %s", znaki/slowa, "znakow na slowo\n")
}
```

Uruchomienie skryptu: `./skrypt.awk plik pocz koniec`.

Inny sposób polega na odmiennym wyborze separatora rekordów `RS`:

```
BEGIN{ pocz=ARGV[2]; koniec=ARGV[3]
       RS=ARGV[3]"\n[.]*\n"ARGV[2]
       ARGV[2]=""; ARGV[3]="" }
{ wiersze++
  slowa = slowa + NF
  for (i=1; i<=NF; i++) znaki = znaki + length($i)
}
```

Blok `END` pozostaje bez zmian.

65. Plik tekstowy ma następującą postać:

```
haslo1
odsylacz
pierwszy wiersz objasnienia
drugi wiersz objasnienia
...
ostatni wiersz objasnienia

haslo2
odsylacz
pierwszy wiersz objasnienia
drugi wiersz objasnienia
...
ostatni wiersz objasnienia
itd.
```

Na przykład:

```
Wydzial Fizyki
http://www.fizyka.amu.edu.pl/
Witryna Wydzialu Fizyki
UAM, Poznan

The Free Software Foundation
http://www.fsf.org/
Witryna ruchu fundacji FSF, sponsora projektu GNU

Python
http://www.python.org/
Zasoby dotyczace jezyka Python
```

Napisz skrypt przekształcający ten plik tekstowy do formatu html. Wszystkie hasła mają być pisane pogrubioną czcionką. Między objaśnieniami a kolejnym hasłem powinien być jeden pusty wiersz. Jeżeli odsyłacz nie rozpoczyna się ciągiem `http://`, wskazuje inne hasło w tym samym dokumencie i powinien być stosownie sformatowany.

```
{ wiersz[NR] = $0 }
END {
  print "<html><head>"
  print "<meta http-equiv=\"content-type\""
  print "content=\"text/html;"
  print "charset=ISO-8859-1\">"
  print "<title>Adresy internetowe</title></head>"
  print "<body>"
  print "<h2>Adresy</h2><br>"

  i=1
  while( i<=NR )
  {
    if( wiersz[i-1] == "" )
    {
      print "<b><a href=\""wiersz[i+1]"\">"\
             wiersz[i]"</a></b><br>"
      i=i+2
    }
    print wiersz[i]"<br>"
    i=i+1
  }
  print "</body></html>"
}
```

Oto otrzymany kod html dla przykładowego pliku:

```
<html><head>
<meta http-equiv="content-type"
content="text/html;
charset=ISO-8859-1">
<title>Adresy internetowe</title></head>
<body>
<h2>Adresy</h2><br>
<b><a href="http://www.fizyka.amu.edu.pl/">
Wydzial Fizyki</a></b><br>
Witryna Wydzialu Fizyki<br>
UAM, Poznan<br>
<br>
<b><a href="http://www.fsf.org/">
The Free Software Foundation</a></b><br>
```

```
Witryna ruchu fundacji FSF, sponsora projektu GNU<br>
<br>
<b><a href="http://www.python.org/">Python</a></b><br>
Zasoby dotyczace jezyka Python<br>
</body></html>
```

66. W pliku zawarte są podstawowe dane dotyczące obciążenia serwera w ciągu jednej doby: godzina, liczba użytkowników, liczba megabajtów zajętej pamięci operacyjnej oraz obciążenie procesora:

```
0:01  8  357 2.84
0:02  9  361 3.07
...
9:20 31  805 4.77
...
15:42   35 1192 6.18
...
24:00  7  316 2.26
```

(a) Zmień postać pliku. Godziny i minuty mają się pojawiać osobno:

```
 0  1  8  357 2.84
 0  2  9  361 3.07
...
 9 20 31  805 4.77
...
15 42   35 1192 6.18
...
24  0  7  316 2.26
```

Skrypt:

```
/^.:/s/^/ /
s/:0/  /
s/:/ /
```

(b) Drukuj maksymalną liczbę użytkowników w systemie tego dnia

```
{ if ($3 > max) { max = $3 }
}
END{ print max }
```

(c) Drukuj te wiersze pliku, w których liczba użytkowników różni się od maksymalnej co najwyżej o 5

```
BEGIN{ RS="\n\n"; FS="\n"}
{ suma = 0
  for(i=2; i<=NF; i++)
  {
    if (length($i) > 4)
    { suma++ }
  }
  print $1, " Suma godzin zajec: ", suma
  suma_tydzien = suma_tydzien + suma
}
END{
  print "Liczba godzin zajec w calym tygodniu:"
  print suma_tydzien }
```

(d) Drukuj uśrednione dane dla każdego okresu 60-minutowego

```
{ u[$1] = u[$1] + $3
  mem[$1] = mem[$1] + $4
  p[$1] = p[$1] + $5
}
END{
  for (i=1; i<=24; i++)
  {
    print i, u[i]/60, mem[i]/60, p[i]/60
  }
}
```

67. W pliku zapisane są dane dotyczące kursu akcji spółki akcyjnej notowanej na giełdzie. Dane dotyczą całego roku:

```
1
2  45.24
3  44.18
4  44.12
5  44.80
6  44.84
```

```
7
8
9  44.97
10 45.02
11 45.02
12 45.20
13 43.50
14
15
16 43.90
17 43.90
18 44.55
19 44.90
20 45.30
21
22
23 45.80
24 46.05
25 46.15
26 46.40
27 46.50
28
29
30 46.30
31 46.35
32 46.20
33 46.60
34 46.55
35
36
...
365
```

W pierwszej kolumnie znajduje się numer dnia w roku, a w drugiej kurs na zakończenie dnia. W dni wolne od pracy kurs nie jest zapisywany i drugie pole jest puste. Drukuj dane o najdłuższym trendzie niemalejącym (liczą się tylko dni robocze, dni wolne od pracy nie są wliczane do długości trendu), kiedy kurs zamknięcia nie jest niższy od kursu z poprzedniego dnia. Dane te powinny zawierać numer dnia, w którym trend się rozpoczął, kurs z tego dnia oraz numer ostatniego dnia tego trendu i kurs zamknięcia tego dnia.

```
BEGIN{ m=0 }
$2 != "" { m++; w[m,1]=NR; w[m,2]=$2 }
END{
  rosnacy=0
  nrciagu=0
  for (k=2; k<=m; k++)
  {
    if (w[k,2] >= w[k-1,2])
    {
      if (rosnacy==0)
      {
        nrciagu++
        i=1
        rosnacy=1
        dlugosc[nrciagu]=1
        pocz[nrciagu]=w[k-1,1]
        x[nrciagu,i]=w[k-1,2]
      }
      if (rosnacy==1)
      {
        i++
#        print k,i,nrciagu
        dlugosc[nrciagu]=dlugosc[nrciagu]+1
koniec[nrciagu]=w[k,1]
        x[nrciagu,i]=w[k,2]
      }
    }
    if (w[k,2] < w[k-1,2]) { rosnacy=0 }
  }
  for (j=1; j<=nrciagu; j++)
  {
    if (dlugosc[j] > max)
    {
      max = dlugosc[j]
      numer=j
    }
  }
  print pocz[numer],x[numer,1],
        koniec[numer],x[numer,max]
}
```

68. Pliki `A` i `B` mają równą liczbę wierszy. W pliku `A` dane są umieszczone w 4 kolumnach oddzielonych spacjami. W pliku `B` dane mają od 5 do 7 kolumn danych.

    (a) Dopisz przedostatnią kolumnę z każdego wiersza pliku `B` na końcu wiersza o tym samym numerze w pliku `A`.

```
FILENAME == "A" { a[NR] = $0; l++ }
FILENAME == "B" { b[NR-l] = $(NF-1) }
END{
  for (i=1; i<=l; i++)
  { print a[i], b[i] }
}
```

    Uruchomienie skryptu:

```
awk -f skrypt.awk A B
```

    (b) Utwórz inny plik, w którym kolejne wiersze z pliku `A` i `B` przeplatają się: po wierszu z pliku `A` następuje wiersz z pliku `B` itd.

