J.Nawrocki, M. Antczak, K. Kwarciak, G. Palik

# Ćwiczenie nr 6

# Wyrażenia regularne i język AWK

## Środowisko uruchomieniowe

1. Pobrać plik ***SimpleAWKforWindows.zip*** (*AWK95*) (`http://www.cs.put.poznan.pl/gpalik/wdi/SimpleAWKforWindows.zip`).
2. Rozpakować powyższe archiwum w katalogu roboczym (na zajęciach najprawdopodobniej C:\Temp).
3. Ważne pliki wykorzystywane podczas kompilacji:
   a. *awk95.exe* – wykonywalny plik reprezentujący interpreter programów napisanych w języku AWK,
   b. *in.txt* – plik tekstowy zawierający dane wejściowe (tekst, który powinien zostać przetworzony),
   c. *out.txt* – wyjściowy plik tekstowy zawierający rezultaty (zwrócony tekst będący wynikiem przetwarzania zapisanego w kodzie programu na danych znajdujących się w pliku wejściowym),
   d. *prog.awk* – kod źródłowy programu zapisany w języku AWK,
   e. *run.bat* – zawiera składnię polecenia uruchamiającego interpreter AWK programu zapisanego w pliku *prog.awk*, który będzie operował na danych z pliku wejściowego *in-.txt*, a wyniki przetwarzania będą zapisywane w pliku wyjściowym *out.txt*.

      ```
      awk95 -f kod_zrodlowy.awk <strumien_wejsciowy >strumien_wyjsciowy
      ```

      np.:

      ```
      awk95 -f prog.awk <in.txt >out.txt
      ```

## AWK w pigułce

*Składnia wywołania AWK:*

```
awk [-Fs] "program" [plik1 plik2...]      # komendy zapisywane są w linii poleceń
                                          # DOSa.
awk 'program{print "foo"}' plik1          # pojedyncze cudzysłowy otaczają argumenty
                                          # wywołania, które mogą zawierać podwójne
                                          # cudzysłowy.
    # Uwaga: Dopóki AWK akceptuje pojedyncze cudzysłowy znajdujące się wokół argumentów
    # podawanych z linii poleceń systemu operacyjnego, oznacza to, że ścieżki do plików, które
    # zawierają w sobie pojedyncze cudzysłowy nie są rozpoznawane przez AWK, pomimo nawet,
    # że są poprawnymi ścieżkami z punktu widzenia systemu operacyjnego. Aby AWK rozpoznał
    # plik foo'bar to podana nazwa musi być zapisana w następujący sposób foo'"'"'bar.

awk [-Fs] -f plik_źródłowy_programu [plik1 plik2...] # komendy zapisywane są w
                                                     # linii poleceń DOSa.
```

Jeżeli plik1 jest pominięty wtedy AWK zakłada, że analizowane dane są pobierane ze standardowego wejścia (konsola systemowa).
Argument wywołania oznaczony jako `-Fz` ustawia separator pól *FS* na znak *"z"*.

Plik źródłowy AWK składa się z następujących reguł przetwarzania:
*"wzorzec {instrukcje}"*

- Jeżeli *{instrukcje}* są pominięte, w regule przetwarzania, wtedy domyślnie jest wykonywane wypisanie wiersza na standardowe wyjście *{print $0}*.
- Jeżeli *"wzorzec"* jest pominięty w regule przetwarzania, wtedy każdy wiersz pliku wejściowego jest poddawany działaniu instrukcji zdefiniowanych w bloku *{instrukcje}*.

Pola są separowane najczęściej przez jedną lub więcej spacji lub tabulatorów: "pole1  pole2".

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

Jeżeli poniżej opisane przykłady poleceń znajdują się w pliku źródłowym, a nie są uruchamiane z linii poleceń, wtedy mogą zostać pominięte podwójne cudzysłowy.

*Podstawowe polecenia AWK:*

`"NR == 5" plik` wypisuje w rezultacie 5 wiersz (linia od góry) z pliku o nazwie `plik`.
**Uwaga**: *"=="* oznacza operator porównania.

`{FOO = 5}` przypisanie do zmiennej FOO wartości "5".
**Uwaga:** pojedyncze *"="* oznacza operator przypisania.

`"$2 == 0 {print $1}"` Jeżeli zawartość drugiego pola (pola są indeksowane od 1) jest równa 0 to w wyniku, na wyjściu, zostanie wypisana zawartość pola pierwszego.

`"$3 < 10"` Jeżeli numeryczna zawartość trzeciego pola jest mniejsza od 10, wtedy wiersz zawierający to pole zostaje wypisany na wyjście. (numeryczne porównanie).

`'$3 < "10"'` podczas porównywania łańcuchów wykorzystywane są pojedyncze cudzysłowy.

`-f pgmfile [$3 < "10"]` wykorzystywane jest polecenie *"-f pgmfile"* podczas porównywania łańcuchów.

`"$3 ~ /regexp/"` wypisanie wiersza, w którym trzecie pole spełnia opisane wyrażenie regularne */regexp/*.

`'$3 ~ "regexp"'` *regexp* może wystąpić w łańcuchu oznaczonym dwoma cudzysłowami. Podwójne cudzysłowy mogą zastępować *„backlashes”* w wyrażeniach regularnych. Wyrażenie zapisane z wykorzystaniem cudzysłowów wymagają istnienia znaku dopasowania (~).

`"NF > 4"` wypisanie wierszy, w których występuje 5 lub więcej pól.

`"$NF > 4"` wypisanie wierszy, w których w ostatnim polu znajduje się wartość, co najmniej 5.

`"{print NF}"` wypisuje na wyjściu ilość pól (wyrazów) znajduje się w każdym wierszu.

`"{print $NF}"` wypisuje ostatnie pole dla każdego przetwarzanego wiersza.

`"/regexp/"` wypisuje tylko te wiersze, które zawierają wyrażenie regularne *"regexp"*.

`"/text|file/"` wiersze zawierające *"text"* lub *"file"* (wielkość liter ma znaczenie!)

`"/foo/ {print "za", NR}"` błędnie zapisany argument zawierający wyrażenie regularne, który nie zadziała poprawnie podczas uruchomienia z linii poleceń!!

