Podstawy informatyki

WYKŁAD nr 02

Fizyka Techniczna, WFT PP

Michał Hermanowicz

Zakład Fizyki Obliczeniowej, Instytut Fizyki, Politechnika Poznańska

Rok akademicki 2018/2019

Spis rzeczy

Plan ramowy przedmiotu

Nr wykładu	Poruszane zagadnienia
I	Organizacja; forma i warunki zaliczenia; wprowadzenie
II	Powłoka **bash** i elementy programowania
III	Przetwarzanie danych #1
IV	Przetwarzanie danych #2
V	Reprezentacja danych (wykresy 2D i 3D) – *gnuplot*
VI	System składu tekstu LaTeX
VII	Pół-otwarty test zaliczeniowy

Każdemu z wykładów odpowiadają ćwiczenia realizowane na zajęciach w pracowni komputerowej.

Języki kompilowane i interpretowane

Jeden z możliwych podziałów języków programowania pozwala wyodrębnić:

- języki **kompilowane** – kod źródłowy jest tłumaczony na kod maszynowy za pomocą kompilatora (ANSI C, C++, FORTRAN i in.),
- języki **interpretowane** – kod źródłowy jest wykonywany bezpośrednio przez interpreter (Bash, Python, Perl, PHP i in.).

Języki kompilowane i interpretowane

Jeden z możliwych podziałów języków programowania pozwala wyodrębnić:

- języki **kompilowane** – kod źródłowy jest tłumaczony na kod maszynowy za pomocą kompilatora (ANSI C, C++, FORTRAN i in.),
- języki **interpretowane** – kod źródłowy jest wykonywany bezpośrednio przez interpreter (Bash, Python, Perl, PHP i in.).

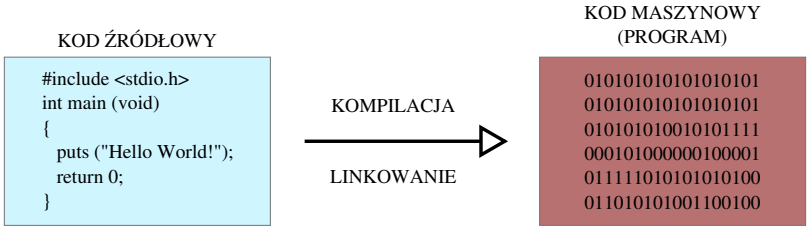
Narzędzia programistyczne:

- edytor tekstu, kompilator (w tym linker)/interpreter, debugger,
- opcjonalnie: środowisko programistyczne zawierające funkcjonalność wszystkich powyższych narzędzi (i więcej).

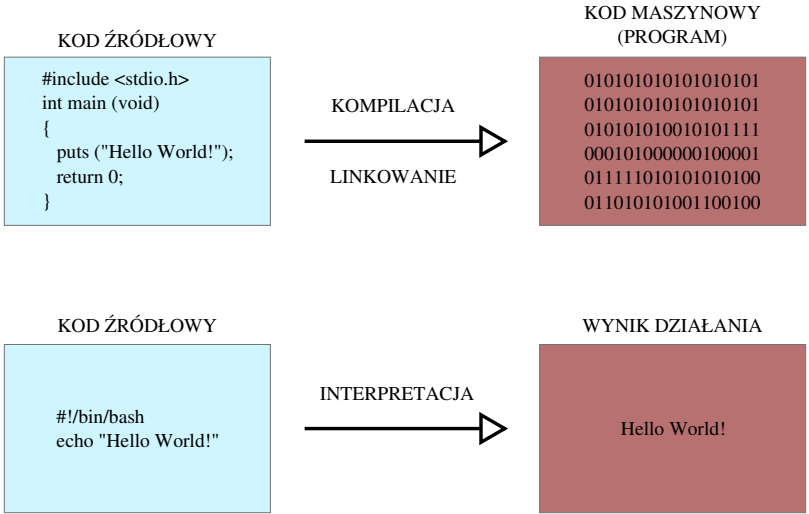
Przykładowe narzędzia:

- vim, emacs (edytory),
- gcc, gfortran (kompilatory); gdb (debugger).

