**Specyfikacje skryptów pisanych w ramach zajęć**

Wskazówki ogólne:

Prosił bym BARDZO o sprawdzenie zgodności swoich skryptów z tą specyfikacją

bo ja sprawdzałem tylko na wyrywki, ale i to nie wszystkie Wasze skrypty były

zgodne z tym co tu jest napisane :-( O wszelkie niejasności w mojej specyfikacji

skryptów czy tu, czy na zajęciach, a tym bardziej o niezgodności pomiędzy tymi

dwoma można i należy pytać czy to na zajęciach, czy emilem do mnie.

Pierwsza linijka KAŻDEGO ze skryptów powinna zawierać właściwe odwołanie do

właściwego interpretera - każde problemy w tej linijce mogą prowadzić do

całkiem zabawnych błędów przy próbie wykonania :-) Wykonanie będzie się odbywało

po nadaniu plikowi atrybutu wykonywalności, pod systemem operacyjnym Linux, ale

w zasadzie wszystkie tu napisane skrypty powinny dać się uruchomić pod MacOSem

czy którymś wcieleniem BSD tak samo, by nie powiedzieć identycznie, identyczne

powinno być także ich działanie :-)

Druga linijka każdego skryptu powinna być komentarzem zawierającym minimum

Wasze imię i nazwisko oraz numer grupy.

Jeśli Państwo nie pakujecie odpowiednio swoich skryptów to oczywiście atrybuty

(w tym atrybut wykonywalności) nie zostaną przekazane do mnie poprawnie, ale

... ja umiem ustawić atrybut wykonywalności u siebie :-)

**==> POWŁOKA - każdy skrypt piszemy w wersji bash i tcsh <==**

Pierwszy skrypt (oczywiście para bash/tcsh):

- wywołany bez żadnych argumentów podaje login name oraz imię i nazwisko

WYWOŁUJĄCEGO (nie autora bo imię i nazwisko autora powinny być zamieszczone

w komentarzu, a jak ktoś chce to także podawane w pomocy, o której poniżej),

pod warunkiem, że umiemy do tej informacji dotrzeć - jak widać po desktopie

ta informacja JEST dostępna, a więc dla chętnych należy sprawdzić jak do

niej dotrzeć :-)

- wywołany z opcją -h/--help podaje krótki opis tego co robi/do czego

służy - patrz wyżej :-) opcja -h ma najwyższy priorytet więc wywołanie z

opcjami -q i -h W DOWOLNEJ KOLEJNOŚCI daje to samo co samo -h czyli podanie

pomocy :-) Ma ona priorytet nadrzędny nawet nad opcjami nieznanymi (patrz

poniżej);

- wywołany z opcją -q/--quiet nie robi nic tj. kończy działanie

- wywołanie z opcją inną niż te dwie daje podanie help'a (jak w punkcie drugim

powyżej), ale przy tym na początku jednolinijkowy komentarz o niezrozumiałej

opcji, a zaraz po podaniu pomocy zakończanie działania z kodem wyjścia 1 - w

obu poprzednich przypadkach kod wyjścia 0 czyli, poprawne wykonanie :-)

Zachowanie takie jest wtedy gdy nie występuje ŻADNA z opcji -q i -h.

- wywołanie z argumentem nie mającym wyglądu opcji jest ignorowane w tym

sensie, że program działa tak jakby był wywołany bez żadnych argumentów czy

opcji i kończy działanie z kodem błędu 0.

Drugi skrypt (a właściwie sześć skryptów):

- pierwsza para drukuje ładnie sformatowaną tabelkę mnożenia od 1 do 9 oraz

ignoruje wszelkie opcje i argumenty;

- druga para drukuje tabelkę mnożenia od pierwszego argumentu do drugiego

argumentu, a więc niemal to samo co poprzedni tylko w sposób kontrolowany

argumentami wywołania. Jeśli argumentów jest mniej niż dwa (znaczy jeden :-)

i jest on numeryczny to traktujemy go jako górną granicę (dolna 1), bez

żadnych argumentów działa jak skrypt poprzedni tj. pokazuje tabelkę mnożenia

od 1 do 9.

