Systemy Operacyjne

Wejściówka 3

1. Pobieranie danych od użytkownika:

```
#!/bin/bash

#pierwszy sposób
read -p "Kotek robi miał? " dane1

#drugi sposób
echo "Czy kotki są fajne: "
read dane2
```

2. Argumenty i ich wyświetlanie:
    - **$#** - liczba argumentów pliku
    - **$1** … **$2** … **$3** – argumenty podane przez użyszkodnika
    - **$@ -** przechowuje listę argumentów jako tablice
    - **$*** - przechowuje wszystkie argumenty wiersza poleceń, łącząc je ze sobą
    - **${tablica[0]}** – wyświetlanie pierwszego argumentu z tablicy
    - **${#tablica[*]}** … **${#tablica[@]}** – ilość argumentów w tablicy
    - **${tablica[*]}** … **${tablica[@]}** – wszystkie argumenty w tablicy

3. Generowanie randomowej liczby:
    - **zmienna=$((RANDOM%100))** – generowanie zmiennej z przedziału 0 - 99
4. Operacje matematyczne:
    - **let**

```
#!/bin/bash

liczba1=5
liczba2=5

let wynik[0]=liczba1+liczba2
let wynik[1]=liczba1-liczba2
let wynik[2]=liczba1/liczba2
let wynik[3]=liczba1*liczba2

echo ${wynik[*]}
```

    - **expr**

```
#!/bin/bash

liczba1=1
liczba2=3
#uwaga używamy tego na tyldzie `
wynik[0]=`expr $liczba1 + $liczba2`
wynik[1]=`expr $liczba1 - $liczba2`
wynik[2]=`expr $liczba1 / $liczba2`
wynik[3]=`expr $liczba1 \* $liczba2` #uwaga - mnożenie

echo ${wynik[*]}
```

    - **$**

```
#!/bin/bash

liczba1=5
liczba2=5
```

```
5.
6.  wynik[0]=$(($liczba1+$liczba2))
7.  wynik[1]=$(($liczba1-$liczba2))
8.  wynik[2]=$((liczba1*$liczba2))
9.  wynik[3]=$(($liczba1/$liczba2))
10. wynik[4]=$(($liczba1%$liczba2))
11. wynik[5]=$(($liczba1==$liczba2))
12. wynik[6]=$(($liczba1!=$liczba2))
13. wynik[7]=$((${wynik[0]}+10))
14.
15. echo ${wynik[*]}
```

5. Instrukcja warunkowa **if**:
    - Dostępne operatory porównania:
        - **-eq** [$zmienna1 -eq $zmienna2] porównania ==
        - **-ne** [$zmienna1 -ne $zmienna2] różny !=
        - **-gt** [$zmienna1 -gt $zmienna2] większości >
        - **-lt** [$zmienna1 -lt $zmienna2] mniejszości <
        - **-ge** [$zmienna1 -ge $zmienna2] większy lub równy >=
        - **-le** [$zmienna1 -le $zmienna2] mniejszy lub równy <=
        - **-o** (||) [$zmienna1 -lt $zmienna2 -o $zmienna1 -gt $zmienna2] operator lub
        - **-a** (&&)[$zmienna1 -lt $zmienna2 -a $zmienna1 -gt $zmienna2] operator i
    - Dostępne operatory „sprawdzenia czy”:
        - **-b plik** Wykrywanie, czy plik jest plikiem urządzenia blokowego, a jeśli tak, to zwraca true
        - **-c plik** Wykrywanie, czy plik jest plikiem urządzenia znakowego, a jeśli tak, to zwraca true
        - **-d plik** Wykrywanie, czy plik jest katalogiem, a jeśli tak, to zwraca true
        - **-f plik** Wykrywanie, czy plik jest zwykłym plikiem, a jeśli tak, to zwraca true
        - **-g plik** Wykrywanie, czy plik ma nieco SGID, a jeśli tak, to zwraca true
        - **-k plik** Wykrywanie, czy plik ma lepką Sticky bit (bit), a jeśli tak, to zwraca true
        - **-p plik** Wykrywanie, czy plik jest nazwany potok, a jeśli tak, to zwraca true
        - **-u plik** Wykrywanie, czy plik ma bit SUID, a jeśli tak, to zwraca true
        - **-r plik** Wykrywa, czy plik jest czytelny, a jeśli tak, to zwraca true
        - **-w plik** Wykrywanie, czy plik można zapisać, a jeśli tak, to zwraca true
        - **-x plik** Wykrywanie, czy plik jest wykonywalny, a jeśli tak, to zwraca true
        - **-s plik** Wykrywanie, czy plik jest pusty (rozmiar pliku jest większy niż 0), nie jest pusta return true
        - **-e plik** Wykrywanie pliku (w tym katalogu) istnieje, a jeśli tak, to zwraca true