Języki kompilowane i interpretowane



Języki kompilowane i interpretowane



vim: *vi improved* (edytor tekstu)

```
\section{Programowanie w środowisku GNU/Linux}
\subsection{Wprowadzenie}

\frame{
\frametitle{Języki kompilowane i interpretowane}

\begin{block}{Jeden z możliwych podziałów języków programowania pozwala wyodrębnić dwie (niejednoznaczne) kategorie:}
\begin{itemize}
\item języki \textbf{kompilowane} -- kod źródłowy jest tłumaczony na kod maszynowy za pomocą kompilatora (ANSI C, C++, FORTRAN i in.),
\item języki \textbf{interpretowane} -- kod źródłowy jest wykonywany bezpośrednio przez interpreter (bash, Python, Perl, PHP i in.).
\end{itemize}
\end{block}

\pause

\begin{exampleblock}{Narzędzia programistyczne:}
\begin{itemize}
\item edytor tekstu, kompilator (w tym linker)/interpreter, debugger,
\item opcjonalnie: środowisko programistyczne zawierające funkcjonalność wszystkich powyższych narzędzi (i więcej).
\end{itemize}
\end{exampleblock}

\small
Przykładowe narzędzia:
\begin{itemize}
\item vim, emacs (edytory),
\item gcc, gfortran (kompilatory); gdb (debugger).
\end{itemize}
}
```

- Edytor tekstu wydany na wiele plaform systemowych; bogactwo możliwości edycyjnych; wygodne *środowisko* programistyczne w połączeniu z kompilatorem gcc i narzędziami powłoki;
- wiele trybów edycji (podstawowa obsługa na zajęciach w pracowni).

vim: *vi improved* – wybrane polecenia (tryb *NORMAL*)

```
$ vim nazwa_pliku
```

- `:w nazwa` – zapis do pliku,
- `:wq` – zapis i wyjście,
- `:q!` – wyjście (ignoruj zmiany),
- `i` – tryb edycji,
- `[ESC]` – opuść tryb edycji,
- `u` – cofnij,
- `<Ctrl-R>` – powtórz,
- `/wzorzec` – szukaj wzorca (`n` – nast. / `N` – poprzedni),
- `:!polecenie` – polec. powłoki,
- `dd` – usuń bieżący wiersz,
- `D` – usuń stąd do końca wiersza,
- `:%s/raz/dwa/g` – znajdź i zastąp (*raz*→*dwa*),
- `:s/raz/dwa/g` – znajdź i zastąp w bieżącym wierszu,
- `:s/raz/dwa/gc` – znajdź i zastąp z potwierdzeniem,
- `v` – zaznacz (przesuwając kursor: `[←]`, `[↑]`, `[↓]`, `[→]`),
- `y` – skopiuj zaznaczenie,
- `d` – wytnij zaznaczenie,
- `p` – wklej skopiowane.

Kompilacja kodu źródłowego

```
student@wftlab-180:~$
```

Kompilacja kodu źródłowego

```
student@wftlab-180:~$ cat hello.c
```

Kompilacja kodu źródłowego

```
student@wftlab-180:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@wftlab-180:~$
```

Kompilacja kodu źródłowego

```
student@wftlab-180:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@wftlab-180:~$ gcc hello.c
```

Kompilacja kodu źródłowego

```
student@wftlab-180:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@wftlab-180:~$ gcc hello.c
student@wftlab-180:~$
```

Kompilacja kodu źródłowego

```
student@wftlab-180:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@wftlab-180:~$ gcc hello.c
student@wftlab-180:~$ ls
```

Kompilacja kodu źródłowego

```
student@wftlab-180:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@wftlab-180:~$ gcc hello.c
student@wftlab-180:~$ ls
a.out  hello.c
student@wftlab-180:~$
```

Kompilacja kodu źródłowego

```
student@wftlab-180:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@wftlab-180:~$ gcc hello.c
student@wftlab-180:~$ ls
a.out  hello.c
student@wftlab-180:~$ file a.out
```

