LABORATORIUM SOP
'BASH'

PARAMETRY zadania

1. Napisz program **zad1.sh** wyświetlający:
   a) jego nazwę
   b) kolejno wartości jego argumentów
   c) wszystkie argumenty
   d) liczbę podanych argumentów
   e) ID procesu

PRZYKŁAD:
XXXXX@szuflandia:~$ bash zad1.sh a b
program nazywa się: zad1.sh
param1: a
param2: b
param: a b*
param#: 2
proc PID: 25323

2. Napisz program **zad2.sh** który będzie działał jak zad1.sh obsługując dowolną liczbę parametrów

PRZYKŁAD:
XXXXX@szuflandia:~$ bash zad2.sh a b c d e f g h i j k
program nazywa sie: zad2.sh
param 1: a
param 2: b
param 3: c
param 4: d
param 5: e
param 6: f
param 7: g
param 8: h
param 9: i
param 10: j
param 11: k
param: a b c d e f g h i j k*
param#: 11
proc PID: 25323

PLIKI I FOLDERY zadania

3. Napisz program **zad3.sh** który sprawdzi czy w bieżącym katalogu istnieje plik podany jako argument wywołania.

PRZYKŁAD:
XXXXX@szuflandia:~$ ./zad2.sh maile.txt
W biezacym folderze jest plik maile.txt

XXXXX @szuflandia:~$ ./zad2.sh mailer.txt
W biezacym folderze nie ma mailer.txt

4. Napisz program **zad4.sh** który będzie wyświetlał menu wyboru oraz w zależności od wyboru:
   c – aktualny katalog
   u - nazwę użytkownika
   h - katalog domowy

warianty:
zad41.sh – zastosować własne funkcje menu oraz wybor
zad4s.sh –funkcje menu oraz wybor czytać z własnej biblioteki funkcje.f

POMOCE NAUKOWE

http://szuflandia.pjwstk.edu.pl/~toli/SOP_2020_02/SOP_BASH1.pdf
http://dief.republika.pl/read.html (link może nie działać, dlatego przekład poniżej)

Polecenie read ()

Czyta ze standardowego wejścia pojedynczy wiersz.

Składnia:

```
read -opcje nazwa_zmiennej
```

Przykład:

```
#!/bin/bash
echo -n "Wpisz coś:\a"

read wpis
echo "$wpis"
```

To co zostało wpisane trafi do zmiennej **wpis**, której to wartość czyta polecenie **read wpis**, zmienna nie musi być wcześniej tworzona, jeśli istniała wcześniej, jej zawartość zostanie zastąpiona tym co wpisaliśmy.

Przykład:

```
#!/bin/bash
echo "Wpisz coś:"

answer="napis"
read
echo "$answer"
```

Wcześniejsza wartość zmiennej **answer** została zastąpiona.

Polecenie **read** pozwala na przypisanie kilku wartości kilku zmiennym.

Przykład:

```
#!/bin/bash
echo "Wpisz cztery wartości:"

read a b c
echo "Wartość zmiennej a to: $a"
echo "Wartość zmiennej b to: $b"
echo "Wartość zmiennej c to: $c"
```

Nie przypadkiem w powyższym przykładzie pojawiło się polecenie wpisania czterech wartości, pierwsza wartość trafi do zmiennej **a**, druga do zmiennej **b**, natomiast trzecia i czwarta oraz rozdzielające je znaki separacji przypisane zostaną zmiennej **c**.

Wybrane opcje:

- **-p**

  Pokaże znak zachęty bez kończącego znaku nowej linii.

  ```
  #!/bin/bash

  read -p "Pisz:" odp
  echo "$odp"
  ```

- **-a**

  Kolejne wartości przypisywane są do kolejnych indeksów zmiennej tablicowej.

Przykład:

```
#!/bin/bash
echo "Podaj elementy zmiennej tablicowej:"

read tablica
echo "${tablica[*]}"
```

- **-e**

  Jeśli nie podano żadnej nazwy zmienej, wiersz trafia do **$REPLY**.