`'/foo/ {print "za", NR}'` poprawnie zapisany atrybut!! Jeżeli analizowany wiersz będzie zawierał *"foo"*, wtedy na wyjściu zostanie wypisane słowo *„za”* i numer obecnego wiersza.

`"$3 ~ /B/ {print $2,$3}"` Jeżeli trzecie pole zawiera *"B"*, wtedy na wyjście wypisywane jest drugie i trzecie pole.

`"$4 !~ /R/"` wypisuje wiersze, których czwarte pole nie zawiera *"R"*.

`'$1=$1'` Usuwa dodatkowe spacje pomiędzy polami i puste wiersze.

`'{$1=$1;print}'` Usuwa dodatkowe spacje pomiędzy polami, zostawiając puste wiersze.

`'NF'` Usuwa wszystkie puste wiersze.

*AND(&&), OR(||), NOT(!)*

`"$2 >= 4 || $3 <= 20"` wypisuje wiersze, w których zawartość drugiego pola jest większa bądź równa 4 lub zawartość trzeciego pola jest mniejsza bądź równa 20.

`"NR > 5 && /with/"` wypisuje wiersze, których numer jest, co najmniej równy 6 i które zawierają wzorzec *"with"*.

`"/x/ && NF > 2"` wypisuje wiersze, które zawierają więcej niż dwa pola i ich zawartość zawiera wzorzec *"x"*.

`"$3/$2 != 5"` wypisuje wiersze, dla których iloraz pola trzeciego przez drugie jest różny od 5. **Uwaga**: *„!=”* operator nierówności zarówno dla liczb jak i łańcuchów.

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

`"$3 !~ /regexp/"` wypisuje wiersze, które w trzecim polu nie zawierają wyrażenia regularnego *„regexp”*.

`"!($3 == 2 && $1 ~ /foo/)"` wypisuje wiersze, które nie spełniają warunku zdefiniowanego w nawiasie (zawartość pola trzeciego musi być równa 2 i zawartość pola pierwszego musi zawierać wzorzec *„foo”*).

`"{print NF, $1, $NF}"` dla każdego wiersza wypisywana jest ilość jego pól, zawartość pierwszego pola i zawartość ostatniego pola.

`"{print NR, $0}"` wypisuje każdy wiersz poprzedzony prefiksem, który reprezentuje numer wiersza.

`'{print NR ": " $0}'` wypisuje każdy wiersz poprzedzony prefiksem, który reprezentuje numer wiersza, dwukropek i spację.

`"NR == 10, NR == 20"` wypisuje wiersze reprezentowane numerami od 10-20 włącznie.

`"/start/, /stop/"` wypisuje zawartość każdego wiersza pomiędzy wzorcem *"start"* i *"stop"*.

`"length($0) > 72"` wypisuje wszystkie wiersze, których długość jest większa niż 72 znaki.

`"{print $2, $1}"` następuje na wyjściu odwrócenie dwóch pierwszych pól, wszystkie pozostałe pola są pomijane.

`"{print substr($0,index($0,$3))}"` wypisuje zawartość wiersza od trzeciego pola do końca linii.

*Wykorzystanie klauzuli END{...}*

Instrukcje w klauzuli END są uruchamiane po analizie wszystkich wierszy wejściowych. Jest wykorzystywana najczęściej do wypisywania końcowych statystyk.

1) END { print NR } # w rezultacie wypisywana jest ilość analizowanych wierszy.

2) `{s = s + $1 }` # wypisywana jest suma oraz średnia wszystkich liczb znajdujących się w polu
`END {print "sum is", s, "average is", s/NR}` # pierwszym dla każdego
# analizowanego wiersza.

3) `{names=names $1 " " }` # konwertuje całą zawartość pola 1 z wszystkich analizowanych
`END { print names }` # wierszy konkatenując je w jedną linię, np.:

```
            +      Beth   4.00   0     #
 plik       |      Mary   3.75   0     # plik wejściowy jest konkatenowany do:
wejściowy   |      Kathy  4.00   10    # "Beth Mary Kathy Mark" na wyjściu
            +      Mark   5.00   30    #
```

4) `{ field = $NF }` # wypisuje ostatnie pole ostatniego analizowanego wiersza.
`END { print field }`

*PRINT, PRINTF: wypisywanie wyrażeń, sposoby formatowania polecenia print*

`print(expr1, expr2, ..., exprn)` # mogą zostać wykorzystane wszystkie możliwe
# operatory: <, <=, ==, >, >=.

`print` # skrót dla instrukcji `{print $0}`.
`print ""` # wypisuje na wyjściu pusty wiersz.
`printf(expr1,expr2,expr3,\n}` # dodaj znak nowej linii do polecenia *„printf”*.
`print "expression" > "file name"` # rezultat wyrażenia może być zapisany do pliku
# wyjściowego.

*KONWERSJA FORMATU:*

```
BEGIN{ RS="";    FS="\n";      # oznaczamy wejściowy separator wiersza za pomocą pustej
   ORS="\n"; OFS="," }         # linii i separator pola za pomocą znaku nowej linii.
{$1=$1; print }                # następnie ustawiamy na wyjściu separator wiersza na znak
                               # nowej linii i separator pola na przecinek.
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

*PARAGRAFY:*

```
'BEGIN{RS="";ORS="\n\n"};/foo/'        # wypisuje każdy wiersz zawierający 'foo' w osobnym
                                       # paragrafie.
'BEGIN{RS="";ORS="\n\n"};/foo/&&/bar/'  # wymagane jest jednoczesne dopasowanie
                                       # obu wzorców
'BEGIN{RS="";ORS="\n\n"};/foo|bar/'        # wystarczy, że jeden ze wzorców
                                       zostanie
                                       # dopasowany.
```

*PRZEKAZYWANE ZMIENNE:*