Kompilacja kodu źródłowego

```
student@wftlab-180:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@wftlab-180:~$ gcc hello.c
student@wftlab-180:~$ ls
a.out  hello.c
student@wftlab-180:~$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32 (...)
student@wftlab-180:~$
```

Kompilacja kodu źródłowego

```
student@wftlab-180:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@wftlab-180:~$ gcc hello.c
student@wftlab-180:~$ ls
a.out  hello.c
student@wftlab-180:~$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32 (...)
student@wftlab-180:~$ ./a.out
```

Kompilacja kodu źródłowego

```
student@wftlab-180:~$ cat hello.c
#include <stdio.h>
int main (void)
{
puts ("Hello World!");
return 0;
}
student@wftlab-180:~$ gcc hello.c
student@wftlab-180:~$ ls
a.out  hello.c
student@wftlab-180:~$ file a.out
a.out: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32 (...)
student@wftlab-180:~$ ./a.out
Hello World!
```

Uruchamianie skryptów powłoki

Więcej informacji nt GCC i możliwych opcji:

```
student@wftlab-180:~$ man gcc
```

```
student@wftlab-180:~$
```

Uruchamianie skryptów powłoki

Więcej informacji nt GCC i możliwych opcji:

```
student@wftlab-180:~$ man gcc
```

```
student@wftlab-180:~$ cat hello.sh
```

Uruchamianie skryptów powłoki

Więcej informacji nt GCC i możliwych opcji:

```
student@wftlab-180:~$ man gcc
```

```
student@wftlab-180:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@wftlab-180:~$
```

Uruchamianie skryptów powłoki

Więcej informacji nt GCC i możliwych opcji:

```
student@wftlab-180:~$ man gcc
```

```
student@wftlab-180:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@wftlab-180:~$ file hello.sh
```

Uruchamianie skryptów powłoki

Więcej informacji nt GCC i możliwych opcji:

```
student@wftlab-180:~$ man gcc
```

```
student@wftlab-180:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@wftlab-180:~$ file hello.sh
hello.sh: Bourne-shell script, ASCII text executable
student@wftlab-180:~$
```

Uruchamianie skryptów powłoki

Więcej informacji nt GCC i możliwych opcji:

```
student@wftlab-180:~$ man gcc
```

```
student@wftlab-180:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@wftlab-180:~$ file hello.sh
hello.sh: Bourne-shell script, ASCII text executable
student@wftlab-180:~$ ./hello.sh
```

Uruchamianie skryptów powłoki

Więcej informacji nt GCC i możliwych opcji:

```
student@wftlab-180:~$ man gcc
```

```
student@wftlab-180:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@wftlab-180:~$ file hello.sh
hello.sh: Bourne-shell script, ASCII text executable
student@wftlab-180:~$ ./hello.sh
Hello World!
student@wftlab-180:~$
```

Uruchamianie skryptów powłoki

Więcej informacji nt GCC i możliwych opcji:

```
student@wftlab-180:~$ man gcc
```

```
student@wftlab-180:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@wftlab-180:~$ file hello.sh
hello.sh: Bourne-shell script, ASCII text executable
student@wftlab-180:~$ ./hello.sh
Hello World!
student@wftlab-180:~$ ls -l hello.sh
```

Uruchamianie skryptów powłoki

Więcej informacji nt GCC i możliwych opcji:

```
student@wftlab-180:~$ man gcc
```

```
student@wftlab-180:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@wftlab-180:~$ file hello.sh
hello.sh: Bourne-shell script, ASCII text executable
student@wftlab-180:~$ ./hello.sh
Hello World!
student@wftlab-180:~$ ls -l hello.sh
-rwxr-xr-x 1 student student 33 paź 22 12:13 hello.sh
```

Uruchamianie skryptów powłoki

Więcej informacji nt GCC i możliwych opcji:

```
student@wftlab-180:~$ man gcc
```

```
student@wftlab-180:~$ cat hello.sh
#!/bin/bash
echo "Hello World!"
student@wftlab-180:~$ file hello.sh
hello.sh: Bourne-shell script, ASCII text executable
student@wftlab-180:~$ ./hello.sh
Hello World!
student@wftlab-180:~$ ls -l hello.sh
-rwxr-xr-x 1 student student 33 paź 22 12:13 hello.sh
```

Shebang line – linia specjalna:

```
#!/bin/bash
```

Uruchamianie skryptów powłoki

```
$ ./hello.sh
```

Ale:

musi być wykonywalny
(`chmod +x hello.sh`).

```
$ bash ./hello.sh
```

Wówczas:

nie musi być wykonywalny
(wystarczy `-r--------`).