  Przykład:

```
#!/bin/bash
echo "Wpisz coś:"

read -e
echo "$REPLY"
```

Funkcje

Coś w rodzaju podprogramów. Stosuje się je gdy w naszym skrypcie powtarza się jakaś grupa poleceń, po co pisać je kilka razy, skoro można to wszystko umieścić w funkcjach. Do danej funkcji odwołujemy się podając jej nazwę, a wykonane zostanie wszystko co wpisaliśmy między nawiasy **{ }**, skraca to znacznie długość skryptu.

Składnia:

```
function nazwa_funkcji
{
polecenie1
polecenie2
polecenie3
}
```

Przykład:

```
#!/bin/bash

function napis
{
echo "To jest napis"
}

napis
```

Nazwę funkcji umieszczamy po słowie kluczowym **function**, w powyższym przykładzie mamy funkcje o nazwie **napis**, odwołujemy się do niej podając jej nazwę, wykonane zostaną wtedy wszystkie polecenia, jakie jej przypiszemy.
Funkcje moga się znajdować w innym pliku, co uczyni nasz skrypt bardziej przejrzystym i wygodnym, tworzy się własne pliki nagłówkowe, wywołuje się je tak:

```
. ~/naszplik_z_funkcjami
nazwa funkcji
```

Trzeba pamiętać o podaniu kropki + spacja przed nazwą pliku

Przykład:

```
#!/bin/bash
function nasza_funkcja
{
echo -e 'Właśnie użyłeś funkcji o nazwie "nasza_funkcja".\a'
}
```

Teraz pozostało jeszcze utworzyć skrypt w którym wywołamy funkcje: **nasza_funkcja**:

```
#!/bin/bash
echo "Test funkcji."
. funkcja
nasza_funkcja
```

Instrukcja warunkowa if

Sprawdza czy warunek jest prawdziwy, jeśli tak to wykonane zostanie polecenie lub polecenia znajdujące się po słowie kluczowym **then**. Instrukcja kończy się słowem **fi**.

Składnia:

```
if warunek
then
  polecenie
fi
```

Przykład:

```
#!/bin/bash
if [ -e ~/.bashrc ]
then
  echo "Masz plik .bashrc"
fi
```

Najpierw sprawdzany jest warunek: czy istnieje w twoim katalogu domowym plik **.bashrc**, zapis ~/ oznacza to samo co **/home/twój_login** lub **$HOME**. Jeśli sprawdzany warunek jest prawdziwy to wyświetlony zostanie napis **Masz plik .bashrc**. W przeciwnym wypadku nic się nie stanie.

W sytuacji gdy test warunku zakończy się wynikiem negatywnym można wykonać inny zestaw poleceń, które umieszczamy po słowie kluczowym **else**:

Składnia:

```
if warunek
then
  polecenie1
else
  polecenie2
fi
```

Przykład:

```
#!/bin/bash
if [ -e ~/.bashrc ]
then
  echo "Masz plik.bashrc"
else
  echo "Nie masz pliku .bashrc"
fi
```

Jeśli warunek jest fałszywy skrypt poinformuje Cię o tym.

Można też testować dowolną ilość warunków, jeśli pierwszy warunek nie będzie prawdziwy, sprawdzony zostanie następny, kolejne testy warunków umieszczamy po słowie kluczowym **elif**.

Składnia:

```
if warunek
then
  polecenie1
elif warunek
  polecenie2
fi
```

Przykład:

```
#!/bin/bash
if [ -x  /opt/kde/bin/startkde ]
then
  echo "Masz KDE w katalogu /opt"
elif [ -x /usr/bin/startkde ]
  echo "Masz KDE w katalogu /usr"
elif [ -x /usr/local/bin/startkde ]
  echo "Masz KDE w katalogu /usr/local"
```

```
else
  echo "Nie wiem gdzie masz KDE"
fi
```
Ten skrypt sprawdza gdzie masz zainstalowane **KDE**, sprawdzane są trzy warunki, najpierw czy plik wykonywalny **startkde** znajduje się w katalogu **/opt/kde/bin** jeśli go tam nie ma, szukany jest w **/usr/bin**, gdy i tu nie występuje sprawdzany jest katalog **/usr/local/bin**.