```
FILENAME == "A" { a[NR] = $0; l++ }
FILENAME == "B" { b[NR-l] = $0 }
END{
  for (i=1; i<=l; i++)
  {
    print a[i]
    print b[i]
  }
}
```

    Skrypt należy uruchomić podobnie, jak w punkcie (a).

69. W pliku zapisane są nazwy i ceny towarów:

```
A 15.80
B 20.50
C 18.90
D  5.00
E 120
...
```

Posortuj dane względem ceny towaru tak, by kolejność pól w każdym wierszu pozostała niezmieniona. Najdroższe towary powinny pojawiać się na początku pliku.

```
awk '{ print $2, $1 }' plik |\
  sort -nr | awk '{ print $2, $1 }'
```

70. Dane o książkach mają następującą postać:

```
83-01-13403-8
PWN
2001
Surdykowski
Jerzy
brak
Duch Rzeczypospolitej

83-204-1086-X
WNT
1991
Silvester
Peter
P.
System operacyjny Unix

83-02-13325-2
PWN
2000
brak
brak
brak
Encyklopedia Krakowa

83-7279-280-1
Mikom
2002
Hontanon
Ramon
J.
Bezpieczenstwo systemu Linux
```

```
83-204-2015-6
WNT
1995
Bach
Maurice
J.
Budowa systemu operacyjnego UNIX

83-7279-299-2
Mikom
2003
Floyd
Michael
brak
Poznaj XSLT
```

(a) Drukuj wiersze zawierające informacje o książkach wydawnictwa WNT.

```
grep '^[0-9-];;WNT' plik
```

(b) Drukuj dane o książkach nie mających autora

```
awk '$5 == ""' plik
```

(c) Pierwsze dwa człony numeru ISBN książki identyfikują wydawnictwo. Na przykład, numery ISBN wydawnictwa PWN rozpoczynają się ciągiem `83-01-`. Napisz skrypt drukujący te rekordy, w których numery ISBN nie są zgodne z nazwą wydawnictwa w drugim polu rekordu.

(d) Zapisz dane dotyczące różnych wydawnictw w oddzielnych plikach.

```
{ print $0 >> $2 }
```

71. Mamy dwa pliki zawierające adresy www. Adresy z pierwszego pliku mają być dopisane do drugiego (nie dopisywać jednak, jeśli ten sam adres już znajduje się w drugim pliku).

```
n=`cat $1 | wc -l`
i=0
while [ $i -lt $n ]
do
  i=$(($i+1))
  echo `head -$i $1 | tail -1` >> $2
done
sort $2 | uniq >> $2.tmp
mv $2.tmp $2
```

Uruchomienie skryptu:

```
./skrypt.sh plik1 plik2
```

72. Napisz skrypt, który będzie drukował informacje o stopniu obciążenia komputera (`cat /proc/loadavg`). Oprócz wiersza tekstu informacja powinna być drukowana w postaci graficznej. Na przykład, jedna gwiazdka * mogłaby odpowiadać obciążeniu 0.1.

73. Utwórz plik zawierający dane o rozmowach zagranicznych (nr kier. kraju różny od 48).

    (a) Za pomocą awk:

    ```
    awk '$0 !~ /^+48/ { print >> "zagraniczne" }' rozmowy
    ```

    (b) Za pomocą sed:

    ```
    sed -n '/^\+48/!p' rozmowy >> zagraniczne
    ```

74. Napisz skrypt, który przetworzy plik uzyskany w poprzednim zadaniu tak, by dane o rozmowach z różnymi krajami znalazły się w różnych plikach. Nazwy tych plików powinny zawierać numer kierunkowy kraju (ale bez znaku +), rok, miesiąc.

```
# rozmowy.sh
sed 's/^\+//; /[0-9][0-9]*[ ]/s/[ ].*$//' rozmowy\
  > numery_kier_krajow
n=`cat numery_kier_krajow | wc -l`
i=0
```

```
while [ $i -lt $n ]
do
  i=$(($i+1))
  nr=`head -$i numery_kier_krajow | tail -1`
  awk -f rozmowy.awk $nr rozmowy >> rozmowy.$nr
done
```

```
# rozmowy.awk
BEGIN { numerkier = ARGV[1]; ARGV[1] = "" }
{ d=length($1)
  if (substr($1,2,d) == numerkier) { print $0 }
}
```

75. Dowolnym sposobem wydrukuj listę numerów, z którymi się łączono. Każdy numer może pojawić się w tym pliku tylko raz. Posortuj tę listę numerów tak, by numery z poszczególnych krajów sąsiadowały ze sobą.

```
awk '{ print $1, $2 }' | sort | uniq
```

76. Książka telefoniczna (tylko dla numerów stacjonarnych) zawiera następujące dane dla każdego abonenta: numer kierunkowy, numer lokalny, nazwa abonenta (imię i nazwisko lub nazwa instytucji lub firmy), miejscowość, kod pocztowy, nazwa ulicy i numer domu.

    (a) Przedstaw (jak najbardziej precyzyjnie) format pliku tekstowego zawierającego te dane.

    - rekordy są wielowierszowe
    - separator rekordów: `RS="\n\n"`
    - separator pól: `FS="\n"`
    - numer kierunkowy
    - numer lokalny
    - nazwa
    - miejscowość
    - kod pocztowy
    - nazwa ulicy
    - numer domu i mieszkania

(b) Napisz skrypt obsługujący tę książkę. Skrypt powinien drukować pełne dane abonenta. Do skryptu mogą być przekazane parametry w dwóch wariantach:

• numer telefonu w postaci nr_kierunkowy nr_lokalny, na przykład `./skrypt.sh 61 8601122`

```
BEGIN { RS="\n\n"; FS="\n"
        nrkier=ARGV[1]; nrlok=ARGV[2];
        ARGV[1]=""; ARGV[2]="" }
nrkier == $1 && nrlok == $2 { print }
```

• nazwa miejscowości oraz nazwa abonenta (lub część nazwy), na przykład

`./skrypt.sh Poznan Ewa Kowalska`

`./skrypt Gniezno szkola`

Nazwa miejscowości i nazwa abonenta mogą być pisane zarówno wielkimi, jak i małymi literami. Należy pominąć polskie znaki diakrytyczne, tzn. zamiast ą pisać a, zamiast ć pisać c itd.

```
BEGIN { RS="\n\n"; FS="\n"
        miejsc=ARGV[1]; nazwa=ARGV[2];
        ARGV[1]=""; ARGV[2]="" }
miejsc == $6 && nazwa ~ $3 { print }
```

77. Napisz skrypt obsługujący dopisywanie danych do uproszczonej książki telefonicznej (tylko nazwa abonenta i numer telefonu). Dopisywane dane są umieszczone w innym pliku tekstowym. Zakładamy, że książka dotyczy tylko numerów z obszaru Polski. Można wybrać dowolną strukturę danych w obu plikach. Skrypt powinien sprawdzać poprawność numeru:

• czy liczba cyfr numeru kierunkowego wynosi 2, przy czym pierwsza cyfra jest różna od zera

```
echo "Wpisz nazwe abonenta:"
read nazwa
echo "Wpisz numer kierunkowy:"
read nrkier
echo "Wpisz numer lokalny:"
```

```
read nrlok
if [ ! `echo "$nrkier" | grep '^[1-9][0-9]$'` ]
then
  echo "Niepoprawny numer kierunkowy"
else
  echo "$nazwa $nrkier $nrlok" >> ksiazka-telefoniczna
fi
```

• czy liczba cyfr numeru lokalnego wynosi 7

```
if [ ! `echo "$nrlok" | grep '^[1-9][0-9]\{6\}$'` ]
then
  echo "blad"
else
  ...
fi
```

W razie niespełnienia któregoś z tych trzech warunków należy drukować krótki komunikat o błędzie. Spróbuj zezwolić na wpisywanie numeru lokalnego w każdej z następujących postaci: 1234567, 123 4567, 123-4567, 1234 567, 1234-567, 12 34 567, 12-34-567, 123 45 67, 123-45-67.