    - Składnia if-a (suchy przykład):

```
1.  if [ warunek ]
2.  then
3.      < blok kodu >
4.  elif [ warunek ]
5.  then
6.      < blok kodu >
7.  else
8.      < blok kodu >
```

```
fi
```

6. Instrukcja **case**:
    - Składnia case (suchy przykład):

```
case wartosc in
wzor_1 )
    < blok kodu >
;;
wzor_2 | wzor_3 | wzor_4 )
    < blok kodu >
;;
*)
    < blok kodu >
;;
esac
```

7. Pętla **for**:
    - Przeskok po zakresie (suchy przykład)::

```
for wartosc in zakres
do
    < blok kodu >
done
```

    - Taka zwykła pętla (suchy przykład):

```
for (( wartosc_poczatkowa; warunek_konca; operacja_na_liczniku ))
do
    < blok kodu >
done
```

8. Pętla **while**:
    - Składnia while (suchy przykład):

```
while [ warunek ]
do
    < blok kodu >
done
```

9. Funkcje, użycie i suchy przykład:

```
nazwa_funkcji (){
    < blok kodu >
    #gdy podajemy zmienne to traktujemy je jak w nowym skrypcie ($1 $2)
}

#wywołanie
nazwa_funkcji

#wywołanie z przekazaniem argumentów
```

```
10. nazwa_funkcji $zmienna $inna_jebana_zmienna
```

10. Prawa dostępu **chmod** (powtórka):

		Pliki	**Katalogi**
Read (odczyt)	r	Pozwala na odczyt pliku i kopiowanie	Pozwala na wyświetlenie zawartości folderu przez komendę ls
Write (zapis)	w	Pozwala na modyfikacje zawartości pliku	Pozwala na utworzenie, wykasowanie lub zmianę nazwy pliku i podkatalogu
Execute (wykonanie)	X	Pozwala na wykonanie pliku	Pozwala na wejście do katalogu

Add (+)	Dodaje uprawnienie
Revoke (-)	Odbiera uprawnienie
Assign (=)	Przypisuje tylko to uprawnienie co wpisujemy

User (u)	Właściciel pliku lub katalogu
Group (g)	Zbiór użytkowników, którzy potrzebują takiego samego dostępu do plików i katalogów, które współdzielą. Informacje o grupach znajdują się w pliku /etc/group, a użytkownicy są przypisani do tych grup
Others (o)	Wszyscy inni użytkownicy systemu (poza właścicielem pliku lub katalogu i członkami grupy, do której należy plik czy katalog)
All (a)	Wszyscy

r	4
w	2
x	1

0	– – –	Brak uprawnień
1	– – x	Wykonywanie
2	– w –	Zapis
3	– w x	Zapis i wykonywanie
4	r – –	Odczyt
5	r – x	Odczyt i wykonywanie
6	r w –	Odczyt i zapis
7	r w x	Odczyt, zapis, wykonywanie

Symbole typów elementów:

- - – zwykły plik
- b – specjalny plik blokowy
- c – specjalny plik znakowy
- d – katalog
- l – dowiązanie symboliczne
- p – nazwany potok
- s – gniazdo

Przykłady użycia:

- chmod 644 plik (-rw-r--r--)
- chmod 744 plik (-rwxr--r--)
- chmod 700 plik (-rwx------)
- chmod u+x plik
- chmod a=rwx (-rwxrwxrwx)
-

11. Instrukcja warunkowa w praktyce:

```
1.  #!/bin/bash
2.
3.  if [ $# -eq 2 ];
4.    then
5.    if [ $1 -gt $2 ];
6.                        then
7.              echo "$1 jest wieksze niz $2."
8.    else
9.              echo "$2 jest wieksze niz $1."
10.   fi
11.   if [ $1 -lt $2 ];
12.                       then
13.                       echo "$1 jest mniejsze niz $2."
14.   else
15.             echo "$2 jest mniejsze niz $1."
16.   fi
17.   if [ $1 -eq $2 ];
18.                       then
19.                       echo "$1 jest rowne $2."
20.   fi
21.   if [ $1 -ne $2 ];
22.                       then
23.                       echo "$1 nie jest rowne $2."
24.             fi
25. else
26.   echo "Podano niewlasciwa liczbe arguemntow"
27. fi
```

12. Pętla for, przeskok po zakresie w przykładzie:

```
#!/bin/bash

#dzieli tekst i wyświetla pojedyncze słowa
tekst="Tekst wyswietlany w petli for"

for tekst in $tekst
do
echo $tekst
done

#wyświetla liczby od 1 do 10
echo "Wyswietlenie 10 liczb."
for i in {1..10}
  do
  echo $i
done

#wyświetla liczby od 20 do 0 co 2
echo "Wyswietlenie 10 liczb."
for i in {20..0..2}
  do
  echo $i
done
```

13. Skakanie po argumentach:

```
#!/bin/bash

for arg in "$@";
do
echo "tworze plik $arg"
touch $arg.txt
done
```

14. Operacje matematyczne (kalkulator):

```
 #!/bin/bash

read -p "podaj pierwsza liczbe " arg1
read -p "podaj druga liczbe " arg2
read -p "podaj znak (dodawanie(+), odejmowanie(-), mnozenie(*), dzielenie(/))" znak

case $znak in
  dodawanie|"+")
            wynik=$(($arg1+$arg2))
```

```
            echo "Wynik $arg1 + $arg2 to: $wynik"
        echo
        ;;
            odejmowanie|"-")
        wynik=`expr $arg1 - $arg2`
            echo "Wynik $arg1 - $arg2 to: $wynik"
        ;;
   mnozenie|"*")
            let wynik=arg1*arg2
            echo "Wynik $arg1 * $arg2 to: $wynik"
        ;;
   dzielenie|"/")
            if [ $arg2 -ne 0 ]
            then
            wynik=$(($arg1/$arg2))
            echo "Wynik $arg1 / $arg2 to: $wynik"
            else
            echo "nie dziel przez 0"
            fi
            ;;
   *)
            echo "niewlasciwe dane"
            echo
            ;;
esac
```

15. Ciąg Fibonacciego:

```
#!/bin/bash

read -p "Podaj liczbe wyrazow ciagu: " ile

if [ $ile -le 0 ];
   then echo "bledne dane"
elif [ $ile -eq 1 ];
  then
            tab[0]=0
else
  tab[0]=0
  tab[1]=1
  for ((i=2;i<$ile;i++))
            do
             let tab[i]=tab[i-2]+tab[i-1]
  done
fi
echo ${tab[*]}
```