Uruchamianie skryptów powłoki

```
$ ./hello.sh
```

Ale:

musi być wykonywalny
(`chmod +x hello.sh`).

```
$ bash ./hello.sh
```

Wówczas:

nie musi być wykonywalny
(wystarczy `-r--------`).

Standardowe wyjście (*stdout*, 1)

```
$ ./hello.sh
Hello World!
```
← standardowe WYjście

Uruchamianie skryptów powłoki

```
$ ./hello.sh
```

Ale:

musi być wykonywalny
(`chmod +x hello.sh`).

```
$ bash ./hello.sh
```

Wówczas:

nie musi być wykonywalny
(wystarczy `-r--------`).

Standardowe wyjście (*stdout*, 1)

```
$ ./hello.sh
Hello World!
```

← standardowe WYjście

Przekierowanie *stdout* do pliku:

```
$ ./hello.sh > plik
```

Uruchamianie skryptów powłoki

```
$ ./hello.sh
```

Ale:

musi być wykonywalny
(`chmod +x hello.sh`).

```
$ bash ./hello.sh
```

Wówczas:

nie musi być wykonywalny
(wystarczy `-r--------`).

Standardowe wyjście (*stdout*, 1)

```
$ ./hello.sh
Hello World!
```
← standardowe WYjście

Przekierowanie *stdout* do pliku:

```
$ ./hello.sh > plik
```

Alternatywnie:

```
$ ./hello.sh 1>plik
```

Uruchamianie skryptów powłoki

Analogicznie (*stderr* = 2):

```
$ ./hello.sh 2>plik_err
```

Uruchamianie skryptów powłoki

Analogicznie (*stderr* = 2):

```
$ ./hello.sh 2>plik_err
```

stderr → *stdout*:

```
$ ./hello.sh 2>&1
```

Uruchamianie skryptów powłoki

Analogicznie (*stderr* = 2):

```
$ ./hello.sh 2>plik_err
```

stderr → *stdout*:

```
$ ./hello.sh 2>&1
```

stdout i *stderr* do **jednego** pliku:

```
$ ./hello.sh &>plik
```

Uruchamianie skryptów powłoki

Analogicznie (*stderr* = 2):

```
$ ./hello.sh 2>plik_err
```

stderr → *stdout*:

```
$ ./hello.sh 2>&1
```

stdout i *stderr* do **jednego** pliku:

```
$ ./hello.sh &>plik
```

Można też:

```
$ ./hello.sh &>/dev/null
```

Uruchamianie skryptów powłoki

Analogicznie (*stderr* = 2):

```
$ ./hello.sh 2>plik_err
```

stderr → *stdout*:

```
$ ./hello.sh 2>&1
```

stdout i *stderr* do **jednego** pliku:

```
$ ./hello.sh &>plik
```

Można też:

```
$ ./hello.sh &>/dev/null
```

A także:

```
$ ./hello.sh 1>plik 2>bledy
```

Uruchamianie skryptów powłoki

/dev/null

to plik specjalny – urządzenie zerowe (tzw. *czarna dziura*). Przekierowanie informacji do tego pliku powoduje ich utratę (bezpowrotne usunięcie).

Uruchamianie skryptów powłoki

/dev/null

to plik specjalny – urządzenie zerowe (tzw. *czarna dziura*). Przekierowanie informacji do tego pliku powoduje ich utratę (bezpowrotne usunięcie).

W systemie GNU/Linux wszystko jest plikiem, również urządzenia, dlatego można zrobić tak:

```
$ ./hello.sh > /dev/lpr
```

`/dev/lpr` to drukarka. Jeżeli jest podłączona do komputera i skonfigurowana w systemie – wówczas wynik działania skryptu zostanie wydrukowany.