```
echo "Wpisz nazwe abonenta"
read nazwa
echo "Wpisz numer kierunkowy"
read nrkier
echo "Wpisz numer lokalny"
read nrlok
if [ ! `echo "$nrlok" | egrep \
  '^[1-9][0-9].*[0-9].*[0-9].*[0-9].*[0-9][0-9]$'` ]
then
  echo "blad"
else
  if [ `echo "$nrlok" | egrep '[^0-9 -]'` ]
  then
    echo "Niedozwolone znaki w numerze lokalnym"
  else
    echo "$nazwa $nrkier $nrlok" >> ksiazka-telefoniczna
  fi
fi
```

78. Napisz skrypt, który skompresuje poleceniem `gzip plik` wszystkie pliki z katalogu, którego nazwa (ścieżka) jest przekazywana do skryptu jako parametr. Jeżeli skrypt zostanie uruchomiony bez podania żadnego parametru, kompresowane mają być pliki z bieżącego katalogu. Należy sprawdzać, czy po kompresji rozmiar pliku jest mniejszy niż przed kompresją (w przypadku małych plików kompresja nie zawsze skutkuje zmniejszeniem rozmiaru pliku). Jeżeli po kompresji rozmiar pliku jest większy, plik powinien pozostać w nieskompresowanej postaci.

```
#!/usr/bin/bash
if [ $# -ne 0 ]
then
  kat=$1
else
  kat="."
fi
for i in $kat/*
do
  echo $i
  if [ ! `echo "$i" | grep '.gz$'` ]
  then
    echo $0

#   Pomijamy plik, z ktorego uruchomiono skrypt
    if [ "$i" != "$0" ]
    then
      gzip $i
    fi
  fi
done
```

79. Napisz skrypt, który wydrukuje łączny rozmiar wszystkich plików w katalogu, następnie uruchomi skrypt wykonujący kompresję, po czym znów obliczy i wydrukuje łączny rozmiar plików w katalogu oraz stopień kompresji w procentach lub w skali od 0 do 1.

```
rozmiar=`ls -al $1 |\
  awk '{ suma = suma + $5 } END{ print suma }'`
./kompresuj.sh $1
```

```
nowyrozmiar=`ls -al $1 | awk '{ suma = suma + $5 }\
  END{ print suma }'`
echo "Rozmiar przed kompresja: $rozmiar"
echo "Rozmiar po kompresji: $nowyrozmiar"
stopien-kompresji=nowyrozmiar/rozmiar
```

80. Napisz polecenie lub skrypt, który wydrukuje adresy stron internetowych zawarte w następującym fragmencie tekstu:

```
www.1keydata.com/sql/resources-awk.html - 7k
 - Kopia - Podobne strony
home.t-online.de/home/berndfinger/awk.htm - 5k
 - Kopia - Podobne strony
homepage1.nifty.com/kazuf/awklink.html - 3k
 - Kopia - Podobne strony
FAQ - comp.lang.awk. ...
dir.yahoo.com/Computers_and_Internet/
Programming_and_Development/
Languages/awk/ - 18k - 24 Maj 2004 - Kopia - Podobne strony
www.possibility.com/Perl/ - 29k - Kopia - Podobne strony
(Usenet). ... You are here: RayCosoft.com >
 Info/Support > Links to Unix
Resources. Top of Page. ...
www.raycosoft.com/rayco/support/resources.html - 48k
 - Kopia - Podobne strony
www-106.ibm.com/developerworks/library/l-awk1.html - 35k
 - Kopia - Podobne
strony
www-106.ibm.com/developerworks/library/l-awk2.html - 39k
 - Kopia - Podobne
strony
[ Wi?cej wyników z www-106.ibm.com ]
www.kohala.com/start/troff/troff.html - 7k
 - Kopia - Podobne strony
goanna.cs.rmit.edu.au/~fscholer/resources.php - 6k
 - Kopia - Podobne strony
```

81. Drukuj wszystkie wiersze pliku po usunięciu drugiego pola z każdego wiersza.

```
{ for (i=1; i<=NF; i++)
  {
    if (i != 2)
    { printf("%s",$i) }
```

```
  }
  printf("\n")
}
```

82. Napisz skrypt, który kopiuje wszystkie pliki ze wskazanego katalogu lub katalogów do bieżącego katalogu. Nazwy katalogów są przekazywane do skryptu jako parametry:

```
./skrypt kat1 kat2 kat3
```

gdzie `kat1`, `kat2` itd. to pełne ścieżki dostępu do katalogów (np. `/x/y/z` lub `/usr/local`). Skrypt przyjmuje dowolną liczbę argumentów (ale powiedzmy, że nie więcej niż kilka, żeby zbytnio nie komplikować). Kopiowane pliki otrzymują nową nazwę. Początek nazwy informuje o katalogu, z którego pochodzi plik, np. plik `/x/y/z/plik1` otrzymuje nazwę `x_y_z_plik1`.

```
for i in $#
do
  for j in $i/*
  do
    nowanazwa=`echo $j | sed 's/\//_/g'
    cp $j ./$nowanazwa
  done
done
```

83. Dany jest plik tekstowy zawierający znaki drukowalne ASCII.

(a) Drukuj listę wszystkich znaków pojawiających się w pliku. Wielkie i małe litery są rozróżniane, tzn. `A` i `a` są różnymi znakami. Wskazówka: można np. zdefiniować separator pól tak, by był pustym ciągiem znaków. Wówczas każdy znak będzie stanowił jedno pole.

```
# skrypt.awk
BEGIN{ FS="" }
{ for (i=1; i<=NF; i++)
  {
    print $i
  }
}
```

Uruchomienie:

```
awk -f skrypt.awk plik | sort | uniq
```

(b) Drukuj 5 najczęściej występujących znaków wraz z danymi o częstotliwości ich występowania (w skali od 0 do 1).

```
# skrypt1.awk
BEGIN { FS = "" }
FILENAME == "plikznakow" { i++; znaki[i] = $0 }
FILENAME == " plik" { for (i=1; i<=NF; i++)
  {
    suma[$i]++
  }
}
END { for (i=1; i<=z; i++)
      {
        znak = znaki[i]
        printf("10%d %s\n", suma[znak], znak)
      }
}
```

Uruchomienie:

```
awk -f skrypt.awk plik | sort | uniq > plikznakow
awk -f skrypt1.awk plikznakow plik | sort -nr | head -5
```

(c) Drukuj dane o najdłuższym ciągu znaków powtarzających się w jednym wierszu, np. `AAAAA` (załóżmy dla uproszczenia, że jest tylko jeden taki wiersz). Nie wiemy z góry, jaka jest długość szukanego ciągu. Informacja ma mieć następującą postać:

nr wiersza, liczba znaków w ciągu, wiersz

(d) Spróbuj uwzględnić przypadek, kiedy w skrypcie może być kilka wierszy zawierających najdłuższy ciąg powtarzających się znaków.

84. Dany jest plik tekstowy. Napisz skrypt, który przyjmuje na wejściu dwa słowa.

(a) Skrypt ma drukować wszystkie wiersze pliku, w których pojawiają się oba słowa

```
# skrypt.sh
grep "$1" plik | grep "$2"
```

(b) Skrypt ma drukować wszystkie wiersze pliku, w których oba słowa są przedzielone co najwyżej dwoma innymi słowami.

```
BEGIN{ slowo1 = ARGV[1]; slowo2 = ARGV[2]
       ARGV[1] = ""; ARGV[2] = ""
}
$0 ~ slowo1 && $0 ~ slowo2 {
  for (i=1; i<=NF; i++)
  {
    if ($i == slowo1)
    { a = $i }
    if ($i == slowo2)
    { b = $i }
  }
  if (abs(a-b) <= 2)
  { print }
}
```

85. Dany jest plik z kodem źródłowym w języku C. Wydrukuj liczbę instrukcji `if`. Uwaga: ciąg *if* może być częścią nazwy funkcji lub zmiennej. Należy drukować tylko liczbę instrukcji `if`.

```
grep -c 'if' plik.c
```

86. Napisz skrypt, który doda uprawnienia użytkownikom kategorii *others*. Oto wymagania, jakie ma spełniać skrypt:

• ma przyjmować parametr będący nazwą (ścieżką) katalogu, w którym operacja ma być wykonywana,

• jeżeli użytkownik uruchomi skrypt bez parametru, czynności mają być wykonane w bieżącym katalogu,

• jeżeli właściciel ma uprawnienia do odczytu lub do odczytu i do zapisu, użytkownicy kategorii *others* mają mieć tylko uprawnienia do odczytu,

• jeżeli właściciel ma uprawnienia do uruchamiania pliku, użytkownicy kategorii *others* mają mieć tylko uprawnienia do wykonywania,

• uprawnienia użytkowników kategorii *group* mają pozostać bez zmian.

Uwaga: jeśli właściciel ma uprawnienia i do odczytu, i do wykonywania, użytkownik kategorii *others* ma mieć tylko uprawnienia do uruchamiania.