Tu musimy sprawdzić czy argumenty są numeryczne i czy są we właściwej

kolejności - jeśli którykolwiek z dwu pierwszych argumentów jest nienumeryczny

to schodzimy z jakimś ładnym komentarzem. Jeśli argumentów jest więcej niż dwa

to pozostałe argumenty ignorujemy :-)

Jeśli pierwsze dwa argumenty są numeryczne to jeśli kolejność jest OK to

drukujemy tabliczkę mnożenia, jeśli niewłaściwa (pierwszy argument większy od

drugiego) to wypisujemy tabelkę od mniejszego do większego.

- trzecia para skryptów robi to samo co druga, ale dla dowolnego działania

akceptowalnego przez powłokę (wydaje się, że conajmniej +-\*/% i potęgowanie -

jakim to operatorem? ale może są inne możliwości operatorów dwuargumentowych?)

i tu sprawdzamy wszystkie trzy pierwsze argumenty:

jeden z nich, dowolnie położony, musi być znanym operatorem, a dwa pozostałe

muszą być liczbami, sprawdzamy oba te postulaty i w razie gdy któryś z nich

nie jest spełniony schodzimy z odpowiednim komunikatem i kodem błędu (1 jeśli

niepoprawny argument liczbowy, 2 jeśli brak poprawnego operatora), jeśli

skrypt znajduje te trzy argumenty to wypisuje tabelkę tego działania dokładnie

w kolejności argumentów, a więc w razie potrzeby od większego do mniejszego :-)

Trzeci skrypt (para bash/tcsh):

- pingujemy WSZYSTKIE numery IP od tego zadanego pierwszym argumentem do

zadanego drugim argumentem - jeśli argumenty są w złej kolejności to

kolejność tę odwracamy - patrz niżej. Pozostałe argumenty, jeśli wystąpią,

są ignorowane. Tu oczywiście musimy sprawdzić, że oba argumenty są poprawnymi

numerami IP - tę funkcjonalność umieszczamy w osobnej FUNKCJI :-)

Wyniki ustawiamy w kolejności od niższego do wyższego niezależnie od kolejności

argumentów. Dla każdego numeru IP chcę parę: numer\_IP żywy/martwy :-) Jak

wygenerować sekwencję jeśli oba numery nie są w tej samej domenie klasy C?

Co z tcsh, w którym (podobno) nie ma funkcji? alias? osobny skrypt wywoływany

z głównego? Na ile skomplikowane było by akceptowanie także adresów w formie

czytelnej dla człowieka jak np. mail.uj.edu.pl ??

UWAGA do skryptów 3 i 4, w bash'u mogę (ale nie muszę) użyć pseudourządzenia

/dev/tcp - pod warunkiem, że mój bash to udostępnia :-) Czego użyć jeśli nie

wiem nic (i nie chcę nic wiedzieć) o /dev/tcp? Czego użyć pod (t)csh?

Czwarty skrypt (para bash/tcsh):

- prosta modyfikacja skryptu trzeciego: dowiadujemy się czy i jakie serwery

odpowiadają na portach określonych jako trzeci argument wywołania, którym jest

rozdzielona przecinkami lista numerycznych portów dla numerów IP określonych

jak poprzednio przez dwa pierwsze argumenty. Za całą odpowiedź dla danego

numeru IP chcę w jednej linijce najkrótszą możliwą wiedzę na temat każdego z

serwerów np.:

149.156.90.10: ssh - OpenSSH 6.0pl1, http - apache 1.3.67, https - httpd 2.0.57

ew. jako dodatkową daną system operacyjny całości :-) Jeśli na danym porcie

coś odpowiada, ale nie umiemy łatwo określić co to piszemy jedynie "otwarty",

dla portów na których nic nie odpowiada piszemy "zamknięty".

