Awk w przykładach, część trzecia

Spis treści: 1. Łańcuchy, funkcje i... książeczki czekowe? ▾

1. Łańcuchy, funkcje i... książeczki czekowe?

Formatowanie wyjścia

O ile przez większość czasu intrukcja print wykonuje świetną robotę to czasem potrzebne jest jednak więcej. Dla takich sytuacji awk udostępnia dwie dobrze znane funkcje - printf() oraz sprintf(). Tak, te funkcje, podobnie jak wiele innych fragmentów awk, są identyczne jak ich odpowiedniki w C. printf() wydrukuje sformatowany łańcuch na standardowym wyjściu, a sprintf zwróci go jako nowy łańcuch. Opis tych funkcji można znaleźć we wprowadzeniu do języka C, ewentualnie można obejrzeć Uniksową stronę manuala o printf pisząc "man 3 printf".

Poniżej znajduje się przykładowy kod wykorzystujący sprintf() oraz printf(). Jak widać, wszystko działa niemal identycznie jak w C.

Listing 1.1: Przykładowy skrypt awk wykorzystujący printf() oraz sprintf()

```
x=1
b="foo"
printf("%s uzyskał/a %d podczas ostatniego testu\n","Jim",83)
myout=sprintf("%s-%d",b,x)
print myout
```

Ten kod da w wyniku:

Listing 1.2: Wynik

```
Jim got a 83 on the last test
foo-1
```

Funkcje łańcuchowe

Awk dysponuje bardzo szerokim zestawem funkcji narzędziowych przeznaczonych do obsługi łańcuchów. To bardzo korzystne, tym bardziej, że w tym języku są one naprawdę potrzebne, gdyż nie można w nim tratować łańcucha jako tablicy znaków w taki sposób, jak robi się to w C, C++ oraz Python. Na przykład, jeżeli spróbuje się wykonać następujący kod:

Listing 1.3: Przykład

```
mystring="Jak się dzisiaj masz?"
print mystring[3]
```

To w rezultacie pojawi się błąd:

Listing 1.4: Przykładowy błąd

```
awk: string.gawk:59: fatal: attempt to use scalar as array
```

No cóż. Wprawdzie nie są takie wygodne jak pythonowe typy sekwencyjne, ale funkcje narzędziowe spełniają swoje zadanie. Przyjrzyjmy się im.

Po pierwsze dysponujemy funkcją length(), która zwraca długość łańcucha. Wykorzystujemy ją w następujący sposób:

Listing 1.5: Przykład wykorzystania length()

```
print length(mystring)
```

Kod da następujący wynik:

Listing 1.6: Wypisana wartość

```
22
```

Kolejna funkcja nazywa się index i zwraca pozycję wystąpienia podłańcucha w łańcuchu lub 0, jeżeli nie można go odnaleźć:

Listing 1.7: Przykład działania index()

```
print index(mystring,"się")
```

Awk napisze:

Listing 1.8: Wynik funkcji

```
5
```

Kolejne dwie proste funkcje, tolower() oraz toupper(), służą do zmiany liter na odpowiednio małe oraz wielkie. Funkcje te zwracają nowy łańcuch, a nie modyfikują starego. Ten kod:

Listing 1.9: Konwersja łańcucha na małe i wielkie litery

```
print tolower(mystring)
print toupper(mystring)
print mystring
```

...da następujący wynik:

Listing 1.10: Output

```
jak się dzisiaj masz?
JAK SIĘ DZISIAJ MASZ?
Jak się dzisiaj masz?
```

Jak na razie dobrze, ale jak wybrać podłańcuch albo pojedyńczy znak z łańcucha? Do tego celu potrzebna jest funckja substr(), wykorzystywana w ten sposób:

Listing 1.11: Przykład substr()

```
mysub=substr(mystring,startpos,maxlen)
```

mystring powinno być zmienną łańcuchową albo literałem z którego trzeba uzyskać podłańcuch. startpos powinno być pozycją pierwszego znaku, natomiast maxlen powinnno zawierać maksymalną długość uzyskanego łańcucha. Jeżeli lenght(mystring) jest mniejsza niż startpos+maxlen to wynik będzie skrócony. substr() nie modyfikuje oryginalnego łańcucha, jedynie zwraca nowy. Na przykład:

Listing 1.12: Kolejny przykład

```
print substr(mystring,5,4)
```

Awk pokaże:

Listing 1.13: Awk napisze tak

```
się
```

Warto zapamiętać, że substr() jest dla awk substytutem popularnego w innych językach dostępu za pośrednictwem indeksów tablic. Będzie potrzebny do wybierania pojedyńczych znaków oraz podłańcuchów; ponieważ awk jest bazowany na łańcuchach to funkcja ta jest bardzo często używana.