Uruchamianie skryptów powłoki

/dev/null

to plik specjalny – urządzenie zerowe (tzw. *czarna dziura*). Przekierowanie informacji do tego pliku powoduje ich utratę (bezpowrotne usunięcie).

W systemie GNU/Linux wszystko jest plikiem, również urządzenia, dlatego można zrobić tak:

```
$ ./hello.sh > /dev/lpr
```

`/dev/lpr` to drukarka. Jeżeli jest podłączona do komputera i skonfigurowana w systemie – wówczas wynik działania skryptu zostanie wydrukowany.

Inne pliki urządzeń: skanery, plotery, dyski twarde, napędy CD-ROM, stacje dyskietek, streamery itd.

Narzędzie grep

grep (man grep)

przeszukuje zadany plik (lub *stdout*) w poszukiwaniu wierszy zawierających określony *wzorzec*.

SKŁADNIA: `grep WZORZEC plik`

Przykład:

```
$ cat liczby.txt
45678
23
2435
67876
245723
$ grep 23 liczby.txt
23
245723
```

Potok (*pipeline*)

Potok – *pipeline* (symbol: |)

przekierowuje standardowe WYjście (*stdout*) programu na WEjście innego.

Przykład z użyciem narzędzia `grep`:

```
$ cat liczby.txt | grep 23
```

Potok (*pipeline*)

Potok – *pipeline* (symbol: |)

przekierowuje standardowe WYjście (*stdout*) programu na WEjście innego.

Przykład z użyciem narzędzia grep:

```
$ cat liczby.txt | grep 23
23
245723
```

Potok (*pipeline*)

Potok – *pipeline* (symbol: |)

przekierowuje standardowe WYjście (*stdout*) programu na WEjście innego.

Przykład z użyciem narzędzia grep:

```
$ cat liczby.txt | grep 23
23
245723
```

Narzędzie wc (*word count*) - zlicza słowa, wiersze, bajty:

```
$ wc -l liczby.txt
```
(← z przełącznikiem `-l` zlicza wiersze)

Potok (*pipeline*)

Potok – *pipeline* (symbol: |)

przekierowuje standardowe WYjście (*stdout*) programu na WEjście innego.

Przykład z użyciem narzędzia grep:

```
$ cat liczby.txt | grep 23
23
245723
```

Narzędzie wc (*word count*) - zlicza słowa, wiersze, bajty:

```
$ wc -l liczby.txt    (← z przełącznikiem -l zlicza wiersze)
5
```

Potok (*pipeline*)

Potok – *pipeline* (symbol: |)

przekierowuje standardowe WYjście (*stdout*) programu na WEjście innego.

Przykład z użyciem narzędzia grep:

```
$ cat liczby.txt | grep 23
23
245723
```

Narzędzie wc (*word count*) - zlicza słowa, wiersze, bajty:

```
$ wc -l liczby.txt    (← z przełącznikiem -l zlicza wiersze)
5
$ cat liczby.txt | grep 23 | wc -l
```

Potok (*pipeline*)

Potok – *pipeline* (symbol: |)

przekierowuje standardowe WYjście (*stdout*) programu na WEjście innego.

Przykład z użyciem narzędzia grep:

```
$ cat liczby.txt | grep 23
23
245723
```

Narzędzie wc (*word count*) - zlicza słowa, wiersze, bajty:

```
$ wc -l liczby.txt
```
(← z przełącznikiem -l zlicza wiersze)

```
5
$ cat liczby.txt | grep 23 | wc -l
2
```

Potok (*pipeline*)

Narzędzie `tee` (`man tee`): czyta WEjście i zapisuje na WYjście oraz do pliku.

Przykład:

```
$ cat liczby.txt | tee out
```

Potok (*pipeline*)

Narzędzie `tee` (`man tee`): czyta WEjście i zapisuje na WYjście **oraz** do pliku.

Przykład:

```
$ cat liczby.txt | tee out
45678
23
2435
67876
245723
```

Potok (*pipeline*)

Narzędzie `tee` (`man tee`): czyta WEjście i zapisuje na WYjście oraz do pliku.