```
if [ $# -eq 0 ]
then
  kat="."
else
  kat=$1
fi

for i in $kat/*
do
  if [ `ls -l $i | grep '^.r[w-]'` ]
  then
    chmod o=r $i
  fi
  if [ `ls -l $i | grep '^...x'` ]
  then
    chmod o=x $i
  fi
done
```

87. Inna wersja poprzedniego zadania. Skrypt ma być uruchamiany z trzema parametrami. Pierwszym parametrem jest nazwa katalogu (ścieżka), drugim – ciąg znaków odpowiadający uprawnieniom właściciela, trzecim – ciąg znaków odpowiadający uprawnieniom użytkowników *others*.

```
for i in $1/*
do
  if [ `ls -l $i | grep '^$2'` ]
  then
    chmod o=$3 $i
  fi
done
```

88. Dany jest plik tekstowy zawierający pewien tekst literacki.

(a) Oblicz średnią długość jednego wyrazu w tym tekście. Najpierw należy usunąć z tekstu następujące znaki: `! ? . : ;` Ponadto, znak `-` może pojawić się jako pauza (np. `slowo1 - slowo2`). W takim przypadku go usuwamy. Znak `-` może także pojawić się jako łącznik spinający dwa słowa (np. *słodko-kwaśny*). W takim przypadku usuwamy cały ten złożony wyraz.

```
sed 's/[\!\?\.:;]//g; s/[ ][ ]*-//g; s/-[ ][ ]*//g' \
  plik | sed 's/[A-Za-z]+-[A-Za-z]+//g' |\
  awk -f skrypt.awk
```

gdzie `skrypt.awk` ma następującą postać:

```
{ for (i=1; i<=NF; i++)
  {
    liczbaslow++
    liczbaliter = liczbaliter + length($i)
  }
}
END { print liczbaliter/liczbaslow }
```

(b) Drukuj wiersz zawierający najdłuższe słowo. Wynik powinien mieć następującą postać:

numer_wiersza długość_słowa zawartość_wiersza

Zakładając, że tylko jedno słowo ma maksymalną długość:

```
{ for (i=1; i<=NF; i++)
  {
    if (length($i) > max)
    { max = length($i) }
  }
}
END { print max }

# Uruchomienie skryptu: ./skrypt.sh nazwapliku
max=`awk -f skrypt.awk $1`
echo "Najdluzsze slowo ma $max znakow"
awk -f skrypt1.awk $max $1
```

gdzie `skrypt1.awk` ma następującą postać:

```
BEGIN { max=ARGV[1]; ARGV[1]="" }
{ for (i=1; i<=NF; i++)
  {
    if (length($i) == max) { print $0 }
  }
}
```

Jednak wierszy zawierających słowa największej długości może być wiele. Oto skrypt, który działa poprawnie również w takim przypadku:

```
{ for (i=1; i<=NF; i++)
  maxw = 0
  {
    if (length($i) > maxw)
    { maxw = length($i) }
  }
  maxwwierszu[NR] = maxw
  if (maxw > max)
  { max = maxw }
}
END { for (i=1; i<=NR; i++)
      {
        if (maxwwierszu[i] == max)
        { print $0 }
      }
}
```

(c) Drukuj liczbę słów jednoliterowych

```
{ for (i=1; i<=NF; i++)
  {
    if (length($i) == 1)
    { suma++ }
  }
}
END { print suma }
```

89. Kody pocztowe. Sprawdzenie poprawności danych. Kod pocztowy może być podany w postaci 12345 lub 12-345. Do skryptu kod jest przekazywany jako parametr w poleceniu uruchamiającym skrypt. Napisz w skrypcie instrukcję lub instrukcje sprawdzające

    (a) czy liczba cyfr w przekazanym parametrze jest różna od 5

    (b) czy przekazany parametr zawiera przynajmniej jeden znak inny niż cyfra lub -

    (c) czy znak - (jeśli występuje) pojawia się w innym miejscu niż między 2. a 3. cyfrą. Jeśli któryś z warunków (a), (b) lub (c) jest spełniony, drukować krótki komunikat o błędzie.

```
if [ ! `echo $1 |\
  grep '^[0-9][0-9][-]?[0-9][0-9][0-9]$'` ]
then
  echo "blad"
else
  if [ `echo $1 | grep '[^0-9-]'` ]
  then
    echo "blad"
  fi
fi
```

90. Napisz skrypt, który w bieżącym katalogu zmienia nazwy wszystkich plików postaci plik.htm na plik.html.

```
for i in ./*.htm
do
  mv $i ${i}l
done
```

91. Napisz skrypt wyodrębniający adresy internetowe (ale tylko nazwy domen, bez katalogów i nazw plików) z pliku tekstowego dołączonego poniżej. Interesują nas tylko adresy niezwiązane z witryną Google. W pliku wynikowym żaden adres nie może pojawić się więcej niż jeden raz.

```
www.bigwebmaster.com/1850.html - 39k
 - [17]Cached - [18]Similar pages

www.bigwebmaster.com/60.html - 40k
 - [20]Cached - [21]Similar pages

www-psych.stanford.edu/~gruffydd/290/webscripts.html
 - 9k - [24]Cached - [25]Similar pages

wirerimmed.com/ index.php?section=article&album_id=23
 - 14k - 9 Mar
2004 - [27]Cached - [28]Similar pages

www.ai.mit.edu/~szilagyi/sh.html - 14k
 - [30]Cached - [31]Similar pages

www.sys-con.com/ java/readmessage.cfm?id=1945&article=2962
 - 21k - [33]Cached - [34]Similar pages

sed.sourceforge.net/grabbag/scripts/testo.htm
 - 32k - [36]Cached - [37]Similar pages

www.johnmckinnon.com/
 docs/scripts/perform_ftp_distribution_a_ksh.html
- 6k - [39]Cached - [40]Similar pages

www.johnmckinnon.com/docs/scripts/
 perform_step_1_ksh.html - 7k -
[42]Cached - [43]Similar pages

www.intuitive.com/wicked/ - 13k
 - 8 Mar 2004 - [45]Cached - [46]Similar pages

   1. http://www.google.com/webhp?hl=en
   2. http://www.google.com/advanced_search?
   3. http://www.google.com/preferences?
  13. http://www.google.com/url?sa=X&oi=news&
start=0&num=1&q=http://books.slashdot.org/books/
04/03/02/0036227.shtml
  18. http://www.google.com/search?hl=en&lr=&
ie=UTF-8&q=related:
www.bigwebmaster.com/1850.html
  19. http://www.bigwebmaster.com/60.html
  20. http://66.102.9.104/search?q=cache:1w2RiGuKKwAJ:
www.bigwebmaster.com/60.html
  44. http://www.intuitive.com/wicked/
```

92. Usuń człon `www.` pojawiający się na początku niektórych adresów.

```
sed 's/^www\.//' plik
```

93. Odwróć kolejność pól w każdym wierszu.

```
{ for (i=NF; i>=1; i--)
  { printf("%s ",$i) }
  printf("\n")
}
```

94. Otrzymany plik posortuj kolejno względem każdego pola. Zrobienie tego od razu w pełnej ogólności może sprawić trudności. Na początek można to zrobić dla dwóch pierwszych pól. Jak to zrobić? Najpierw należy sformatować wydruk tak, by kolejne pola były wyrównane do lewej strony, np.