16. Operacje na zasobach:

```
#!/bin/bash

while [ 1 == 1 ]
do
echo
echo "Możliwe opcje:
1) Utwórz katalog:
2) Utwórz plik:
3) Wyświetl zawartość bieżącego katalogu:
4) Wyświetl zawartość pliku
5) Wpisz tekst do pliku w terminalu
6) Kopiuj plik do katalogu
7) Zmień nazwę katalogu
8) Wyświetl zawartość katalogu
9) Wyświetli liczbę katalogów w bieżącej lokalizacji:
```

```
10) Wyświetli liczbę plików zwykłych w bieżącej lokalizacji:
11) Zakończ: "
echo
read -p "wybierz opcje: " o

case $o in
  1)
            read -p "podaj nazwe katalogu: " arg
            if [ ! -d "./$arg" ]
            then
                      echo "Tworze katalog $arg"
                      mkdir $arg
            else
                      echo "Katalog $arg juz istnieje"
            fi
            ;;
  2)
            read -p "podaj nazwe pliku: " arg
            read -p "podaj rozszerzenie pliku: " arg2
            if [ ! -f "./$arg.$arg2" ]
            then
                      echo "Tworze plik $arg.$arg2"
                      touch $arg.$arg2
            else
                      echo "Plik $arg.$arg2 juz istnieje"
            fi
            ;;
  3) ls ;;
  4)
            read -p "podaj nazwe pliku: " arg
            read -p "podaj rozszerzenie pliku: " arg2
            if [ -f "./$arg.$arg2" ]
            then
                      echo "Wyświetlam zawartość pliku $arg.$arg2 "
                      cat $arg.$arg2
            else
                      echo "Plik $arg.$arg2 nie istnieje"
            fi
            ;;
  5)  read -p "podaj nazwe pliku: " arg
            read -p "podaj rozszerzenie pliku: " arg2
            if [ -f "./$arg.$arg2" ]
            then
                      echo "Wpisz tekst do pliku $arg.$arg2 "
                      echo "Aby zakończyć wpisywanie naciśnij ctrl+d"
                      cat > $arg.$arg2
            else
                      echo "Plik $arg.$arg2 nie istnieje"
            fi
            ;;
  6)  read -p "podaj nazwe pliku: " arg
            read -p "podaj rozszerzenie pliku: " arg2
            if [ -f "./$arg.$arg2" ]
            then
                      read -p "podaj nazwe katalogu: " arg3
                      if [ -d "./$arg3" ]
                      then
                                echo "Kopiuje plik $arg.$arg2 do $arg3"
                                cp $arg.$arg2 $arg3
                      else
                                echo "Katalog $arg3 nie istnieje"
                      fi
            else
                      echo "Plik $arg.$arg2 nie istnieje"
            fi
            ;;
  7)
            read -p "podaj nazwe katalogu: " arg
            read -p "podaj nową nazwe katalogu: " arg2
            if [ -d "$arg" -a ! -d "$arg2" ]
```

```
86.             then
87.                     echo "Zmieniam nazwę katalogu $arg na $arg2"
88.                     mv $arg $arg2
89.             else
90.                     echo "katalog nie istnieje lub nowa nazwa jest już zajęta"
91.             fi
92.             ;;
93.   8)
94.             read -p "podaj nazwę katalogu " arg
95.             if [ -d "./$arg" ]
96.             then
97.                     echo "Wyświetlam zawartość katalogu $arg"

98.                     ls $arg
99.             else
100.                            echo "Katalog $arg nie istnieje"
101.                    fi
102.                    ;;
103.            9) ls -l | grep ^d | wc -l ;;
104.            10) ls -l | grep ^- | wc -l ;;
105.            11) echo "Koniec programu "
106.                    break ;;
107.            *)
108.                    echo "opcja nie została rozpoznana" ;;
109.  esac
110.  done
```