```
gawk -v var="/regexp/" 'var{print "Here it is"}'      # zmienna "var" jest
                                                      # wyrażeniem regularnym
                                                      # "regexp", które może być
                                                      # wykorzystywane wewnątrz
                                                      # kodu programu AWK.
gawk -v var="regexp" '$0~var{print "Here it is"}'     # zmienna "var" jest
                                                      # łańcuchem otoczonym
                                                      # podwójnymi cudzysłowami.
gawk -v num=50 '$5 == num'     # zmienna "var" jest wartością numeryczną.
```

*Zmienne wbudowane:*

`ARGC` ilość argumentów przekazywanych z konsoli systemowej.
`ARGV` tablica wartości argumentów przekazywanych z linii poleceń systemu (ARGV[0...ARVC-1]).
`FILENAME` nazwa obecnie analizowanego pliku wejściowego.
`FNR` ilość wszystkich rekordów w obecnie przetwarzanym pliku wejściowym.
`FS` wejściowy separator pola (domyślnie spacja).
`NF` ilość pól w wejściowym i obecnym wierszu.
`NR` numer odpowiadający obecnemu, wejściowemu rekordowi liczony od początku pliku wejściowego.
`OFMT` definicja wyjściowej reprezentacji formatu dla liczb (domyślnie "%.6g").
`OFS` wyjściowy separator pola (domyślnie spacja).
`ORS` wyjściowy separator wiersza (domyślnie znak nowej linii).
`RLENGTH` długość łańcucha zwróconego w wyniku dopasowania wyrażenia regularnego.
`RS` wejściowy separator rekordu (domyślnie znak nowej linii).
`RSTART` początkowa pozycja łańcucha zwróconego przez dopasowanie.
`SUBSEP` separator tablicy indeksów dolnych formularza [i,j,...] (domyślnie ^\).

*Sekwencje wyjściowe:*

`\b` znak cofnięcia się (^H),
`\n` znak nowej linii (DOS, CR/LF; Unix, LF),
`\r` znak powrotu karetki,
`\t` znak tabulacji (^I),
`\ddd` wartość oktalna *`ddd'*, gdzie *`ddd'* to cyfry od 1 do 3, z zakresu pomiędzy 0 i 7.
`\c` każdy inny znak jest dosłowny, np.: \" for " i \\ for \.

*Funkcje znakowe w AWK:*

W poniższych funkcjach wyróżniamy następujące terminy:

- *`r'* reprezentuje wyrażenie regularne *"regexp"*,
- *`s'* i *`t'* to łańcuchy,
- *`i'* i *`n'* to liczby całkowite,
- *`&'* w zastępowanym łańcuchu, w poleceniu *SUB* lub *GSUB*, jest zastępowany przez dopasowany łańcuch.

`gsub(r,s,t)` zastępuje wszystkie możliwe wyrażenia regularne *r,* łańcuchem *s*, znalezione w wejściowych danych *t*. W rezultacie zwrócona zostaje ilość udanych zastąpień.

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

Jeżeli łańcuch *t* jest pominięty, wtedy domyślnie stosowany jest cały wiersz ($0).

`gensub(r,s,h,t)` zastępuje *h* pierwszych wyrażeń regularnych *r,* łańcuchem *s*, znalezionych w wejściowych danych *t*. Jeżeli *h = 'g'* to w wyniku uzyskamy postać funkcji `gsub(r,s,t)`. W rezultacie zwrócony zostaje skonwertowany wzorzec, a nie ilość udanych zastąpień. Jeżeli łańcuch *t* jest pominięty, wtedy domyślnie stosowany jest cały wiersz ($0).

`index(s,t)` zwrócona zostaje całkowita liczba reprezentująca indeks początku łańcucha *t* w danych *s*, lub 0 jeżeli łańcuch *s* nie zawiera łańcucha *t*.

`length(s)` zwrócona zostaje całkowita liczba reprezentująca długość łańcucha *s.*

`match(s,r)` zwrócona zostaje całkowita liczba reprezentująca indeks początku dopasowania łańcucha *r* w danych *s* lub 0, jeżeli dopasowanie nie występuje. W rezultacie ustawiane są wartości następujących zmiennych `RSTART` i `RLENGTH`.

`split(s,a,fs)` w rezultacie uzyskiwany zostaje podział łańcucha *s* na pola, przechowywany w tablicy *a*, zgodnie ze zdefiniowanym separatorem pól *fs*. W rezultacie zwrócona zostaje ilość pól, na którą nastąpił podział. Jeżeli szczegółowy separator pola *fs* jest pominięty, wtedy jest wykorzystywany domyślny separator pola FS.

`sprintf(fmt,expr-list)` zwrócona zostanie lista wyrażeń *expr-list* sformatowana zgodnie z definicją formatu *fmt*.

`sub(r,s,t)` zachowanie podobnie do funkcji `gsub`, różniący się tylko faktem, że tylko pierwsze dopasowanie jest zastępowane.

`substr(s,i,n)` zwrócony zostaje podłańcuch o długości *n* łańcucha wejściowego *s* rozpoczynający się od indeksu *i*. Jeżeli wartość *n* zostanie pominięta, wtedy zostanie zwrócony podłańcuch rozpoczynający się od indeksu *i*, a kończący się końcem obecnego wiersza.

*Funkcje arytmetyczne:*

`atan2(y,x)` arcustangens z y/x w radianach.

`cos(x)` cosinus (kąt w radianach).

`exp(n)` wykładnicza (n nie musi być liczbą całkowitą).

`int(x)` skrócenie liczby zmiennopozycyjnej do liczby całkowitej.

`log(x)` logarytm naturalny.

`rand()` pseudo-losowa liczba.

`sin(x)` sinus (kąt w radianach).

`sqrt(x)` pierwiastek z *x*.

`srand(x)` ustawia nową wartość ziarna wykorzystywanego przez generator liczb losowych; zwykle wykorzystywany jest aktualny czas, jeżeli *x* nie zostało zdefiniowane.

*Funkcje definiowane przez użytkownika:*

Postać funkcji jest podobna do języka C. Definicja funkcji składa się ze słowa kluczowego `function`, nazwy funkcji, nazw argumentów wejściowych i definicji ciała funkcji.
Przykład:

```
function add_three(number, temp) {
  temp = number + 3
  return temp
}
```

Wyrażenie może być wywołane w następujący sposób:

```
print add_three(36)      # wypisuje na wyjście 39
```

*Tablice adresowane zawartością (associative arrays):*

Przykład, którego zadaniem jest zliczenie częstotliwości (ilości) występowania słów w pliku wejściowym.

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

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

# Dodatkowe materiały

- Zbiór książek do AWK dostępnych w celu darmowego pobrania - `http://www.computer-books.us/awk.php`,
- Internetowa dokumentacja do AWK - `http://www.cs.bell-labs.com/cm/cs/awkbook/index.html`,
- Zakładka *„materiały dla studentów”* dla przedmiotu *„języki formalne i kompilatory”* na stronie domowej dr inż. W. Complaka – `http://www.cs.put.poznan.pl/wcomplak`.

# Zadania

**Zad. 1(*).** Skompiluj i wykonaj programy prezentowane na wykładzie.

**Zad. 2.** Zaznaczyć poprawne odpowiedzi (wiele może być poprawnych).

a) AWK jest wykorzystywany do projektowania aplikacji biznesowych,
b) AWK to proste narzędzie pozwalające konwertować pliki tekstowe o jasno i ściśle zdefiniowanej strukturze,
c) AWK to język wykorzystywany podczas modelowania oprogramowania,
d) składnia języka przypomina Pascal,
e) składnia języka przypomina C,
f) składnia języka przypomina Asembler,
g) przewaga nad innymi językami wynika głównie ze zwięzłości kodu źródłowego,
h) kod jest trudny w zrozumieniu, gdyż charakteryzuje się dużą rozpiętością.

**Zad. 3.** Zaznaczyć poprawne odpowiedzi (wiele może być poprawnych). Plik wejściowy, który ma zostać poddany analizie, składa się z:

a) kolumn i wierszy,
b) kolumn i pól,
c) pól i wierszy.

**Zad. 4.** Zaznaczyć poprawne odpowiedzi (wiele może być poprawnych). Każdy plik źródłowy napisany w AWK składa się z:

a) dyrektyw,
b) reguł przetwarzania (wzorzec, instrukcje),
c) znaczników XMLowych.

**Zad. 5.** Zaznaczyć poprawne odpowiedzi (wiele może być poprawnych). Załóżmy, że mamy dostępny plik wejściowy, który zawiera kolejne wiersze oraz plik źródłowy, który zawiera kolejne reguły. Przetwarzanie w AWK jest realizowane w następujący sposób:

a) dla każdego wiersza w pliku wejściowym analizowane są kolejno od góry wszystkie reguły zapisane w pliku źródłowym; jeżeli obecnie analizowana zawartość wiersza pliku wejściowego spełnia obecnie sprawdzany wzorzec, będący częścią obecnej reguły przetwarzania, wtedy na tym wierszu dokonywane są operacje zdefiniowane w części instrukcji reguły przetwarzania i ich rezultat jest zapisywany na standardowe wyjście; w przeciwnym razie nie są wykonywane żadne operacje,
b) dla każdej reguły przetwarzania w pliku źródłowym analizowane są kolejno od góry wszystkie wiersze zapisane w pliku wejściowym; na każdym analizowanym wierszu pliku wejściowego dokonywane są operacje zdefiniowane w części instrukcji reguły przetwarzania; następnie analizowana jest zawartość tegoż wiersza pod kątem zgodności z obecnym wzorcem; jeżeli zgodność się potwierdzi wtedy przechodzimy do kolejnego wiersza; w przeciwnym razie cofany jest rezultat wykonanych wcześniej instrukcji.

**Zad. 6.** Zaznaczyć poprawne odpowiedzi (wiele może być poprawnych).

a) pola w AWK są adresowane za pomocą znaku $ i numeru całkowitego pola (np.: 1,2,3,4,...),
b) $0 oznacza pole o numerze 0,
c) $0 oznacza bieżący wiersz.
d) początek komentarza w AWK oznaczany jest za pomocą „//”,
e) początek komentarza w AWK oznaczany jest za pomocą „#”.

**Zad. 7.** W AWK wyróżniamy dwa specyficzne wzorce, które oznaczają:

a) ..... – początek tekstu,
b) ..... – koniec tekstu.

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

**Zad. 8.** Wszystkie relacje traktują operandy, czyli pola pliku wejściowego, na których są one wykonywane jako:

a) jako liczby,
b) jako łańcuchy (porządek leksykograficzny),
c) mieszanie w zależności od zawartości pól.

**Zad. 9.** Podaj przykład i na jego podstawie wyjaśnij ideę działania wzorców zakresu.

**Zad. 10.** Podaj znaczenie następujących znaków specjalnych:

a) „^" - .....,
b) „$" - .....,
c) „." - .....,
d) „[ ]" - .....,
e) „\n" - .....,
f) „\." - .....,
g) „\"" - .....,
h) „\ddd" - .....

**Zad. 11.** Wyrażenie regularne to wzorzec opisujący zbiór ciągów znaków. W takim razie zdefiniuj znaczenie następujących wzorców:

a) 1+ - .....,
b) $1 ~ /1/ - .....,
c) $0 !~ /a/ - .....(!/a/),
d) /^.$/ -.....,
e) /[0-9]/ -.....,
f) !/[0-9]/ -.....,
g) /^[0-9]/ - .....,
h) /[^0-9]/ -.....,
i) /[Adam | Jurek]/ -.....,
j) /[0-9]*/ - .....,
k) $2 ~ /^[0-9][0-9]*$/ - .....,
l) /[0-9]+(D|Q)?/ - .....,
m) /[0-9]+D|Q?/ - .....

**Zad. 12.** Co to są zmienne wbudowane? Na wykładzie poznaliście Państwo następujące zmienne wbudowane:

a) „NF" - .....,
b) „NR" - .....,
c) „FILENAME" - .....

**Zad. 13.** W skład zaawansowanych mechanizmów AWK wchodzą:

a) .....,
b) .....,
c) .....,
d) .....,
e) .....

**Zad. 14.** Podczas uruchamiania programu wykorzystujemy:

a) interpreter awk np.: gawk,
b) plik zawierający dane do analizy (wyjściowy),
c) plik źródłowy .awk,
d) plik z rezultatami konwersji (wyjściowy).

**Zad. 15.** Napisz program, którego zadaniem będzie wypisanie długości najdłuższego wiersza pliku wejściowego, przy jednoczesnym podaniu długości tegoż najdłuższego wiersza i jego samego.

*Plik wejściowy*

```
FName:Alek SName:Gor Salary 700
FName:Jurek SName:Busz Salary 585
FName
```

*Plik wyjściowy*

```
The longest record is FName:Jurek SName:Busz Salary 585
Its size is 33
```

**Zad. 16.** Napisz program, którego zadaniem będzie wypisanie sześciu losowo wygenerowanych liczb z przedziału 0-100 włącznie.

*Plik wyjściowy*

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