```
2   33        14       212
70  104 97  217
104 9 102  66
pl  zielonasiec biuletyn
...
com yahoo groups
```

    Następnie posortuj plik alfabetycznie od lewej do prawej. Dzięki temu podczas sortowania obecność spacji nie przeszkodzi w uzyskaniu kolejności alfabetycznej. Nie używaj znaku tabulacji.

95. Następnie powtórz te same czynności dla tego samego pliku początkowego, ale z danymi o właścicielu witryny i z komentarzem. Wynik końcowy, po posortowaniu powinien zawierać także dane o właścicielu i komentarz. Ponieważ często komentarz mógłby składać się z wielu wierszy, lepiej chyba zacząć od pliku o następującej strukturze:

    adres1

    właściciel witryny (osoba, firma, organizacja)

    opcjonalny komentarz; może pojawiać się lub nie, może składać się z wielu wierszy

adres2

właściciel

komentarz

itd. Oczywiście, w tym przypadku należy zmienić domyślne definicje separatorów pól i rekordów.

96. Dane o właścicielu to na ogół nie tylko nazwa firmy lub imię i nazwisko, ale także adres poczty elektronicznej, numery telefonów, faksu, zwykły adres pocztowy (ulica, nr domu, kod pocztowy, miejscowość) Jak to uwzględnić? W kolejnych wierszach pojawiałyby się kolejne elementy, na przykład:

    adres strony www

    nazwa lub imię i nazwisko

    adres(y) poczty elektronicznej (jeśli np. dwa adresy, mogą być w tym samym wierszu)

    numer(y) telefonów (jeśli więcej niż 1, mogą być w tym samym wierszu)

    numer faksu

    ulica

    nr domu i ew. mieszkania

    kod pocztowy w formacie xxyyy lub xx-yyy lub PL-..., DE-..., jeśli uwzględnić adresy w różnych krajach

    miejscowość

    województwo lub region

    komentarz (może zajmować wiele wierszy)

    Jeśli brak jakichś danych, pisać np. `brak`.

    Jeżeli się rozrasta i przybywa danych, można go podzielić według kraju, regionu, charakteru witryny itp.

97. Może warto mieć kopię każdej witryny na komputerze lokalnym? Napisz skrypt pobierający kopię strony głównej każdej witryny i umieszczający ją w odpowiednim katalogu, np. dla adresu `biuletyn.zielonasiec.pl` w katalogu `./pl/zielonasiec/biuletyn`. Do pobierania witryn można np. użyć przeglądarki lynx:

    ```
    lynx -dump biuletyn.zielonasiec.pl > plik
    ```

Adresy witryn internetowych znajdują się w pliku `adresy`:

```
212.14.33.2
217.97.104.70
66.102.9.104
80.55.165.30
biuletyn.zielonasiec.pl
dziennik.bialystok.uw.gov.pl
edycja.bazafirm.pl
eksplor.linia.pl
fario.fm.interia.pl
free.ngo.pl
free.of.pl
gala.onet.pl
gotowce.com.pl
groups.yahoo.com
main.amu.edu.pl
miasta.gazeta.pl
polambeirut.com
slimak.sciaga.pl
suwalszczyzna.com.pl
tbdwitryna.w.interia.pl
www.agroturystyka.pl
www.alt.cgm.pl
www.avestom.republika.pl
www.bip.zelechlinek.pl
www.bird.pl
www.birding.gt.pl
www.bocian.org.pl
www.cidernet.pl
www.eximus.14lo.lublin.pl
www.expeditions.pl
www.gajdaw.pl
www.gizycko.um.gov.pl
www.gla.gfo.pl
www.huuskaluta.com.pl
www.iucn-ce.org.pl
www.kielce.pios.gov.pl
www.kki.net.pl
www.kreator.pnet.pl
```

```
www.kujawsko-pomorskie.pl
www.kurierbytowski.com.pl
www.lo1.sandomierz.ids.pl
www.montes-tarn-gory.pn.pl
www.mors.opus.chelm.pl
www.mos.gov.pl
www.mto.most.org.pl
www.opole.pl
www.pomorze.mao.pl
www.powiatlublin.lubelskie.pl
www.przyrodapolska.pl
www.republika.pl
www.rzeszow.org.pl
www.salamandra.org.pl
www.slaskie-abc.com.pl
www.sni.edu.pl
www.techweb.pl
www.tomaszow.iturysta.pl
www.turysta.net.pl
www.wigry.win.pl
www.zb.eco.pl
www.zc.invar.net.pl
www.zoo.torun.pl
```

Skrypt:

```
n=`cat adresy | wc -l`
i=0
while [ $i -lt $n ]
do
  i=$(($i+1))
  adreswww=`head -$i adresy | tail -1`
  plik=`echo $adreswww | sed 's/\./\//g'`
  echo $plik
  lynx -dump adreswww > plik
done
```

98. Wydrukuj wiersze pliku tekstowego składające się wyłącznie z cyfr i spacji, ale tylko te, które są poprzedzone wierszami zawierającymi wyłącznie litery i spacje.

```
{ biezacy = $0
  if (poprzedni !~ /A-Za-z /)
  {
    a=1
  }
  else
  {
    if ($0 !~ /0-9 /)
    {
      a=1
    }
    else
    {
      print
    }
  poprzedni = biezacy
}
```

99. Plik zawiera adresy poczty elektronicznej, po jednym adresie w wierszu. Napisz skrypt rozsyłający wiadomość zapisaną w pliku tekstowym do wszystkich odbiorców z tej listy. Użyj polecenia `mail adres < plik`

```
n=`cat adresy | wc -l`
i=0
while [ $i -lt $n ]
do
  i=$(($i+1))
  a=`head -$i adresy | tail -1`
  mail $a < plik
done
```

100. (a) Usuń z bieżącego katalogu wszystkie pliki o rozmiarze 0. Usuń wszystkie puste pliki również ze wszystkich podkatalogów bieżącego katalogu.

```
for i in *
do
  if [ ! -s $i ]
  then
    rm $i
  fi
  if [ -d $i ]
  then
    for j in $i/*
    do
      if [ ! -s $j ]
      then
        rm $j
      fi
    done
  fi
done
```

(b) Przenieś do innego katalogu wszystkie pliki znajdujące się w bieżącym katalogu, których rozmiar nie przekracza 1000 bajtów.

```
if [ ! -d malepliki ]
then
  mkdir malepliki
fi
for i in *
do
  if [ -f $i ]
  then
    rozmiar=`ls -l $i | awk '{ print $5 }'`
    if [ $rozmiar -le 1000 ]
    then
      cp $i ./malepliki
    fi
  fi
done
```

101. Oto przykład wyniku programu `top`. Drukuj dane o średnim obciążeniu procesora w ciągu ostatniej minuty oraz ilości zużytej pamieci.

```
01:38:17 up  2:13,  0 users,  load average: 0.05, 0.08, 0.21
2 processes: 1 sleeping, 1 running, 0 zombie, 0 stopped
CPU states:  4.3% user,  4.1% system,  0.0% nice, 91.6% idle
Mem:  523760K total, 154652K used, 369108K free, 0K buffers
Swap: 786432K total,  17288K used, 769144K free, 0K cached
PID USER   PRI NI SIZE  RSS SHARE STAT %CPU %MEM TIME COMMAND
1612 tktos   8  0 1220 2428    56 R     0.9  0.4 0:00 top
1276 tktos   8  0 1300 3024   108 S     0.0  0.5 0:00 bash
```

```
NR == 1 { print "srednie obciazenie CPU\
                 w ciagu 1 min.:", $8 }
NR == 4 { print "rozmiar zuzytej pamieci:", $4 }
```

```
$ top -n -1 | awk -f skrypt.awk | sed 's/,//'
```

102. Pliki w katalogu bieżącym zawierają dane tekstowe.

(a) Drukuj dane o plikach w następującej postaci: nazwa, liczba wystąpień poszukiwanego znaku, rozmiar pliku w bajtach, częstotliwość pojawiania się poszukiwanego znaku (w skali od 0 do 1, gdzie 1 oznaczałoby, że plik nie ma innych znaków prócz wyszukiwanego)

```
# skrypt.sh
for i in *
do
  rozmiar=`ls -l $i | awk '{ print $5 }' `
  awk -f skrypt.awk "$i" "$1" "$rozmiar"
done
```

gdzie `skrypt.awk` ma następującą postać:

```
BEGIN{ znak=ARGV[2]; rozmiar=ARGV[3]
       ARGV[2] = ""; ARGV[3] = ""
       suma = 0 }
{ for (i=1; i<=length($0); i++)
  {
    if (substr($0,i,1) == znak)
    {
```

```
      suma++
    }
  }
}
END{
  print FILENAME, suma, rozmiar, suma/rozmiar
}
```

(b) Drukuj posortowane dane o 10 plikach z największą liczbą wystąpień poszukiwanego znaku

```
for i in *
do
  rozmiar=`ls -l $i | awk '{ print $5 }' `
  awk -f skrypt.awk "$i" "$1" "$rozmiar"
done
```

```
# skrypt.awk
BEGIN{ znak=ARGV[2]; rozmiar=ARGV[3]
       ARGV[2] = ""; ARGV[3] = ""
       suma = 0 }
{ for (i=1; i<=length($0); i++)
  {
    if (substr($0,i,1) == znak)
    {
      suma++
    }
  }
}
END{
  printf("%10g %s %g %g\n", suma/rozmiar,\
          FILENAME, suma, rozmiar)
}
```

(c) Drukuj posortowane dane o 10 plikach z największą częstotliwością wystąpień znaku.