17. NWD – program:

```
1.  #!/bin/bash
2.
3.  echo "podaj dwie liczby "
4.  read -p "pierwsza liczba " a
5.  read -p "druga liczba " b
6.
7.  NWD_no(){
8.  while [ $a -ne $b ]
9.  do
10.   if [ $a -gt $b ]
11.   then
12.             let a=a-b
13.   else
14.             let b=b-a
15.   fi
16. done
17. echo "NWD to $a"
18. }
19.
20. NWD_zo(){
21. while [ $b -ne 0 ]
22. do
23.   let m=a%b
24.   a=$b
25.   b=$m
26. done
27.  echo "NWD to $a"
28. }
29.
30. NWD_no
31. NWD_zo
```

18. Ciąg arytmetyczny, suma i wypisanie składników:

```
1.   #!/bin/bash
2.
3.  read a r n
```

```
4.
5.  ciag[0]=$a
6.
7.  for ((i=1;i<$n;i++))
8.  do
9.    let ciag[i]=ciag[i-1]+r
10. done
11.
12. echo ${ciag[*]}
13.
14. let sum=(ciag[0]+ciag[n-1])/2
15. let sum*=n
16.
17. echo $sum
```

19. Kalkulator z użyciem funkcji:

```
1.  #!/bin/bash
2.  read -p "podaj pierwsza wartos " a
3.  read -p "podaj druga wartosc " b
4.  read -p "wybierz operator (+,-,*,/) " op
5.
6.
7.  dodaj(){
8.    wynik=$(($1+$2))
9.    echo "Wynik $1 + $2 = $wynik"
10. }
11.
12. odejmij(){
13.   let wynik=$a-$b
14.   echo "$wynik"
15. }
16.
17. mnozenia(){
18.   let wynik=a*b
19.   return $wynik
20. }
21.
22. dzielenie(){
23.   let wynik=a/b
24. }
25.
26. case $op in
27.   +)
28.   dodaj $a $b
29.   ;;
30.   -)
31.   odejmij $a $b
32.   ;;
33.   "*")
34.   mnozenia
35.   echo "Wynik $a * $b = $?"
36.   ;;
37.   /)
38.   if [ $b -eq 0 ]
39.   then
40.   echo "nie dziel przez 0"
41.   else
42.   dzielenie
43.   echo "$wynik"
44.   fi
45.   ;;
46. esac
```

20. Operacje na tablicy – suma, średnia, minimum, maksimum:

```
1.   #!/bin/bash
2.
3.  read -p "Liczba elementow tablicy  " n
4.
5.  for((i=0;i<$n;i++))
6.  do
7.  tab[i]=$((RANDOM%100))
8.  done
9.
10. suma(){
11.   sum=0
12.   for i in ${tab[*]}
13.   do
14.   let sum+=i
15.   done
16. }
17.
18. srednia(){
19.   suma
20.   let sr=sum/$#
21. }
22.
23. minimum(){
24.   min=$1
25.   for i in $@
26.   do
27.             if [ $i -lt $min ]
28.             then
29.                       min=$i
30.             fi
31.   done
32. }
33.
34. maksimum(){
35.   maks=${tab[0]}
36.   for((i=1;i<$n;i++))
37.   do
38.             if [ ${tab[i]} -gt $maks ]
39.             then
40.                       maks=${tab[i]}
41.             fi
42.   done
43. }
44.
45. while [ 1==1 ]
46. do
47. echo
48. echo "Możliwe opcje:
49. 1) Wyświetlenie tablicy
50. 2) Suma elementów tablicy
51. 3) Średnia z wartości w tablicy
```

```
4) Minimum w tablicy
5) Maksimum w tablicy
6) Koniec działania programu"
echo
read -p "Wybierz opcje " o

case $o in
1) echo ${tab[*]} ;;
2)
  suma
  echo "Suma wynosi $sum" ;;
3)
  srednia ${tab[*]}
  echo "Srednia wynosi $sr " ;;
4)
  minimum ${tab[*]}
  echo "Minimum wynosi $min " ;;
5)
  maksimum
  echo "Maksimum wynosi $maks " ;;
6) break ;;
*) echo "nieznana opcja " ;;
esac
done
```