Teraz przejdziemy do ciekawszych funkcji, przy czym pierwszą z nich będzie match(). Jest ona podobna do index() tyle, że zamiast wyszukiwać podłańcuch wykorzystuje wyrażenia regularne. Zwraca pozycję znalezionego podłańcucha lub zero jeżeli wyszukiwanie się nie powiodło. Dodatkowo match() ustawia dwie zmienne, RSTART oraz RLENGTH. RSTART jest taka sama jak zwrócona wartość (pozycja pierwszego znalezionego podciągu pasującego do wyrażenia), natomiast RLENGTH określa jego rozmiar (lub -1 jeżeli nic nie znaleziono). Wykorzystując RSTART, RLENGTH, substr() oraz małą pętlę można łatwo iterować po wszystkich pasujących podciągach w łańcuchu. Poniżej znajduje się przykład wywołania match():

Listing 1.14: Przykładowe wywołanie match()

```
print match(mystring,/się/), RSTART, RLENGTH
```

Awk napisze:

Listing 1.15: Wynik powyższej funkcji

```
5 5 4
```

Podmiana łańcuchów

Teraz opiszę kilka funkcji służących do podmiany łańcuchów, sub() oraz gsub(). Te funkcje różnią się od poprzednich tym, że modyfikują oryginalny łańcuch. Poniżej podany jest przykład użycia sub():

Listing 1.16: Przykład użycia sub()

```
sub(regexp,replstring,mystring)
```

Przy wywołaniu sub() znajdowana jest pierwsza sekwencja znaków w mystring jaka pasuje do regexp, następnie zastępowana jest ona przez replstring. sub() oraz gsub() mają takie same argumenty; jedyną różnicą jest to, że sub() zastąpi tylko pierwsze wystąpienie regexp, natomiast gsub dokona globalnego podstawienia, zamieniając wszystkie dopasowane fragmenty łańcucha. Poniżej znajduje się przykład wykorzystania sub() oraz gsub():

Listing 1.17: Przykładowe wykorzystanie sub() oraz gsub()

```
sub(/i/,"I",mystring)
print mystring
mystring="Jak się dzisiaj masz?"
gsub(/i/,"I",mystring)
print mystring
```

Ustawiliśmy mystring na oryginalną wartość ponieważ pierwsze wywołanie sub() bezpośrednio zmodyfikowało mystring. Po wykonaniu kod ten spowoduje, że awk napisze:

Listing 1.18: Wyjście awk

```
Jak sIę dzisiaj masz?
Jak sIę dzIsIaj masz?
```

Oczywiście są również dopuszczalne bardziej skomplikowane wyrażenia regularne. Przetestowanie ich pozostawione zostaje jako ćwiczenie dla uważnego czytelnika.

Zakończymy opis funkcji obsługujących łańcuchy przez zademonstrowanie funkcji split(). Dzieli ona łańcuch na kawałki i rozmieszcza go w tablicy indeksowanej liczbami. Poniżej znajduje się przykładowe wywołanie split():

Listing 1.19: Przykładowe wywołanie split()

```
numelements=split("sty,lut,mar,kwi,maj,cze,lip,sie,wrz,paź,lis,gru",mymonths,",")
```

Przy wywołaniu split() pierwszy argument zawiera literał lub zmienną przeznaczoną do podzielenia. Następnym argumentem jest nazwa tablicy w której znajdą się wyniki podziału, ostatnim natomiast jest separator pól. Po zakończeniu split() zwraca ilość uzyskanych elementów i umieszcza poszczególne fragmenty w tablicy, zaczynając od indeksu jeden. Tak więc poniższy kod:

Listing 1.20: Przykład

```
print mymonths[1],mymonths[numelements]
```

...da w wyniku:

Listing 1.21: Wynik przykładu

```
sty gru
```

Specjalne formy łańcuchowe

Szybka uwaga -- po wywołaniu length(), sub() lub gsub() można odrzucić ostatni argument a awk zastąpi go $0 (aktualna linia w całości). Można, na przykład, wypisać długości każdego wiersza w pliku, przy użyciu następującego skryptu:

Listing 1.22: Kod wypisujący długość wszystkich wierszy pliku

```
{
    print length()
```

```
}
```