```
0
56
19
81
59
48
```

**Zad. 17.** Napisz program, którego zadaniem będzie wypisanie, co drugich wierszy z pliku wejściowego.

*Plik wejściowy*
Patrz zadanie 15.

*Plik wyjściowy*
```
FName:Alek SName:Gor Salary 700
FName
```

**Zad. 18.** Napisz program, którego zadaniem będzie wypisanie, wszystkich pól wchodzących w skład wierszy w odwrotnej kolejności.

*Plik wejściowy*
Patrz zadanie 15.

*Plik wyjściowy*
```
700 Salary SName:Gor FName:Alek
585 Salary SName:Busz FName:Jurek
FName
```

**Zad. 19.** Napisz program, którego zadaniem będzie wypisanie, ilości linii zawierających wzorzec „*SName”*.

*Plik wejściowy*
Patrz zadanie 15.

*Plik wyjściowy*
```
2
```

**Zad. 20.** Napisz program, którego zadaniem będzie wypisanie, wszystkich wierszy, w których zawartość pola pierwszego jest różna od odpowiadającej jej zawartości pola pierwszego w wierszu poprzedzającym.

*Plik wejściowy*

```
FName:Alek SName:Gor Salary 700
FName:Jurek SName:Busz Salary 585
FName:Jurek SName:Zielony Salary 200
Fname:
Fname:
FName
```

*Plik wyjściowy*
```
FName:Alek SName:Gor Salary 700
FName:Jurek SName:Busz Salary 585
Fname:
FName
```

**Zad. 21.** Załóżmy, że w pliku wejściowym znajdują się wiersze zawierające tylko pola numeryczne. Napisz program, którego zadaniem będzie wypisanie, wartości bezwzględnych dla wszystkich pól znajdujących się w pliku wejściowym.

*Plik wejściowy*
```
1     -2 3  -5 2
0 10  -12 -2 3    1
```

*Plik wyjściowy*
```
1 2 3 5 2
0 10 12 2 3 1
```

**Zad. 22.** Napisz program, którego zadaniem będzie wypisanie, numeru telefonu pracownika o nazwisku *Jones* z pliku wejściowego o poniższej strukturze.

*Plik wejściowy*

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

```
000902|Beavis|Theodore|333-242-2222|149092
000901|Jones|Bill|532-382-0342|234023
```

*Plik wyjściowy*

```
532-382-0342
```

**Zad. 23.** Napisz program, którego zadaniem będzie wypisanie, maksymalnej i minimalnej wartości numerycznego pola pierwszego znajdującego się w pliku wejściowym.

*Plik wejściowy*

Patrz zadanie 22.

*Plik wyjściowy*

```
Max = 000902    Min = 000901
```

**Zad. 24.** Napisz program, którego zadaniem będzie wypisanie, w kolejnych wyjściowych wierszach, wartości pięciu kolejnych wyrazów ciągu fibonacciego.

*Plik wejściowy*

```
5
```

*Plik wyjściowy*

```
1
1
3
5
8
```

**Zad. 25.** Zakładamy, że plik wejściowy zawiera wiersze, które są posortowane względem pola pierwszego. Napisz program, którego zadaniem będzie wypisanie, wszystkich pierwszych unikalnych (względem pierwszego pola) wierszy.

*Plik wejściowy*

```
1 ala ma kota
1 zenek ma kota
1 ela ma kota
2 kot ma pilke
3 ewa ma psa
3 jozek ma psa
```

*Plik wyjściowy*

```
1 ela ma kota
2 kot ma pilke
3 jozek ma psa
```

**Zad. 26.** Napisz program, którego zadaniem będzie wypisanie, wszystkich wierszy w osobnych paragrafach, zawierających wzorzec pobierany jako atrybut podczas wykonywania programu oraz ilości tych dopasowanych wierszy.

*Plik wejściowy*

Patrz zadanie 25.

Wywołanie programu: `awk95 -v search="kot" -f prog.awk 0<in.txt 1>out.txt`

*Plik wyjściowy*

```
1 ala ma kota

1 zenek ma kota

1 ela ma kota

2 kot ma pilke

=============
Search matched 4 times.
The expresssion to search for was [kot].
```

**Zad. 27(*).** Napisz program, którego zadaniem będzie odpowiednie przekonwertowanie pliku wejściowego w odpowiadający mu plik wyjściowy, przy jednoczesnym podaniu jako atrybut wejściowy ilości spacji, za pomocą, których będą tworzone wcięcia. Oba pliki są zdefiniowane poniżej:

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

Wywołanie: `awk95 -v num=1 -f prog.awk 0<in.txt 1>out.txt`

*Plik wejściowy* | *Plik wyjściowy*

```
* Line 1          | 1. Line 1
** Line 2         |   1.1. Line 2
*** Line 3        |     1.1.1. Line 3
*** Line 4        |     1.1.2. Line 4
**** Line 5       |       1.1.2.1. Line 5
***** Line 6      |         1.1.2.1.1. Line 6
***** Line 7      |         1.1.2.1.2. Line 7
** Line 8         |   1.2. Line 8
* Line 9          | 2. Line 9
** Line 10        |   2.1. Line 10
```

**Zad. 28(*).** Napisz program, którego zadaniem będzie odpowiednie przekonwertowanie pliku wejściowego w odpowiadający mu plik wyjściowy, przy jednoczesnym podaniu jako atrybut wejściowy ilości spacji, za pomocą, których będą tworzone wcięcia. Oba pliki są zdefiniowane poniżej:
Wywołanie: `awk95 -v num=1 -f prog.awk 0<in.txt 1>out.txt`

*Plik wejściowy* | *Plik wyjściowy*

```
* Line 1          | A. Line 1
** Line 2         |   1. Line 2
*** Line 3        |     a. Line 3
*** Line 4        |     b. Line 4
**** Line 5       |       (1) Line 5
***** Line 6      |         (a) Line 6
***** Line 7      |         (b) Line 7
** Line 8         |   2. Line 8
* Line 9          | B. Line 9
** Line 10        |   1. Line 10
```

**Zad. 29.** Zakładamy, że wszystkie pola w pliku wejściowym mają charakter numeryczny. Napisz program, którego zadaniem będzie wypisanie sumy wszystkich pól znajdujących się we wszystkich wierszach pliku wejściowego.

*Plik wejściowy*
Patrz zadanie 21.

*Plik wyjściowy*