Instrukcja case

Pozwala na dokonanie wyboru spośród kilku wzorców. Najpierw sprawdzana jest wartość zmiennej po słowie kluczowym **case** i porównywana ze wszystkimi wariantami po kolei. Oczywiście musi być taka sama jak wzorzec do którego chcemy się odwołać. Jesli dopasowanie zakończy się sukcesem wykonane zostanie polecenie lub polecenia przypisane do danego wzorca. W przeciwnym wypadku użyte zostanie polecenie domyślne oznaczone symbolem gwiazdki: ***) polecenie_domyślne**. Co jest dobrym zabezpieczeniem na wypadek błędów popełnionych przez użytkownika naszego skrytpu.

Składnia:

```
case zmienna in
  "wzorzec1") polecenie1 ;;
  "wzorzec2") polecenie2 ;;
  "wzorzec3") polecenie3 ;;
  *) polecenie_domyślne
esac
```

Przykład:

```
#!/bin/bash
echo "Podaj cyfrę dnia tygodnia"
read d
case "$d" in
  "1") echo "Poniedziałek" ;;
  "2") echo "Wtorek" ;;
  "3") echo "Środa" ;;
  "4") echo "Czwartek" ;;
  "5") echo "Piątek" ;;
  "6") echo "Sobota" ;;
  "7") echo "Niedziela" ;;
  *) echo "Nic nie wybrałeś"
esac
```
Jak widać mamy w skrypcie wzorce od **1** do **7** odpowiadające liczbie dni tygodnia, każdemu przypisane jest jakieś polecenie, tutaj ma wydrukować na ekranie nazwę dnia tygodnia. Jeśli podamy 1 polecenie **read** czytające dane ze standardowego wejścia przypisze zmiennej **d** wartość 1 i zostanie wykonany skok do wzorca **1**, na ekranie zostanie wyświetlony napis *Poniedziałek*. W przypadku gdy podamy cyfrę o liczbie większej niż **7** lub wpiszemy inny znak na przykład literę to wykonany zostanie wariant defaultowy oznaczony gwiazdką: ***) echo "Nic nie wybrałeś"**.

Pętla for

Wykonuje polecenia zawarte wewnątrz pętli, na każdym składniku listy (iteracja).

Składnia:

```
for zmienna in lista
do
  polecenie
done
```
Przykład:

```
for x in jeden dwa trzy
do
  echo "To jest $x"
done
```

Zmiennej **x** przypisana jest lista, która składa się z trzech elementów: **jeden**, **dwa**, **trzy**. Wartością zmiennej **x** staje się po kolei każdy element listy, na wszystkich wykonywane jest polecenie: **echo "To jest $x"**. Pętla for jest bardzo przydatna w sytuacjach, gdy chcemy wykonać jakąś operację na wszystkich plikach w danym katalogu. Na przykład chcemy uzyskać listę wszystkich plików o danym rozszerzeniu znajdujących się w jakimś katalogu, robimy to tak:

```
#!/bin/bash
for x in *html
do
  echo "To jest plik $x"
done
```

lub jeśli chcemy zmienić nazwy plików pisane *DUŻYMI* literami na nazwy pisane *małymi* literami:

```
#!/bin/bash
for nazwa in *
do
  mv $nazwa `echo $nazwa tr '[A-Z]' '[a-z]'`
done
```

Za zmianę *DUŻYCH* liter na *małe* (i na odwrót) odpowiedzialne jest polecenie **tr**.