```
./skrypt.sh a | sort -nr | head
```

103. W katalogu bieżącym są dwa podkatalogi A i B. W każdym z nich znajdują się pliki tekstowe.

(a) Przeszukaj wszystkie pliki i zapisz w nowym pliku wszystkie ciągi znaków rozpoczynające się wielką literą (przyjmijmy dla uproszczenia, że wielka litera w interesujących nas ciągach znaków znajduje się na początku wiersza lub jest poprzedzona spacją). Drukuj każdy taki ciąg znaków i nazwę pliku w postaci `./A/plik` lub `./B/plik`, w którym się znajduje.

```
for kat in A B
do
  for i in ./$kat/*
  do
    echo $i
    awk 'BEGIN{ RS=" "}{ print $0 }' $i | egrep '^[A-Z]'
  done
done
```

(b) Przetwórz otrzymany plik tak, by drukować każde znalezione słowo tylko raz (tym razem pomijamy nazwę pliku, z którego pochodzą słowa).

```
for kat in A B
do
  for i in ./$kat/*
  do
    awk 'BEGIN{ RS=" "}{ print $0 }' $i | egrep '^[A-Z]'
  done
done

./10-b.sh | sort | uniq
```

104. (a) W wyniku pomyłki, nazwy niektórych plików z bieżącego katalogu zawierających strony www mają końcówki `.html.html`. Usuń w takich przypadkach ostatni człon `.html`. Nazwy z jednym członem `html` powinny pozostać bez zmian.

```
for i in *.html.html
do
  nowy=`echo $i | sed 's/\.html$//'`
  mv $i $nowy
done
```

(b) W nazwach niektórych plików bez końcówki `.html` trzeba dopisać końcówkę `.html`. Dotyczy to plików zawierających ciąg znaków `href` lub `HREF`.

```
for i in *
do
  a=""
  a=`grep '[Hh][Rr][Ee][Ff]' $i`
  b=`echo $i | grep '.html'`
  if [ -n "$a" -a -z "$b" ]
  then
    mv $i $i.html
  fi
done
```

105. Dany jest zwykły plik tekstowy.

(a) Zastąp ciągi wielu (2 lub większej liczby) spacji pojedynczą spacją. Usuń spacje na początku i na końcu wiersza.

```
sed 's/^[ ]*//; s/[ ]*$//; s/[ ][ ]*/ /g' plik
```

(b) Oblicz, ile jest słów składających się z dokładnie dwóch członów połączonych łącznikiem `-`.

```
BEGIN{ FS=";;" }
{ for (i=1; i<=NF; i++)
  {
    if ($i ~ /-/ && $i !~ /-.*-/) { suma++ }
  }
}
END{ print suma }
```

106. Oto fragment pliku dziennika serwera www Apache.

```
150.254.110.14 - - [19/Jan/2004:10:27:21 +0000]
 "GET /wyklad/html.html HTTP/1.1" 200 8374
150.254.110.14 - - [19/Jan/2004:10:27:32 +0000]
 "GET /wyklad/4.html HTTP/1.1" 200 3842
150.254.110.14 - - [19/Jan/2004:10:27:32 +0000]
```

```
 "GET /wyklad/2.html HTTP/1.1" 200 3842
193.219.28.2 - - [19/Jan/2004:10:28:42 +0000]
 "GET /zadania/1.txt HTTP/1.0" 304 -
193.219.28.2 - - [19/Jan/2004:10:28:57 +0000]
 "GET /zadania/index.html HTTP/1.0" 200 7534
193.219.28.2 - - [19/Jan/2004:10:29:28 +0000]
 "GET /zadania/index.html HTTP/1.0" 304 -
```

(a) Zapisz do osobnego pliku wszystkie adresy IP, z których łączono się z serwerem. Każdy adres może pojawić się w tym pliku tylko raz.

Oto dwa przykłady rozwiązań.

```
awk '{ print $1 } | sort | uniq > adresy_ip

sed 's/ -.*$//' | sort | uniq > adresy_ip
```

(b) Wykorzystując polecenie `nslookup`, znajdź nazwy domen odpowiadające wszystkim adresom IP zapisanym w pliku. Oto przykład polecenia `nslookup`:

```
$ nslookup 150.254.110.14
Server:  rose.man.poznan.pl
Address:  150.254.173.3

Name:    hoth.amu.edu.pl
Address:  150.254.110.14

$ nslookup < adresy_ip | egrep '^Name:' > plik

# s.awk
{ system("nslookup " $0) }

$ awk -f s.awk adresy_ip |\
  sed -n 's/Name:[ ]*//p' > plik

n=`cat adresy_ip | wc -l`
i=0
while [ $i -lt $n ]
do
```

```
  i=$(($i+1))
  ip=`head -$i adresy_ip | tail -1`
  nslookup $ip | awk '/^Name:/ { print $2 }'
done
```

107. Plik tekstowy zawiera m.in. adresy stron internetowych. Zakładamy, że adres zawsze rozpoczyna się ciągiem `http://`.

(a) Drukuj wszystkie adresy, przy założeniu, że są one oddzielone spacjami od pozostałego tekstu, a w wierszu znajduje się maksymalnie jeden adres.

```
sed -n 's/^.*http:\/\//http:\/\//p' plik |\
sed -n 's/ .*$//p'
```

(b) Drukuj wszystkie adresy zakładając, że w wierszu może znajdować się wiele adresów oddzielonych spacjami od pozostałego tekstu.

```
{ for (i=1; i<=NF; i++)
  {
    if ($i ~ /^http:\/\//) print $i
  }
}
```

108. Napisz skrypt sprawdzający co 5 minut liczbę użytkowników zalogowanych w systemie. Drukuj średnią liczbę użytkowników z ostatniej godziny.

```
i=0
while :
do
  i=$(($i+1))
  n=`who | awk '{ print $1 }' | sort | uniq`
  if [ $i -gt 12 ]
  then
    suma=$(($suma+$n-$n1))
    n13=$n
    for i in 1 2 3 4 5 6 7 8 9 10 11 12
    do
```

```
      n`$i`=$(($n+1))
    done
  else
    if [ $i -eq 12 ]
    then
      n12=$n
    fi
  fi
  echo $(($n/12))
sleep 300
done
```

109. W pliku zawarte są dane o produkcie krajowym brutto (PKB) Polski w formacie

```
Rok PKB
```

gdzie PKB podawany jest w mld dolarów (uwaga: przedstawione liczby nie są danymi statystycznymi; są jedynie ilustracją postaci pliku danych):

```
1990  120
1991  110
1992  120
1993  130
1994  132
1995  150
1996  165
1997  185
1998  202
1999  210
2000  212
2001  214
2002  218
2003  225
2004  238
```

(a) Napisz skrypt, który zmieni postać pliku tak, by oprócz powyższych danych w pliku znajdowała się trzecia kolumna zawierająca przyrost PKB w porównaniu z rokiem poprzednim, wyrażony w mld USD, oraz czwarta kolumna zawierająca względny przyrost PKB (ułamek wartości z poprzedniego roku).