W oczywisty sposób serwer może oszukiwać co do swojej tożsamości, ale ...

tę możliwość ignorujemy jako, że na tym etapie nie jest łatwo tego sprawdzić

nie mówiąc o zaradzeniu - no chyba, że ktoś bardzo chce i potrafi :-)

Zwracam uwagę, że niektóre (wszystkie?) serwery nie udzielają żadnych informacji

bez wysłania do nich czegoś przez klienta, to coś musimy sami rozpracować by

dostać od serwera jakąkolwiek rozsądną odpowiedź :-)

Choć skrypt ten bardzo przypomina pewne funkcjonalności nmap'a BARDZO proszę

tegoż NIE używać :-(

Piąty skrypt (jak zwykle para w bash i tcsh):

serwer licznika: przez nc lub /dev/tcp (równoważnik w csh ?) stawiamy na jakimś

(jakim to Wasza sprawa byle dało się go użyć :-) porcie serwer, który serwuje

liczbę swoich wywołań. Serwer musi:

- mieć możliwość konfiguracji portu, na którym odpowiada opcją -p port,

- nie dać się uruchomić NA TYM SAMYM PORCIE w więcej niż jednej kopii - przy

takiej próbie kolejne kopie powinny się wyłączać z ładnym komunikatem,

- pamiętać swój stan pomiędzy uruchomieniami, tj. po ponownym włączeniu nie ma

startować od zera tylko od liczby będącej ostatnim stanem, przy czym należy

rozważyć jak taki stan pamięci pamiętać dla każdego z portów :-)

- dawać się uruchomić także jako swój własny klient, zachowanie regulowane

zarówno opcją wywołania jak i nazwą po której jest wywołany :-)

Umowa co do opcji wywołania: -c - klient, -s - serwer, umowa co do nazw:

serwer.sh i client.sh :-) Sugestia dla niewiedzących: basename/dirname :-)

- położenie "pamięci" między wywołaniami jak i port pracy powinny mieć jakieś

rozsądne defaulty, ale powinny być także konfigurowalne opcjami i z pliku rc,

w kolejności (od najważniejszej): opcja, plik rc, default. Dla chętnych opcja

-f nazwa\_pliku\_rc pozwalająca "nadpisać" wartość defaultową nazwy pliku rc -

przyjmijmy ją wszyscy jako .licznik.rc dla wersji bash'owej oraz .licznik.csh

dla wersji (t)csh - chyba, że umiemy i chcemy napisać skrypty tak by plik

.licznik.rc mógł być wspólny dla obu wcieleń :-)

- opcją -i IP wymuszamy użycie jedynie IP zamiast defaultowego zachowania tj.

użycia wszystkich lokalnych numerów IP :-)

**==> PERL <==**

Skrypt pierwszy: udajemy polecenie cat tj. zawartość plików podanych jako

argumenty wywołania wyrzucany na standardowe wyjście, sprawdzić czy/że

działa także dla plików będących przekierowaniem stdin

Skrypt drugi: cat j.w. ale numeruje linie wyprowadzane na stdout - wersja

prosta, która numeruje linie we wszystkich plikach razem

Skrypt trzeci: cat, ale z wyłączeniem linii zaczynających się od #

Pytanie dodatkowe do skryptów 1-3: czy umieli byśmy to samo zrobić w powłoce?

Jak? Dla dwu pierwszych skryptów sprawdzamy także jak można zrobić to samo z

poziomu powłoki oraz czy/jak zrobić to samo jednym wywołaniem perl'a bez pisania

skryptu - opcja -e 'tu kawałek skryptu' + ew. dodatkowe opcje - poznanie opcji

wywołania samego interpretera :-)

Skrypt czwarty: cat z numerowaniem linii j.w., ale numeruje linie wyprowadzane

na stdout tak, że numeracja nie jest wspólna dla wszystkich plików, ale

w ramach każdego pliku osobno.

Przysyłamy do mnie JEDEN skrypt perlowy sterowany opcjami:

- bez żadnych opcji wyprowadzam na stdout (czy może jednak stderr??) wszystkie

linie wszystkich plików nie numerując ich nijak czyli jak pierwszy ze skryptów

powyżej :-)

-c numeruj wszystkie linie, także te nie pokazywane jeśli użyto także opcji -N

(jak skrypt drugi powyżej),

-N NIE wyprowadzaj na stdout linii zaczynających się od znaku # (jak skrypt

trzeci powyżej),

-n numeruj jedynie linie pokazywane, jak widać wariant opcji -c, co robić

w przypadku połączenia tych opcji w jednym wywołaniu należy do Państwa :-)

-p numeruj linie osobno dla każdego z plików, zaczynając od 1 (jak skrypt

czwarty powyżej).