Zabawa finansowa

Kilka tygodni temu postanowiłem napisać własny program do bilansowania wydatków. Zdecydowałem się na wykorzystanie prostego pliku tekstowanego z polami rozdzielanymi tabulacją, w którym mógłbym wpisywać moje wydatki oraz dochody. Założeniem było przekazywanie tego pliku do skryptu awk który automatycznie sumowałby wartości i podawałby mój bilans. Poniżej znajduje się przykładowy wpis w mojej "Książeczce wydatków w ASCII":

Listing 1.23: Książeczka wydatków w ASCII

```
23 sie 2000    jedz    -    -    Y    Bufet Jimmy'ego    30.25
```

Każde pole w tym pliku rozdzielone jest co najmniej jedną tabulacją. Po dacie (pole 1, $1) znajdują się dwa pola, "kategoria wydatków" oraz "kategoria przychodów". Kiedy podaję w powyższej linii wydatki w pole exp wpisuję czteroliterowy skrót, natomiast w następnym polu wpisuję "-" (pusty wpis). W ten sposób oznaczam tę konkretną pozycję jako "wydatek na jedzenie". Z kolei wpłata wygląda tak:

Listing 1.24: Przykładowa wpłata

```
23 sie 2000    -    doch    -    Y    Szefunio        2001.00
```

W tym przypadku umieszczam "-" (puste) w kategorii wydatków oraz "doch" w kategorii dochodów. "doch" jest moim skrótem dla ogólnych dochodów (w rodzaju pensji). Za pomocą skrótów kategorii mogę rozdzielić moje dochody i wydatki na kategorie. Przeznaczenie pozostałych pól powinno być oczywiste. Pole "zatwierdzone?" ("Y" lub "N") zapisuje czy transakcja została przestałane na moje konto. Poza tym zapisany jest opis transakcji oraz dodatnia wartość w dolarach.

Algorytm wykorzystany do wyliczenia aktualnego bilansu nie jest zbyt złożony. AWK po prostu wczytuje kolejne linie. Jeżeli wymieniona jest tylko kategoria wydatków to mamy doczynienia z wypłatą, jeżeli tylko dochodów, to jest to dochód, a jeżeli wymienione są obie kategorie są wymienione, to mamy doczynienia z "transferem między kategoriami"; to znaczy, wymieniona ilość pieniędzy jest odejmowana z kategorii wydatków i dodawana do kategorii dochodów. Wszystkie kategorie są umowne ale nadal bardzo praktyczne do śledzenia wydatków i dochodów oraz do wyliczania budżetu.

Kod

Spójrzmy na poniższy kod. Zacznijmy od pierwszej linii, bloku BEGIN oraz definicji funkcji:

Listing 1.25: Bilans, część pierwsza

```
#!/usr/bin/env awk -f
BEGIN {
    FS="\t+"
    months="sty lut mar kwi maj cze lip sie wrz paz lis gru"
}

function monthdigit(mymonth) {
    return (index(months,mymonth)+3)/4
}
```

Dodanie wiersza z wpisem "#!..." pozwala na uruchamianie skryptu bezpośrednio z wiersza poleceń o ile wcześniej wykonano na nim "chmod +x myscript". Pozostałe wiersze definiują blok BEGIN który zostanie wykonany zanim awk zacznie przetwarzać plik z danymi. Dodatkowo definiujemy łańcuch nazwany months (miesiące) który zostanie wykorzystany przez zdefiniowaną dalej funkcję monthdigit().

Ostatnie trzy wiersze pokazują jak zdefiniować własną funkcję awk. Składnia jest prosta -- należy napisać "function", następnie nazwę funkcji a potem w nawiasach nazwy parametrów rozdzielone przecinkami. W końcu następuje blok kodu zawarty w nawiasach klamrowych, opisujący kod który funkcja powinna wykonywać. Każda funkcja może korzystać ze zmiennych globalnych (takich jak zmienna months). Dodatkowo awk zapewnia instrukcję "return" która pozwala funkcji zwrócić wartość i działa podobnie do swojego odpowiednika w C, Pythonie oraz innych językach. Ta konkretna funkcja przyjmuje 3-literowy skrót nazwy miesiąca i zwraca odpowiednik liczbowy. Na przykład:

Listing 1.26: Przykładowe wywołanie monthdigit()

```
print monthdigit("mar")
```

...zwróci to:

Listing 1.27: Przykładowe wyjście z monthdigit()

```
3
```

Przejdźmy teraz do kolejnych funkcji.