```
-1
0
```

**Zad. 30.** Napisz program, którego zadaniem będzie wypisanie, ilości wszystkich słów we wszystkich wierszach (separator słów to domyślnie spacja).

*Plik wejściowy*
Patrz zadanie 21.

*Plik wyjściowy*

```
5
6
```

**Zad. 31.** Napisz program, którego zadaniem będzie wypisanie, wszystkich wierszy poprzedzonych ilością pól w nich występujących i znakiem „*:*” (separatorem pola jest „ ”).

*Plik wejściowy*

```
1,-2,3,-5,2
0,10,-12,-2,3,1
```

*Plik wyjściowy*

```
5:1,-2,3,-5,2
6:0,10,-12,-2,3,1
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

**Zad. 32.** Napisz program, którego zadaniem będzie wypisanie, wszystkich wierszy przy jednoczesnym usunięciu z ich początku wszystkich spacji i tabulatorów, powodując, że tekst będzie przypisany do lewej.

*Plik wejściowy*

```
                    1       -2 3    -5 2
               0 10   -12 -2 3        1
```

*Plik wyjściowy*

```
1     -2 3  -5 2
0 10  -12 -2 3    1
```

**Zad. 33.** Napisz program, którego zadaniem będzie wypisanie, wszystkich wierszy przy jednoczesnym usunięciu z ich końca wszystkich spacji i tabulatorów.

*Plik wejściowy*

```
1       -2 3    -5 2
0 10    -12 -2 3        1
```

*Plik wyjściowy*

```
1     -2 3  -5 2
0 10  -12 -2 3    1
```

**Zad. 34.** Napisz program, którego zadaniem będzie centrowanie wszystkich wierszy na 30 znaku szerokości.

*Plik wejściowy*

Patrz zadanie 21.

*Plik wyjściowy*

```
          1  -2 3  -5 2
       0 10 -12 -2 3    1
```

**Zad. 35.** Napisz program, którego zadaniem będzie zastąpienie ciągu *„foo”*, łańcuchem *„bar”*:

*Plik wejściowy*

```
foo afoo befoo allafoo bellafoo fooella
foo afoo befoo allafoo bellafoo fooella bazella bazolla ollabaz
bazo obaz
```

a) pierwsze wystąpienie,

   *Plik wyjściowy*

   ```
   bar afoo befoo allafoo bellafoo fooella
   bar afoo befoo allafoo bellafoo fooella bazella bazolla ollabaz
   bazo obaz
   ```

b) wszystkie dostępne wystąpienia,

   *Plik wyjściowy*

   ```
   bar abar bebar allabar bellabar barella
   bar abar bebar allabar bellabar barella bazella bazolla ollabaz
   bazo obaz
   ```

c) wszystkie dostępne wystąpienia w momencie, gdy wiersz zawiera wzorzec *„baz”*,

   *Plik wyjściowy*

   ```
   bar abar bebar allabar bellabar barella bazella bazolla ollabaz
   bazo obaz
   ```

d) wszystkie dostępne wystąpienia w momencie, gdy wiersz nie zawiera wzorca *„baz”*,

   *Plik wyjściowy*

   ```
   bar abar bebar allabar bellabar barilla
   ```

e) wszystkie dostępne wystąpienia lub dodatkowo jeszcze wszystkie dostępne wystąpienia ciągu *„baz”* również łańcuchem *„bar”*.

   *Plik wyjściowy*

   ```
   bar abar bebar allabar bellabar barella
   bar abar bebar allabar bellabar barella barella barolla ollabar
   baro obar
   ```

**Zad. 36.** Napisz program, którego zadaniem będzie wypisanie wszystkich wierszy w odwrotnej kolejności niż znajdują się one w pliku wejściowym.

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

*Plik wejściowy*
Patrz zadanie 35.

*Plik wyjściowy*
```
bazo obaz
foo afoo befoo allafoo bellafoo fooella bazella bazolla ollabaz
foo afoo befoo allafoo bellafoo fooella
```

**Zad. 37.** Napisz program, którego zadaniem będzie wypisanie wszystkich wierszy przy jednoczesnym sprawdzeniu czy obecny wiersz kończy się backslashem. Jeżeli tak wtedy zastępowany jest ten backslash wartością następnego wiersza z pliku wejściowego.

*Plik wejściowy*
```
foo afoo befoo allafoo bellafoo fooella
foo afoo befoo allafoo bellafoo fooella bazella bazolla ollabaz\
bazo obaz
```

*Plik wyjściowy*
```
foo afoo befoo allafoo bellafoo fooella
foo afoo befoo allafoo bellafoo fooella bazella bazolla ollabazbazo obaz
```

**Zad. 38.** Napisz program, którego zadaniem będzie wypisanie wszystkich wierszy przy jednoczesnym odwróceniu kolejności dwóch pierwszych pól każdego wiersza.

*Plik wejściowy*
Patrz zadanie 35.

*Plik wyjściowy*
```
afoo foo befoo allafoo bellafoo fooella
afoo foo befoo allafoo bellafoo fooella bazella bazolla ollabaz
obaz bazo
```

**Zad. 39.** Napisz program, którego zadaniem będzie wypisanie wszystkich wierszy przy jednoczesnym usunięciu drugiego pola z zawartości każdego wiersza.

*Plik wejściowy*
Patrz zadanie 35.

*Plik wyjściowy*
```
foo  befoo allafoo bellafoo fooella
foo  befoo allafoo bellafoo fooella bazella bazolla ollabaz
bazo
```

**Zad. 40.** Napisz program, którego zadaniem będzie wypisanie wszystkich w odwróconej kolejności wierszy przy jednoczesnym odwróceniu kolejności wszystkich pól wchodzących w skład każdego wiersza.

*Plik wejściowy*
Patrz zadanie 35.