Umawiamy się na opcje pozycyjnie w tym sensie, że jeśli mają wystąpić to PRZED

wszystkimi nazwami plików do przetworzenia. Opcji może nie być wcale, może być

jedna lub więcej, jak to z opcjami bywa, do trzech ma sens, co do czwartej

to już wyżej napisałem, że problem jest dla Państwa, podobnie jak powtórzenie

którejś z opcji :-) Póki co zastanówmy się co zrobić jeśli pojawi się coś co

wygląda jak opcja, ale nie jest jedną z tych, które opisałem powyżej.

Skrypt piąty: wycinacz pól rozgraniczonych "white space" - wersja pierwsza

bierze jako DWA PIERWSZE ARGUMENTY numer pierwszego i ostatniego słowa do

wypisania i wypisuje cały ten zakres słów łączac słowa pojedynczymi spacjami,

jeśli numeracja jest "większa mniejsza" to nadal wypisujemy w tym zakresie

w numeracji słów takiej jak w pliku, jeśli części czy nawet wszystkich danych

brak to ich nie wypisujemy, a więc w ostateczności wypisujemy pustą linijkę.

Wszystkie pozostałe argumenty traktowane są jako nazwy plików do przetworzenia,

jeśli któryś z plików jest nieczytelny (z dowolnego powodu) wypisujemy na stderr

stosowny komunikat i przechodzimy do kolejnych plików.

Skrypt szósty: jak wyżej, ale dwa pierwsze argumenty są numerami dwu słów do

wypisania, jeśli podam 3 1 to chcę je dostać w takiej kolejności tj. napierw 3,

a po niej 1 :-) Tu jeśli brak którejkolwiek z danych to linijkę ignoruję,

to znaczy nie wypisuje jej, a na stderr wypisuję stosowny komunikat z numerem

linii i nazwą pliku.

UWAGA DO SKRYPTÓW 5 i 6: te dwa numery mają być w obu skryptach dwoma

pierwszymi argumentami, tak by liczba plików do przetworzenia mogła być

zmienna (niekoniecznie jeden plik) i bym można było np. użyć \* :-)

Skrypty te mają być także zabezpieczone na głupie dane wejściowe w tych polach

gdzie to może mieć znaczenie, umawiamy się że zarówno linie jak i słowa

numerujemy od 1 i numery te mają być liczbami całkowitymi bez znaku.

Skrypt siódmy: taki jak 5, ale z rozcięciem na znaku lub ciągu znaków zadanym

jako PIERWSZY ARGUMENT (logicznie może też być drugi lub trzeci, ale dla

prostoty wykonania musimy umówić się, który więc umawiamy się na pierwszy :-)

poza tym proszę o usunięcie we wszystkich wypisywanych polach wiodących i

końcowych białych spacji - to ma znaczenie gdy separator nie jest spacją,

a nie miał w wersji poprzedniej, nieprawdaż?

Skrypt ósmy: podobna modyfikacja skryptu 6

UWAGA do skryptów 5 i 7: jeśli oba numery są takie same wypisuję POJEDYNCZY

obiekt!

UWAGA do skryptów 6 i 8: jeśli oba numery są takie same wypisuję DWA razy ten

same obiekt!

Skrypt dziewiąty: uogólnione wc - dodajemy opcje zliczania liczb: jedna opcja

do zliczania dowolnych liczb (-d), w tym także liczby w dowolnym z moich

formatów wykładniczych postaci: e/E/d/D/q/Q/^ :-) oraz inna (-i) do zliczania

liczb całkowitych. Kolejną dodatkową opcją jest -e, które wyłącza z całego

przetwarzania linie zaczynające się od zanaku #, coś co w powłoce mógł bym

osiągnąć przez: grep -v '^#' plik| wc i\_tu\_opcje\_wc.