Funkcje finansowe

Kolejne trzy funkcje obsługują naszą księgowość. W głównym bloku następuje sekwencyjne przetwarzanie każdej linii wejścia, w trakcie czego następuje wywołanie jednej z tych funkcji tak, żeby właściwe transakcje zostały zapisane w tablicy awk. Istnieją trzy podstawowe rodzaje transakcji, wpłata (doincome), wypłata (doexpense) oraz transfer (dotransfer). Wszystkie trzy funkcje przyjmują jeden argument, mybalance. Jest to nazwa dwuwymiarowej tablicy w której przechowywane będą wyniki transakcji. Jak do tej pory nie pokazałem tablic wielowymiarowych, są one jednak dość proste -- należy po prostu rozdzileić wymiary przecinkiem.

Informację zapisujemy w "mybalance" jak następuje. Pierwszy wymiar tablicy ma zakres od 0 do 12 i wyznacza miesiąc lub zero dla oznaczenia całego roku. Drugim wymiarem jest czteroliterowy skrót kategorii, na przykład "jedz" lub "doch". Tak więc całoroczny bilans jedzeniowy zawarty jest w mybalance[0,"jedz"]. Przychód w czerwcu znajduje sięw mybalance[6,"doch"].

Listing 1.28: Zapisywanie informacji o przychodach i dochodach

```
function doincome(mybalance) {
    mybalance[curmonth,$3] += amount
    mybalance[0,$3] += amount
}

function doexpense(mybalance) {
    mybalance[curmonth,$2] -= amount
    mybalance[0,$2] -= amount
}

function dotransfer(mybalance) {
    mybalance[0,$2] -= amount
    mybalance[curmonth,$2] -= amount
    mybalance[0,$3] += amount
    mybalance[curmonth,$3] += amount
}
```

Po wywołaniu którekolwiek z powyższych funkcji transakcja zostaje zapisana w dwóch miejscach -- mybalance[0,category] oraz mybalance[curmonth,category], odpowiednio w bilansie całego roku oraz bierzącego miesiąca. Dzięki temu łatwo jest później wykonać rozlicenie roczne lub miesięczne.

Analizując powyższe funkcje można zauważyć, że tablica wymieniona w mybalance jest przekazywana przez referencję. Poza tym odwołujemy się do kilku zmiennych globalnych: curmonth (aktualny miesiąc w postaci liczbowej), $2 (kategoria wydatków), $3 (kategoria dochodów) oraz amount ($7, wartość w dolarach). Przy wywołaniu doincome() oraz jej podobnych wszystkie te zmienne są już właściwie ustawione dla aktualnie przetwarzanego wiersza.

Blok główny

Poniżej znajduje się główny blok kodu który przetwarza każdą linię danych. Ponieważ FS zostało prawidłowo ustawione, możemy do pierwszego pola odwoływać się przez $1, drugiego przez $2 i tak dalej. Przy wywołaniu funkcji doincome() i podobnych dostępne są wartości curmonth, $2, $3 oraz amount.

Listing 1.29: Bilans, część trzecia

```
{
    curmonth=monthdigit(substr($1,4,3))
    amount=$7

    #record all the categories encountered
    if ( $2 != "-" )
        globcat[$2]="yes"
    if ( $3 != "-" )
        globcat[$3]="yes"

    #tally up the transaction properly
    if ( $2 == "-" ) {
        if ( $3 == "-" ) {
```

```
            print "Błąd: pola kategorii zarówno dochodów jak i przychodów są
puste!"
            exit 1
        } else {
            #this is income
            doincome(balance)
            if ( $5 == "Y" )
                doincome(balance2)
        }
    } else if ( $3 == "-" ) {
        #this is an expense
        doexpense(balance)
        if ( $5 == "Y" )
            doexpense(balance2)
    } else {
        #this is a transfer
        dotransfer(balance)
        if ( $5 == "Y" )
            dotransfer(balance2)
    }
}
```

W głównym bloku pierwsze dwa wiersze ustawiają curmonth na liczbę pomiędzy 1 a 12 oraz amount na wartość pola 7 (aby kod był łatwiejszy do zrozumienia). Następnie w czterech wierszach zapisujemy wartości do tablicy globcat. globcat (global categories czyli kategorie globalne) wykorzystana jest do zapisania wszystkich kategorii napotkanych w pliku -- "doch", "różn", "jedz", "narz" i tak dalej. Na przykład, jeżeli $2 == "doch", to ustawiamy globcat["doch"] na "tak". Później możemy iterować po liście kategorii za pomocą prostej pętli "for (x in globcat)".