Pętla select

Wygeneruje z listy słów po **in** proste ponumerowane menu, każdej pozycji odpowiada kolejna liczba od 1 wzwyż. Poniżej menu znajduje się znak zachęty PS3 gdzie wpisujemy cyfrę odpowiadająca wybranej przez nas pozycji w menu. Jeśli nic nie wpiszemy i wciśniemy ENTER, menu będzie wyświetlone ponownie. To co wpisaliśmy zachowywane jest w zmiennej **REPLY.** Gdy odczytane zostaje **EOF** (ang. *End Of File*) czyli znak końca pliku **(CTRL+D)** to select kończy pracę. Pętla działa dotąd dopóki nie wykonane zostaje polecenie **break** lub **return**.

Składnia:

```
select zmienna in lista
do
  polecenie
done
```

Od razu nasuwa się możliwość zastosowania wewnątrz niej instrukcji case:

```
#!/bin/bash
echo "Co wybierasz?"
select y in X Y Z Quit
do
  case $y in
    "X") echo "Wybrałeś X" ;;
    "Y") echo "Wybrałeś Y" ;;
    "Z") echo "Wybrałeś Z" ;;
    "Quit") exit ;;
    *) echo "Nic nie wybrałeś"
  esac
break
done
```

Najpierw zobaczymy proste ponumerowane menu, składające się z czterech elementów: **X**, **Y**, **Z** i **Quit**, teraz wystarczy tylko wpisać numer inetersującej nas opcji, a resztę zrobi instrukcja case. Polecenie **break**, które znajduje się w przedostatniej linii skryptu, kończy pracę pętli.
A teraz bardziej praktyczny przykład, poniższy skrypt **(Uwaga!)** przeznaczony dla dystrybucji **Slackware** wygeneruje menu składające się z listy Window Mangerów, po wybraniu konkretnej pozycji uruchomiony zostanie dany WM. Oczywiście należy skrypt zmodyfikować pod kątem własnego systemu. Jeśli komuś odpowiada takie rozwiązanie, wystarczy utworzyć

alias: **alias startx="~/.ten_skrypt"** i po ponownym zalogowaniu mamy po wpisaniu polecenia **startx** menu wyboru Window Managerów.

```
#!/bin/bash
echo ""
echo "[ JAKI WINDOW MANAGER URUCHOMIĆ? WYBIERZ CYFRĘ Z LISTY: ]"
echo ""
select l in BLACKBOX ENLIGHTENMENT GNOME ICEWM KDE MWM OPENWIN TWM WMAKER
WYJŚCIE
do
  case "$l" in
    "BLACKBOX") cat /etc/X11/xinit/xinitrc.blackbox > ~/.xinitrc; startx $@
;;
    "ENLIGHTENMENT") cat /etc/X11/xinit/xinitrc.e > ~/.xinitrc; startx $@
;;
    "GNOME") cat /etc/X11/xinit/xinitrc.gnome > ~/.xinitrc; startx $@ ;;
    "ICEWM") cat /etc/X11/xinit/xinitrc.icewm > ~/.xinitrc; startx $@ ;;
    "KDE") cat /etc/X11/xinit/xinitrc.kde > ~/.xinitrc; startx $@ ;;
    "MWM") cat /etc/X11/xinit/xinitrc.mwm > ~/.xinitrc; startx $@ ;;
    "OPENWIN") cat /etc/X11/xinit/xinitrc.openwin > ~/.xinitrc; startx $@
;;
    "TWM") cat /etc/X11/xinit/xinitrc.twm > ~/.xinitrc; startx $@ ;;
    "WMAKER") cat /etc/X11/xinit/xinitrc.wmaker > ~/.xinitrc; startx $@ ;;
    "WYJŚCIE") exit ;;
    *) startx $@
  esac
break
done
```