Bez podania opcji ma się zachowywać jak systemowe wc, zliczenia podajemy

zarówno per plik (dla każdego z plików z osobna), jak i wszystkich razem - jak

systemowe wc. Jeśli do przetworzenia podany był tylko jeden plik to podsumowania

nie ma (bo i po co :-) Opcje -h, -v oraz -L mogą, ale nie muszą być

implementowane, opcja -m tylko wtedy kiedy umiemy (czyli także nie musi jeśli

ktoś nie umie :-) Jeśli ktoś chce i umie proszę także zaimplementować opcje

długie (--help czy temu podobne :-)

Skrypt piszemy na początek w jednym kawałku (monolityczny), a w następnym

podejściu jako modularny: ćwiczymy stworzenie własnego, prostego modułu i jego

wykorzystanie :-)

Poproszę o umieszczenie w module CONAJMNIEJ sprawdzenia faktu czy coś jest

liczbą całkowitą i czy jest poprawną (w/g mojej specyfikacji) liczbą

rzeczywistą z zapisem wykładnika dziesiętnego w jednej z postaci: e, E, d, D,

q, Q, ^ (odpuszczamy sobie \*\*). Takiego modułu możemy sobie użyć potem

w jakimś innym skrypcie, np. prostym obliczaczu wartości wyrażeń arytmetycznych

dopisując tam regułkę transformacji tych mniej standardowych form do formy

zrozumiałej przez perl'a :-)

Dla chętnych taki rozszerzony moduł i prosty kalkulator jak wyżej :-)

Dla wszystkich do zastanowienia i sprawdzenia: czy bardziej efektywnie jest

zliczać tylko te obiekty, które mamy pokazać, czy też może raczej zliczać

wszystkie możliwe obiekty, a dopiero przy pokazywaniu wyników użyć wiedzy

o tym co należy pokazać? Bo każdy widzi, że pod względem efektywności

mierzonej np. czasem wykonania dla tych samych danych te dwa rozwiązania

NIE są/NIE MUSZĄ być równoważne :-)

Skrypt dziesiąty - dzienniczek: generujemy podsumowanie pliku zawierającego

linie (wiele linii) postaci:

Imię Nazwisko Ocena (tak, pojedyncza ocena w każdej linii!!)

do postaci tabelki z posortowaną alfabetycznie listą:

Nazwisko Imię: Lista ocen: średnia (z dokładnością do dwu cyfr po przecinku).

Jeśli podam kilka argumentów to dostaję tabelkę osobno dla każdego pliku,

tabelki te umieszczane są w plikach o nazwie pliku wejściowego z rozszerzeniem

.oceny, na koniec takiego pliku liczona jest średnia ocena dla całego pliku/

grupy.

Dodatkowe wymagania:

- projekt realizujemy wykorzystując typ danych zwany słownikiem (w tym języku

hash :-)

- w projekcie używamy w/g uznania rozszerzonego modułu z poprzedniego projektu,

w rozszerzeniu tym powinna znaleźć się funkcja, która sprawdza czy coś jest

poprawną oceną i zwraca wartość, której należy użyć do liczenia średniej.

Poprawne oceny to także np. 3+ czy +3 i one znaczą ... o 0.25 w górę, podobnie

jak -4 i 4- są poprawne i znaczą o 0.25 w dół. Ocena 3.3 i temu podobne są

także poprawne i jako takie maję być wliczone do średniej :-)

Przypominam, że zarówno imiona jak i nazwiska mogą być wpisane dowolną

kombinacją dużych i małych liter, ale zarówno w sensie "klucz w słowniku"

jak i późniejsze wypisanie chcę mieć postać ogólnie przyjętą w polskiej

konwencji tj. pierwsze litery i imienia i nazwiska od dużej litery, ale reszta

już małymi.

Nasz skrypt ma działać poprawnie dla wszystkich poprawnych linijek pliku

wejściowego, a linijki niepoprawne ma wyrzucać na stderr, zaś jako dodatkowe

zadanie, dla chętnych, staramy się uodpornić nasz skrypt na najprostrzy błąd

w pliku wejściowym czyli rozdzielenie poprawnej linijki na dwie np. po

imieniu lub po nazwisku.