W kolejnych wierszach analizowana jest zawartość pół $2 oraz $3 po czym transakcja jest odpowiednio zapisywana. Jeżeli $2=="-" a $3!="-" to otrzymaliśmy dochów, więc wywołujemy doincome(). W przypadku odwrotnym wywołujemy doexpense(). Jeżeli zarówno $2 jak i $3 zawierają kategorie to wywołujemy dotransfer. W każdym wypadku przekazujemy do wywołanej funkcji tablicę "balance" więc tam zostaną zapisane informacje.

W kilku miejscach znajduje się również polecenie "if ($5 == "Y") to zapisz tę samą transakcję w "balance2". W polu $5 zapisywano "Y" lub "N", oznaczając przez "Y" transakcje które zostały wysłane na konto. Ponieważ w "balance2" zapisujemy tylko takie transakcje, to balance2 będzie zawierało aktualny bilans konta, natomiast "balance" zapisuje wszystkie transakcje, niezależnie od tego, czy zostały już przesłane. "balance2" można wykorzystać do zweryfikowania wpisanych danych (ponieważ powinien się zgadzaćze stanem konta według banku), natomiast "balance" można wykorzystać, aby uniknąć debetu na koncie (ponieważ bierze pod uwagę wypisane czeki które nie zostały jeszcze zrealizowane).

Generowanie raportu

Po tym jak główny blok przetworzy wszystkie wpisy z wejścia, dysponujemy dość dokładnym zapisem wypłat oraz wpłat, podzielonych na kategorie oraz miesiące. Teraz musimy jeszcze zdefiniować blok END, który wygeneruje raport, w tym przypadku dość skromny:

Listing 1.30: Generowanie raportu końcowego

```
END {
    bal=0
    bal2=0
    for (x in globcat) {
        bal=bal+balance[0,x]
        bal2=bal2+balance2[0,x]
    }
    printf("Dostępne fundusze: %10.2f\n", bal)
    printf("Bilans konta:      %10.2f\n", bal2)
}
```

Ten raport wypisuje podsumowanie które wygląda tak:

Listing 1.31: Ostateczny raport

```
Dostępne fundusze:    1174.22
Bilans konta:         2399.33
```

W bloku END wykorzystaliśmy pętlę "for (x in globcat)" do iterowania przez wszystkie kategorie, zliczając główny bilans oparty na wszystkich zapisanych transakcjach. Ostatecznie wyliczyliśmy obydwa bilansy, dla dostępnych funduszy

oraz bilansu konta. Aby wykonać program oraz przetworzyć własne wydatki zapisane w `mycheckbook.txt`, należy zapisać powyższy kod w pliku `balance`, wykonać `chmod +x balance` a następnie wykonać "`./balance mycheckbook.txt`". Skrypt zsumuje wszystkie transakcje i wypisze dwuwierszowe podsumowanie.

Ulepszenia

Wykorzystuję bardziej zaawansowaną wersję tego programu do zarządzania funduszami osobistymi oraz finansowymi. Moja wersja (której nie mogłem tutaj zamieścić ze względu na ograniczenia rozmiaru) wypisuje rozliczenie miesięczne, uwzględniając obroty, dochód ogólny oraz kilka innych rzeczy. Co więcej, raport zapisany jest w HTML, tak więc mogę go przeglądać w przeglądarce. Zachęcam do ulepszania programu we własnym zakresie. Cała potrzebna informacja jest już zawarta w balance oraz balance2 - trzeba tylko ulepszyć blok END.

Mam nadzieję, że seria spełniła swoje zadanie. Poniżej wyliczone są dodatkowe zasoby, w których można znaleźć więcej informacji o awk.

2. Zasoby

Przydatne odnośniki

- Inne artykuły o Awk autorstwa Daniela na developerWorks: Common threads: Awk by example, Cześć pierwsza orazCzęść druga.
- Książka O'Reilly's sed & awk, 2nd Edition jest doskonałym wyborem dla tych, którzy wolą papier.
- Konieczne należy odwierdzić FAQ comp.lang.awk. Zawiera ono również mnóstwo dodatkowych linków związanych z Awk.
- Samouczek AWK Patricka Hartigana zawiera mnóstwo przydatnych skryptów
- Thompson's TAWK Compiler kompiluje skrypty awk czyniąc z nich szybkie binarki. Jest on dostępny w wersjach dla Windows, OS/2, DOS i UNIX.
- Podręcznik użytkownika GNU Awk zawiera informator.