*Plik wyjściowy*
```
obaz bazo
ollabaz bazolla bazella fooella bellafoo allafoo befoo afoo foo
fooella bellafoo allafoo befoo afoo foo
```

**Zad. 41.** Napisz program, którego zadaniem będzie wypisanie wszystkich unikalnych wierszy.

*Plik wejściowy*
```
ala
bala
ala
bala
bola
bela
```

*Plik wyjściowy*
```
ala
bala
bola
bela
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

**Zad. 42.** Napisz program, którego zadaniem będzie wypisanie wszystkich linii będących wynikiem konkatenacji kolejnych 5 wierszy pliku wejściowego, przy założeniu, że pola będą separowane przecinkiem.

*Plik wejściowy*
Patrz zadanie 41.

*Plik wyjściowy*
```
ala,bala,ala,bala,bola
bela
```

**Zad. 43.** Napisz program, którego zadaniem będzie wypisanie pierwszej linii pliku wejściowego.

*Plik wejściowy*
Patrz zadanie 41.

*Plik wyjściowy*
```
ala
```

**Zad. 44.** Napisz program, którego zadaniem będzie wypisanie ostatnich dwóch wierszy pliku wejściowego w odwrotnej kolejności.

*Plik wejściowy*
Patrz zadanie 41.

*Plik wyjściowy*
```
bola
bela
```

**Zad. 45.** Napisz program, którego zadaniem będzie wypisanie bezpośrednio poprzedzający wiersz, wiersza zawierającego wyrażenie regularne *„bola”*.

*Plik wejściowy*
Patrz zadanie 41.

*Plik wyjściowy*
```
bala
```

**Zad. 46.** Napisz program, którego zadaniem będzie wypisanie bezpośrednio następny wiersz, wiersza zawierającego wyrażenie regularne *„bola”*.

*Plik wejściowy*
Patrz zadanie 41.

*Plik wyjściowy*
```
bela
```

**Zad. 47.** Napisz program, którego zadaniem będzie wypisanie wszystkich wierszy, których długość znajduje się w zakresie pomiędzy 3 i 10 znaków.

*Plik wejściowy*
```
a
bala
ala
bala
bola
bela
dsajdlkas dsadlsajd
```

*Plik wyjściowy*
```
bala
ala
bala
bola
bela
```

**Zad. 48.** Napisz program, którego zadaniem będzie wypisanie zawartości pliku wejściowego od wyrażenia regularnego *„ola”*.

*Plik wejściowy*
Patrz zadanie 47.

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

*Plik wyjściowy*

```
bola
bela
dsajdlkas dsadlsajd
```

**Zad. 49.** Napisz program, którego zadaniem będzie wypisanie wszystkich wierszy z zakresu od 5-6 włącznie. Jak można zoptymalizować wykonanie programu, gdy analizujemy duży plik wejściowy?.

*Plik wejściowy*

Patrz zadanie 47.

*Plik wyjściowy*

```
bola
bela
```

**Zad. 50.** Napisz program, którego zadaniem będzie wypisanie części pliku wejściowego znajdującego się pomiędzy pierwszymi rekordami znajdującymi następujące wzorce *„ola”* i *„ela”*.

*Plik wejściowy*

Patrz zadanie 47.

*Plik wyjściowy*

```
bola
bela
```

**Zad. 51.** Napisz program, którego zadaniem będzie wyznaczenie końcowej wypłaty (wypłata*dodatek) dla danego pracownika, przy jednoczesnym założeniu, że długość nazwiska pracownika powinna być równa 8 znaków (w razie niedoboru dopełniamy spacjami), a dokładność końcowej wypłaty powinna się zmieścić w 6 znakach, z czego dwa znaki to wartości po przecinku.

*Plik wejściowy*

```
      Wypłata     Dodatek
Beth  4.00        0
Dan   3.75        0
Kathy 4.00        10
Mark  5.00        20
Mary  5.50        22
Susie 4.25        18
```

*Plik wyjściowy*

```
Beth     is $  0,00
Dan      is $  0,00
Kathy    is $ 40,00
Mark     is $100,00
Mary     is $110,00
Susie    is $ 72,00
```

**Zad. 52.** Napisz program, którego zadaniem będzie wypisanie podsumowania całkowitego dla obszaru i populacji zgrupowanych po wszystkich przedstawionych poniżej państwach.

*Plik wejściowy*

```
USSR        8649  275   Asia
Canada      3852  25    North America
China       3705  1032  Asia
USA         3615  237   North America
Brazil      3286  134   South America
India       1267  746   Asia
Mexico      762   78    North America
France      211   55    Europe
Japan       144   120   Asia
Germany     96    61    Europe
England     94    56    Europe
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

*Plik wyjściowy*

```
   COUNTRY   AREA   POP   CONTINENT

      USSR   8649   275   Asia
    Canada   3852    25   North America
     China   3705  1032   Asia
       USA   3615   237   North America
    Brazil   3286   134   South America
     India   1267   746   Asia
    Mexico    762    78   North America
    France    211    55   Europe
     Japan    144   120   Asia
   Germany     96    61   Europe
   England     94    56   Europe

     TOTAL  25681  2819
```

**Zad. 53.** Napisz program, którego zadaniem będzie utworzenie dwóch nowych plików w rezultacie: *"bigpop"*, *"smallpop"* w zależności od progowej wartości populacji wczytanej jako parametr uruchomieniowy podczas wywołania programu.

*Plik wejściowy*

Patrz zadanie 52.

Wywołanie: `awk95 -f prog.awk 0<in.txt`

*Pliki wyjściowe*

*"bibpop"*

```
USSR 275
China 1032
USA 237
Brazil 134
India 746
Japan 120
```

*"smallpop"*

```
Canada 25
Mexico 78
France 55
Germany 61
England 56
```

**Zad. 54.** Napisz program, którego zadaniem będzie wypisanie wszystkich argumentów przekazywanych z linii poleceń podczas uruchomienia programu.

*Plik wejściowy*

Wywołanie: `awk95 -f prog.awk 1<out.txt`

*Pliki wyjściowe*

```
awk95
```

**Zad. 55.** Napisz program, którego zadaniem będzie wypisanie statystyk związanych z wartościami numerycznymi znajdującymi się w kolumnach. W skład statystyk dla każdej kolumny ma zostać wypisana suma wszystkich pól tej kolumny oraz średnia związana z daną kolumną.