**==> PYTHON <==**

Skrypt pierwszy w pythonie: taki jak pierwsze cztery, a właściwie sam czwarty,

w perlu, specyfikacja ta sama, działanie to samo, dodatkowo sprawdzamy, na

jakichś kilku dużych plikach na raz, który z języków jest bardziej efektywny

tj. w którym z nich dla NA PRAWDĘ dużej liczby linii skrypty nasze wykonają

się szybciej :-) Poproszę o jeden skrypt, ten w perlu oznaczony jako czwarty

oraz wyniki testu szybkości oraz podsumowanie o czym wyniki te świadczą :-)

Dla chętnych: sprawdzić różnice między skryptami pod 2.7 i 3 (jakakolwiek

by ta ostatnia była :-)

Skrypt drugi: jak wc pisane w perlu (specyfikacja jak dla skryptu dziewiątego

w perlu), piszemy skrypt monolityczny (w jednym pliku) i testujemy go tak jak

pierwszy skrypt pythonowy tj. sprawdzamy, która z dwu naszych wersji (perlowa

czy pythonowa) jest szybsza i na ile :-)

Dla chętnych: sprawdzić różnice między skryptami pod 2.7 i 3 (jakakolwiek

by ta ostatnia była :-)

Skrypt trzeci: dzienniczek w/g specyfikacji dla wersji perlowej,

wersja modularna lub monolityczna, jak Państwu wygodniej, poproszę o sam

skrypt oraz wyniki czasu wykonania wersji pythonowej i perlowej dla jakiegoś

naprawdę dużego pliku z danymi, powiedzmy kilka-dziesiąt/kilka-set tysięcy

linii :-) Jakie możemy wyciągnąć wnioski na temat efektywności słowników

w obu językach?

Skrypt czwarty: przeszukiwacz kartotek: mogę podać jedną (lub więcej)

kartotekę do przeszukania (w dół/głąb) z wyszukaniem we wszystkich plikach

znajdujących się wewnątrz kartoteki fraz zadanych argumentami nie będącymi

kartotekami. Kartoteki zadajemy opcjami -d, po każdym -d DOKŁADNIE jedna

kartoteka, jeśli chcemy ich więcej to podajemy więcej razy opcję -d z

następną nazwą, wszystko co nie jest bezpośrednio po -d, czyli nie jest

kartoteką, jest frazą do wyszkukania. Jeśli po kartotece jest następne -d

to słowa do wyszukania są po ostatniej z nich, ale nie są wspólne dla

wszystkich więc wywołanie postaci:

./szukacz.py -d ~/lib -d ~/scripts pier -d ~/bin dru

szuka słów z pier w kartotekach ~/lib i ~/scripts, a słów z dru w ~/bin.

Podawane są wyłącznie statystyki/liczby słów zawierającyh każdą z fraz per

kartoteka zgłoszona jako argument :-) Fraza może być albo całym słowem, albo

jego częścią, frazy zawierające spacje itp. zależy podawać w cudzysłowach.

Liczba zliczeń wzorca w tekście powinna być zgodna z wynikami dawanymi

w danym przypadku przez systemowy grep. Na skrypcie tym ćwiczymy także

definiowanie własnego modułu, w którym kodujemy część funkcjonalności tego

zadania, którą część zostawiam Państwu :-) Ale mogę się także zgodzić na

program monolityczny jeśli tylko działa poprawnie :-)

Przeglądając kartoteki wgłąb nie podążamy za dowiązaniami miękkimi, ale

w ramach ćwiczeń można sobie dorobić jakąś opcję, która to umożliwi :-)

A jak będzie z dowiązaniami twardymi - czy będę liczył wielokrotnie te same

wzorce jeśli są w plikach będących dowiązaniami twardymi, czy też nie? Czy

da się to zmienić?

Nieoobowiązkowa opcja -v, która WŁĄCZA standardowo wyłączone wypisywanie linii

ze znalezioną frazą (wraz z nazwą pliku). Opcja jest nieobowiązkowa w tym

sensie, że implementuje ją ten kto chce :-) Opcja ta dotyczy każdej z sekcji:

-d dir1 -d dir2 ... pat1 pat2 ...patn

Ten skrypt piszemy także w perlu i powłoce i testujemy ich efektywność na

różnych dużych kartotekach :-) Poza samymi skryptami chciałbym od każdego kto

to zrobi dostać wiadomość o porównaniu efektywności każdego z rozwiązań

zmierzonej np. systemowym poleceniem time, albo czymś równoważnym dla tych

samych argumentów wywołania :-) I bardzo proszę by wersja perlowa i pythonowa

wykorzystywała jedynie mechanizmy tegoż języka, a nie powtarzała tego co

napisano w wersji powłokowej bo wtedy testowanie efektywności traci cały

sens :-)