```
NR == 1 { pkb[$1]=$2
          print $1, $2, "nie dotyczy", "nie dotyczy"
}
NR > 1 { pkb[$1]=$2
         print $1, $2, pkb[$1-1],\
         (pkb[$1]-pkb[$1-1])/pkb[$1-1]
}
```

(b) Załóżmy, że podkatalog `kraje` katalogu bieżącego zawiera wiele plików w tym formacie. Jeden plik zawiera dane dotyczące jednego kraju, np. plik `DE` zawiera dane dotyczące Niemiec, `GB` – Wielkiej Brytanii itd. Napisz skrypt wykonujący czynności opisane w punkcie (a) dla każdego pliku z katalogu `kraje`.

```
for i in ./kraje/*
do
  awk -f skrypt.awk $i > $i.1
done
```

(c) Utwórz skrypt przyjmujący rok jako parametr, na przykład `./skrypt.sh 2002`. Skrypt ma drukować dane o przyroście PKB w formacie (dwa pola):

```
Kraj ulamkowy przyrost PKB
```

dla pięciu krajów, których przyrost PKB w danym roku był największy.

Przyrost bezwzględny:

```
rok=$1
for i in ./kraje/*
do
  a=`grep "^$rok" $i | awk '{ print $3 }'`
  echo "$a $i" | sort -nr | head -5
done
```

Przyrost względny:

```
rok=$1
for i in ./kraje/*
do
  a=`grep "^$rok" $i | awk '{ print $4 }'`
  echo "$a $i" | sort -nr | head -5
done
```

(d) Utwórz skrypt drukujący dane dotyczące kraju z największym przyrostem ułamkowym (względnym) między dwoma wybranymi latami, np. `./skrypt.sh 1995 2002`

```
rok1=$1
rok2=$2
for i in ./kraje/*
do
  pkb2=`grep "^$rok1" $i | awk '{ print $2 }'`
  pkb1=`grep "^$rok2" $i | awk '{ print $2 }'`
  przyrost=$(($pkb2-$pkb1))
  przyrost-wzgledny=$(($przyrost/$pkb1))
  echo "$przyrost-wzgledny $i"
done
```

Powyższy skrypt uruchamiamy za pomoca innego skryptu:

```
./skrypt.sh $1 $2 | sort -nr | head -1
```

110. Napisz skrypt, który przyjmuje nazwę pliku jako argument i sprawdza, czy w pliku istnieją znaki inne niż litery (wielkie lub małe), cyfry, spacja, kropka, ukośnik i `-`. Drukuj liczbę wszystkich znaków w pliku i liczbę znaków innych niż litery.

111. Polecenie `last` drukuje dane o dacie i godzinie logowania użytkowników do systemu. Przyjmijmy, że w systemie przechowywane są dane z ostatnich 12 miesięcy.

```
kokosz    acacia.man.pozna Sat Oct 30 12:58 - 13:05  (00:06)
yeti      lab-227M-1.cs.pu Sat Oct 30 12:56 - 12:56  (00:00)
kokosz    acacia.man.pozna Sat Oct 30 12:31 - 12:55  (00:23)
pakulam   ph58.poznan.cvx. Sat Oct 30 12:01 - 12:03  (00:01)
kokosz    acacia.man.pozna Sat Oct 30 12:01 - 12:26  (00:25)
aczajka   wsnhid-gw.man.po Sat Oct 30 11:22 - 11:23  (00:00)
kokosz    acacia.man.pozna Sat Oct 30 11:20 - 11:52  (00:32)
psiwczak  azk128.neoplus.a Sat Oct 30 10:46 - 10:46  (00:00)
psiwczak  azk128.neoplus.a Sat Oct 30 10:46 - 10:46  (00:00)
cecko     poppy.man.poznan Sat Oct 30 10:12   still logged in
niwinska  c55-36.icpnet.pl Sat Oct 30 10:12 - 10:12  (00:00)
budsoft   bbg187.neoplus.a Sat Oct 30 10:07 - 10:07  (00:00)
stachoo   c24-16.icpnet.pl Sat Oct 30 09:46   still logged in
mkk       bbl195.neoplus.a Sat Oct 30 09:41 - 09:50  (00:08)
zupa      sage-39.man.pozn Sat Oct 30 09:41 - 09:42  (00:01)
```

```
...
naber     fikus.man.poznan Fri Oct 29 13:42 - 16:51  (03:08)
beny      150.254.17.41    Fri Oct 29 13:42 - 13:43  (00:00)
ariel     beech-bis.man.po Fri Oct 29 13:42 - 14:56  (01:14)
holy      aif126.internetd Fri Oct 29 13:41 - 13:42  (00:00)
magdaniw  c9-iir-058.au.po Fri Oct 29 13:18 - 13:19  (00:00)
kontakt   fresia.man.pozna Fri Oct 29 13:16 - 15:09  (01:52)
zgdan     nmr.chemd.amu.ed Fri Oct 29 13:15 - 13:21  (00:06)
mzawadzk  hawthorn.man.poz Fri Oct 29 12:56 - 17:33  (04:37)
...
```

(a) Dla każdego miesiąca utwórz listę użytkowników, którzy logowali się w tym czasie. Dane z każdego miesiąca zapisz w osobnym pliku.

```
last | awk '{ print >> $5 }'
```

(b) Dla każdego z miesięcy utwórz listę użytkowników, którzy ani razu się nie zalogowali.

Załóżmy, że plik `nazwy-uzytkownikow` zawiera nazwy wszystkich użytkowników, po jednej nazwie w wierszu.

```
n=`wc -l nazwy-uzytkownikow | awk '{ print $1 }'`
for m in Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
do
  i=0
  while [ $i -lt $n ]
  do
    i=$(($i+1))
    u=`head -$i nazwy-uzytkownikow | tail -1`
    s=`grep "$u" $m`
    if [ -z "$s" ]
    then
      echo $u >> $m.nieobecni
    fi
  done
done
```

(c) Dla każdego miesiąca drukuj listę 10 użytkowników o największej liczbie logowań.

```
n=`wc -l nazwy-uzytkownikow`
for m in Jan Feb Mar Apr May Jun\
```

```
         Jul Aug Sep Oct Nov Dec
do
  uzytk=""
  max=0
  i=0
  while [ $i -lt $n ]
  do
    i=$(($i+1))
    u=`head -$i nazwy-uzytkownikow | tail -1`
    k=`grep -c "$u"`
    if [ $k -gt $max ]
    then
      uzytk=$u
      max=$k
    else
      if [ $k -eq $max ]
      then
        uzytk=$uzytk" "$u
      fi
    fi
  done
# Drukuje nazwe miesiaca, maksymalna liczbe wejsc
# i nazwe najczesciej logujacego sie uzytkownika
# lub liste nazw kilku uzytkownikow
# (jezeli jest kilku uzytkownikow z maksymalna
# liczba wejsc danego miesiaca)
  echo "$m $max $uzytk"
done
```

(d) Drukuj średnią liczbę logowań przypadającą na użytkownika w każdym miesiącu. Pomiń użytkowników, którzy nie logowali się ani razu.

```
BEGIN{ log=ARGV[1]; ARGV[1]="" }
END{ print log/12 }

n=`wc -l nazwy-uzytkownikow`
i=0
while [ $i -lt $n ]
do
  i=$(($i+1))
```

```
  u=`head -$i nazwy-uzytkownikow | tail -1`
  for m in Jan Feb Mar Apr May Jun\
           Jul Aug Sep Oct Nov Dec
  do
    k=`grep -c "$u" $m`
    log=$(($log+$k))
  done
  echo "$u `awk -f skrypt.awk $log 1wiersz `
done
```

(e) Dla każdego użytkownika osobno oblicz średni czas trwania jednej sesji (w minutach). Można w tym celu wykorzystać np. skrypt, który co 60 sekund sprawdza, kto jest zalogowany.

(f) Drukuj średni czas trwania jednej sesji w każdym z 12 miesięcy.

(g) Zapisz w oddzielnym pliku dane o sesjach, które już się zakończyły.

```
grep 'n$' $1 > plik-sesje-zakonczone
```

(h) Zapisz w oddzielnych plikach dane o sesjach z poszczególnych dni. Na przykład, dane dotyczące sesji rozpoczętych w sobotę 30 października, powinny znaleźć się w pliku `Sat.Oct.30`, dane sesji rozpoczętych w piątek, 29 października – w pliku `Fri.Oct.29` itd.