*Plik wejściowy*

```
1     2 3   5 2
0 10  12 2 3      1
```

*Plik wyjściowy*

*Suma: 1 12 15 7 5 1*

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

*Srednia: 0,5 6 7,5 3,5 2,5 1*

**Zad. 56.** Przyjmijmy, że słowo jest to niepusty ciąg znaków różnych od spacji, znaku tabulacji i nowej linii (zatem ciąg a+b*c jest jednym słowem). Napisać program obliczania liczby słów w pliku. Wiersze, w których pierwszym znakiem jest średnik nie są brane pod uwagę.

*Plik wejściowy*
```
jeden  2  !!!
; on on on on on on on on
;on on on on on on on on
four ;; six
```
*Plik wyjściowy*
```
6
```

**Zad. 57.** Plik wejściowy zawiera dowolny tekst. W tekście tym mogą występować następujące sekwencje: inicjał imienia (duża litera z kropką) a za tym bezpośrednio nazwisko. Każdą taką sekwencję należy odwrócić, tzn. najpierw napisać nazwisko, potem jedną spację i inicjał imienia (czyli dużą literę z kropką). Przy okazji należy znormalizować odstępy do postaci jednej spacji. Pozostałe znaki mają być przepisane bez zmian.

*Plik wejściowy*
```
J.Nawrocki           i W.Complak

Plik    z     danymi to t.in.
Istotna jest liczba K. C jest licznikiem.
```
*Plik wyjściowy*
```
Nawrocki J. i Complak W.

Plik z danymi to t.in.
Istotna jest liczba K. C jest licznikiem.
```

**Zad. 58.** Plik wejściowy ma postać trzech kolumn. Pierwsza kolumna zawiera imię pracownika, druga podaje jego nazwisko, a w trzeciej znajduje się jego zarobek. Pierwszy wiersz zawiera opisy poszczególnych kolumn (patrz przykład). Napisać program obliczania łącznego zarobku wszystkich pracowników.

*Plik wejściowy*
```
Imię        Nazwisko    Zarobek
Wojciech    Complak     458.50
Jerzy       Nawrocki    501.50
Iwo         Lewandowski 440.00
```
*Plik wyjściowy*
```
razem=1399
```

**Zad. 59.** Plik wejściowy ma postać macierzy o wymiarach *m x n* (*n* może być różne od *m*). Napisać program obliczania sumy elementów leżących na przekątnej głównej (czyli od lewego górnego rogu do prawego dolnego).

**Przykład**

*Plik wejściowy*
```
5  0  1
0  5  0
1  0  5
```
*Plik wyjściowy*
```
15
```
*Plik wejściowy*
```
6  0  1  7
0  6  0  6
1  0  6  7
```
*Plik wyjściowy*
```
18
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

*Plik wejściowy*
```
6  0
0  6
1  2
```
*Plik wyjściowy*
```
12
```

**Zad. 60.** Plik wejściowy zawiera nieinteresujące i interesujące fragmenty (ciągi wierszy). Początek interesującego fragmentu zaczyna się od wiersza, w którym pierwsze trzy znaki mają postać "$$$". Ostatni wiersz interesującego fragmentu zaczyna się od "!!!".
Należy napisać program w AWK, który przepisuje na wyjście tylko interesujące fragmenty i w każdym takim fragmencie zastępuje każdą spację symbolem podkreślenia.
**Przykład**
*Plik wejściowy* (każda linia ma na końcu jedną spację przed znakiem końca wiersza):
```
1:   X X
  $$$
2:   X
!!!
$$$
3:    X
$$$
4:   X X
!!!
```
*Plik wyjściowy*
```
__$$$_
2:___X
!!!
$$$_
3:____X_
$$$_
4:___X_X_
!!!_
```

**Zad. 61.** W każdej linii pliku wejściowego znajduje się niepusty ciąg nieujemnych liczb całkowitych określających współczynniki przy kolejnych (od najwyższych do najniższych) potęgach wielomianu. Należy napisać program rozwijający skrótowy zapis wielomianu, to znaczy:
- do każdego niezerowego współczynnika należy dopisać ciąg „x^n” gdzie n jest potęgą wynikającą z pozycji współczynnika,
- jeżeli współczynnik jest zerowy należy pominąć odpowiedni wyraz.
**Przykład**
*Plik wejściowy*
```
4 5 0 2
3 2 1
```
*Plik wyjściowy*
```
4x^3+5x^2+2
3x^2+2x+1
```

**Zad. 62.** Plik wejściowy zawiera ciąg wierszy. W każdym wierszu znajdują się trzy liczby całkowite (o małych wartościach bezwzględnych), oddzielone od siebie dowolną liczbą spacji i znaków tabulacji. Napisać program obliczający dla każdego wiersza sumę liczb w nim zawartych i drukujący zestawienie tabelaryczne uzyskanych wyników w postaci: nagłówek 'SUMA' dla ostatniej kolumny, w kolejnych wierszach, wyrównane w kolumnach do lewej składowe sumy i wartość sumy, oddzielone od siebie znakami '|'.
*Plik wejściowy*
```
1     5 18
13  2     -5
```
*Plik wyjściowy*
```
                          SUMA
| 1      | 5      | 18     | 24
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

```
| 13    | 2     | -5    | 10
```

**Zad. 63.** Plik wejściowy składa się z ośmiu kolumn. Pierwsza i druga kolumna określają imię i nazwisko studenta, pozostałe zaś podają liczbę punktów uzyskanych z poszczególnych sprawdzianów (6 sprawdzianów). Może się zdarzyć, że student nie pisał jakiegoś sprawdzianu i wtedy w odpowiedniej kolumnie jest puste miejsce. Napisać program obliczający dla każdego studenta łączną liczbę uzyskanych punktów. Plik wyjściowy ma składać się z trzech kolumn: imienia, nazwiska i łącznej liczby punktów. Pierwsza kolumna ma mieć szerokość 10 znaków, natomiast druga - 15 znaków. Między kolumnami pierwszą, a drugą i drugą, a trzecią ma się znajdować dodatkowa spacja w roli separatora.

*Plik wejściowy*

```
Wojciech    Complak  5 5 5 4 4 4
Jerzy       Nawrocki 5   4 3   2
Czerwony    Kapturek 2 3 4 5 5 5
Jan         Nieuk
```

*Plik wyjściowy*

```
Wojciech   Complak         27
Jerzy      Nawrocki        14
Czerwony   Kapturek        24
Jan        Nieuk           0
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.