Elementy składowe listy w pętli **select**, noszą takie same nazwy jak wzorce w instrukcji **case** co umożliwia wykonanie skoku do danego wzorca i wykonania poleceń jemu przypisanych. Jak to wygląda w praktyce? Na przykład chcemy uruchomić **KDE**, wybieramy więc z menu opcje o wyżej wymienionej nazwie, następnie polecenie **cat** nadpisuje nasz domowy plik **.xinitrc**, kopiując do niego zawartość pliku **xinitrc** zoptymalizowanego dla **KDE**, znajdującego się w katalogu: **/etc/X11/xinit/xinitrc.kde**, po czym wykonywane jest polecenie **startx**. Zmienna **$@** to zmienna specjalna umożliwiająca przekazywanie do skryptu parametrów (**startx** to też skrypt powłoki), dzięki czemu możemy spokojnie stosować wszelkie parametry np. dwukrotne odpalenie **X**-ów: **startx -- :0** na pierwszej konsoli i **startx -- :1** na drugiej. Gdy nie wpiszemy żadnych parametrów **$@** jest pusta. A co się stanie w przypadku gdy podczas wyboru Window Managera podamy większą cyfrę niż tą jaką ma ostatni element menu lub jakiś inny znak? Uruchomiony zostanie ten WM, który ostatnio odpalaliśmy.

Ten sam skrypt przeznaczony dla dystrybucji **Red Hat** W tym przypadku polecenie: **echo "exec window_manager"** nadpisuje plik **.XClients** znajdujący się w naszym katalogu **home**. Z aliasem postępujemy analogicznie jak w powyższym przykładzie.

```
#!/bin/bash
echo ""
echo "[ JAKI WINDOW MANAGER URUCHOMIĆ? WYBIERZ CYFRĘ Z LISTY: ]"
echo ""
select l in BLACKBOX ENLIGHTENMENT GNOME ICEWM KDE MWM OPENWIN TWM WMAKER
WYJŚCIE
do
  case "$l" in
    "BLACKBOX") echo "exec blackbox" > ~/.XClients; startx $@ ;;
    "ENLIGHTENMENT") echo "exec enligtenment" > ~/.XClients; startx $@ ;;
    "GNOME") echo "exec gnome-session" > ~/.XClients; startx $@ ;;
    "ICEWM") echo "exec icewm" > ~/.XClients; startx $@ ;;
    "KDE") echo "exec startkde" > ~/.XClients; startx $@ ;;
    "MWM") echo "exec mwm" > ~/.XClients; startx $@ ;;
    "OPENWIN") echo "exec openwin" > ~/.XClients; startx $@ ;;
    "TWM") echo "exec twm" > ~/.XClients; startx $@ ;;
    "WMAKER") echo "exec wmaker" > ~/.XClients; startx $@ ;;
    "WYJŚCIE") exit ;;
    *) startx $@
```

```
  esac
break
done
```

Pętla while

Najpierw sprawdza warunek czy jest prawdziwy, jeśli tak to wykonane zostanie polecenie lub lista poleceń zawartych wewnątrz pętli, gdy warunek stanie się fałszywy pętla zostanie zakończona.

Składnia:

```
while warunek
do
  polecenie
done
```

Przykład:

```
#!/bin/bash
x=1;
while [ $x -le 10 ]; do
  echo "Napis pojawił się po raz: $x"
  x=$[x + 1]
done
```

Sprawdzany jest warunek czy zmienna **x** o wartości początkowej 1 jest mniejsza lub równa 10, warunek jest prawdziwy w związku z czym wykonywane są polecenia zawarte wewnątrz pętli: **echo "Napis pojawił się po raz: $x"** oraz **x=$[x + 1]**, które zwiększa wartość zmiennej **x** o 1. Gdy wartość **x** przekroczy 10, wykonanie pętli zostanie przerwane.

Pętla until

Sprawdza czy warunek jest prawdziwy, gdy jest fałszywy wykonywane jest polecenie lub lista poleceń zawartych wewnątrz pętli, miedzy słowami kluczowymi **do** a **done**. Pętla **until** kończy swoje działanie w momencie gdy warunek stanie się prawdziwy.

Składnia:

```
until warunek
do
  polecenie
done
```

Przykład:

```
#!/bin/bash
x=1;
```