```
awk '{ print >> $4"."$5"."$6 }' plik
```

(i) Napisz skrypt drukujący dane o sesjach użytkownika, którego nazwa jest przekazywana do skryptu jako parametr: `./skrypt.sh nazwa`

```
grep "^$1" plik
```

(j) Jeżeli sesja trwa dłużej niż 24 godziny, czas trwania sesji, drukowany w ostatnim polu, ma postać `n+gg:mm`, gdzie `n` – liczba dni, `gg` – liczba godzin, `mm` – liczba minut. Oto przykład:

```
bla     carlina.man.pozn Thu Oct 28 14:27 - 15:57 (1+01:29)
grucha  willow.man.pozna Thu Oct 28 13:34 - 20:03 (1+06:29)
```

(k) Zapisz dane o sesjach trwających ponad 24 godziny do oddzielnego pliku.

```
egrep "\+..:..\)$" plik
```

(l) Drukuj listę użytkowników, którzy logowali się do systemu w ciągu ostatniego tygodnia. Nazwa każdego użytkownika powinna pojawiać się tylko raz. Wynik powinien być posortowany alfabetycznie.

```
i=0
while [ $i - lt 8 ]
do
  dzien=`date -d "$i days ago" |\
         awk '{ print $1, $2, $3 }'`
  uzytkownik=`egrep "$dzien" plik | awk '{ print $1 }'`
  i=$[$i+1]
done
```

(m) Drukuj nazwy 10 użytkowników, którzy otworzyli najwięcej sesji.

Skrypt w awk,

```
{
  # tablica s zawiera liczbe sesji kazdego uzytkownika
  s[$1]++
  # jezeli nazwa uzytkownika pojawia sie
  # po raz pierwszy, dopisz go do listy uzytkownikow
  if (s[$1] == 1)
  {
    m++
    uzytkownik[m]=$1
  }
}
END{
  for (i=1; i<=m; i++)
  {
    print s[uzytkownik[i]], uzytkownik[i]
  }
}
```

uruchamiany poleceniem `awk -f skrypt.awk plik |sort -nr`

(n) Histogram wejść do systemu w poszczególnych godzinach w ciągu doby, od godziny 00 do 23. Dla wszystkich przedziałów 1-godzinnych drukuj łączną liczbę wejść wszystkich użytkowników z całego tygodnia.

```
i=0
while [ $i -le 23 ]
do
  if [ $i -lt 10 ]
  then
    godz=0$i
  else
    godz=$i
  fi
  echo $godz `awk '{ print $7 }' plik |\
              grep -c "^$godz"`
  i=$[$i+1]
done
```

(o) Drukuj liczbę sesji otwieranych w poszczególne dni tygodnia, od poniedziałku do niedzieli.

```
for i in Mon Tue Wed Thu Fri Sat Sun
do
  echo "$i" `awk '{ print $4 }' plik | grep -c "$i"`
done
```

112. Napisz skrypt usuwający znaczniki <...> z pliku html.

```
./skrypt.awk plik.html | sed '/^[ ]*$/s'
```

gdzie `skrypt.awk` ma postać

```
BEGIN { RS="\<[^>]*\>" }
{ print }
```

Filtrowanie wyniku za pomocą sed służy usunięciu na wyjściu wierszy pustych lub wypełnionych tylko spacjami. Można ten element pominąć.

113. Napisz skrypt wczytujący dwa pliki tekstowe i drukujący na wyjściu obok siebie wiersze obu plików. Na przykład, jeżeli w pierwszym pliku mamy

```
cos1
cos2
...
```

a w pliku2 mamy

```
abc1
abc2
...
```

na wyjściu mamy otrzymać

```
cos1 abc1
cos2 abc2
...
```

Najprościej można to zrobić za pomocą instrukcji `paste`:

```
paste plik1 plik2
```

Wada tego rozwiązania polega na tym, że nie zachowano formatu plików. Na wyjściu między wiersze z obu plików wstawiany jest znak tabulacji. Jeżeli chcemy rozdzielić te fragmenty plików w inny sposób, trzeba skorzystać z opcji `-d`:

```
paste -d ' ' plik1 plik2
```

W tym przypadku treść obu plików jest rozdzielona pojedynczą spacją. Separatorem fragmentów może być ciąg pusty:

```
paste -d '' plik1 plik2
```

Wówczas na wyjściu nie pojawiają się żadne dodatkowe znaki. Oto inne rozwiązanie, skrypt powłoki bash, w którym nie korzystamy z polecenia `paste`.

```
n=`wc -l plik1 | cut -c1-7`
# mozna tez tak:
# n=`wc -l plik1 | awk '{ print $1 }'`
i=1
while [ $i -le $n ]
do
  a=`head -$i plik1 | tail -1`
  b=`head -$i plik2 | tail -2`
  echo "$a$b"
  i=$(($i+1))
done
```

Wadą tego rozwiązania jest wielokrotne otwieranie plików – w każdym kroku pętli `while` użyto polecenia `head`, co oznacza za każdym razem otwarcie pliku w celu wyświetlenia określonej jego cześci.

W rozwiązaniu za pomocą skryptu awk wystarczy użyć dwóch zmiennych domyślnych: `FILENAME` i `FNR`. Pierwsza z nich oznacza nazwę bieżącego pliku danych, a druga – numer rekordu w bieżącym pliku.

```
FILENAME == "plik1" { a[FNR] = $0 }
FILENAME == "plik2" { b[FNR] = $0 }
END{ for (i=1; i<=NR; i++)
  {
# Sklejenie w taki sposob, aby nie bylo
# spacji miedzy odpowiednimi czesciami
# plikow w kazdym wierszu:
    print a[i]""b[i]
# Jesli potrzebne sa spacje:
#    print a[i], b[i]
# lub
#    print a[i]" "b[i]
  }
}
```

# 9. Zasoby internetowe

### Witryny internetowe

- `http://cnswww.cns.cwru.edu/~chet/bash/bashtop.html`, strona główna powłoki Bash
- http://cm.bell-labs.com/cm/cs/awkbook/, witryna książki *The AWK Programming Language*, której autorami są twórcy tego języka
- `http://www.tcl.tk`, Tcl Developer Xchange
- `http://www.activestate.com/Products/ActiveTcl`, dystrybucja interpretera i narzędzi języka Tcl przygotowana w firmie ActiveState
- `http://www.python.org`, witryna główna języka Python
- `http://www.activestate.com/ASPN/Python`, dystrybucja interpretera i narzędzi języka Python przygotowana w firmie ActiveState
- `http://www.pythonware.com`, oprogramowanie w języku Python

### Grupy dyskusyjne Usenet

- `comp.lang.awk`
- `comp.lang.python`
- `comp.lang.tcl`
- `comp.unix.shell`

# Literatura


A.V. Aho, B.W. Kernighan, P.J. Weinberger, *The AWK Programming Language*, Addison-Wesley, 1988

D.M. Beazley, *Programowanie: Python*, RM, 2002

L.S. Borkowski, *Unix i jego emulatory*, Wyd. Nauk. UAM, Poznań, 2004

D. Dougherty, A. Robbins, *sed i awk*, Helion, 2002

C. Flynt, *Tcl/Tk for Real Programmers*, Morgan Kaufmann, 1997

J.E.F. Friedl, *Wyrażenia regularne*, Helion, 2001

B.W. Kernighan, R. Pike, *The UNIX Programming Environment*, Prentice-Hall, 1984

M. Lutz, *Programming Python*, O'Reilly, 1996

A. Martelli, *Python in a Nutshell*, O'Reilly, 2003

C. Newham, B. Rosenblatt, *Learning the Bash Shell*, O'Reilly, 1998

J.K. Ousterhout, *Scripting: Higher level programming for the 21st Century*, IEEE Computer, 31(3), 23-30, 1998, `http://home.pacbell.net/ouster/scripting.html`

J.K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley, 1993

A. Robbins, *Effective Awk Programming*, Specialized System Consultants, 1997

D. Tansley, *Unix and Linux Shell Programming*, Addison-Wesley, 2000

B. Welch, *Practical Programming in Tcl and Tk*, Prentice-Hall, 1995