Przykład:

```
$ cat liczby.txt | tee out
45678
23
2435
67876
245723
$ cat out
```

Potok (*pipeline*)

Narzędzie `tee` (`man tee`): czyta WEjście i zapisuje na WYjście **oraz** do pliku.

Przykład:

```
$ cat liczby.txt | tee out
45678
23
2435
67876
245723
$ cat out
45678
23
2435
67876
245723
```

Pętla for:

Pętla for:

```
for i in 1 2 3 4 5
do
  echo $i
done
```

Pętla for:

Pętla for:

```
for i in 1 2 3 4 5
do
  echo $i
done
```

Wynik działania:

```
1
2
3
4
5
```

Pętla for:

Pętla for:

```
for i in 1 2 3 4 5
do
  echo $i
done
```

Wynik działania:

```
1
2
3
4
5
```

Dla każdego elementu `i` z zadanego zbioru wykonaj instrukcje zawarte pomiędzy `do` a `done`. Zmienne wywołujemy poprzedzając je znakiem $.

Pętla `for`:

Można inaczej:

```
$ seq 1 1 5
```

Pętla `for`:

Można inaczej:

```
$ seq 1 1 5
1
2
3
4
5
```

Pętla `for`:

Można inaczej:

```
$ seq 1 1 5
1
2
3
4
5
```

SKŁADNIA: `seq [początek] [krok] [koniec]`

Pętla `for`:

Można inaczej:

```
$ seq 1 1 5
1
2
3
4
5
```

SKŁADNIA: `seq [początek] [krok] [koniec]`

A w skrypcie (symbol ‘ pod klawiszem ESC):

```
for i in ‘seq 1 1 5‘
do
  echo $i
done
```

Pętla `for`:

```
$ seq 0.0 0.5 2
0.0
0.5
1.0
1.5
2.0
```

Pętla `for`:

```
$ seq 0.0 0.5 2
0.0
0.5
1.0
1.5
2.0
```

Uwaga na separator dziesiętny – zależy od lokalizacji środowiska (`export LC_NUMERIC=C`).

Pętla `for`:

```
$ seq 0.0 0.5 2
0.0
0.5
1.0
1.5
2.0
```

Uwaga na separator dziesiętny – zależy od lokalizacji środowiska (`export LC_NUMERIC=C`).

Alternatywna składnia pętli `for`:

```
for ((i=0; i<10; i++))
do
  echo $i
done
```

Pętla while:

Pętla while:

```
x=1
while [ $x -le 4 ]
do
  echo "x=$x"
  x=$[x + 1]
done
```

Pętla while:

Pętla while:

```
x=1
while [ $x -le 4 ]
do
  echo "x=$x"
  x=$[x + 1]
done
```

Wynik działania:

x=1
x=2
x=3
x=4

Pętla `while`:

Pętla `while`:

```
x=1
while [ $x -le 4 ]
do
  echo "x=$x"
  x=$[x + 1]
done
```

Wynik działania:

```
x=1
x=2
x=3
x=4
```

Wykonuj instrukcje w pętli dopóki `[ warunek ]` pozostaje spełniony.

Pętla until:

Pętla until:

```
x=1
until [ $x -gt 4 ]
do
  echo "x=$x"
  x=$[x + 1]
done
```

Pętla until:

Pętla until:

```
x=1
until [ $x -gt 4 ]
do
  echo "x=$x"
  x=$[x + 1]
done
```

Wynik działania:

x=1
x=2
x=3
x=4

Pętla `until`:

Pętla `until`:

```
x=1
until [ $x -gt 4 ]
do
  echo "x=$x"
  x=$[x + 1]
done
```

Wynik działania:

x=1
x=2
x=3
x=4

Wykonuj instrukcje w pętli dopóki `[ warunek ]` pozostaje **nie**spełniony.

Instrukcja warunkowa `if`

Konstrukcje typu [warunek] (spacje przed/po są konieczne!) realizują instrukcję `test` (`man test`).

Na przykład:

```
if [ -e plik ]
then
  operacja1
else
  operacja2
fi
```

Jeżeli *plik* istnieje – wykonaj operację 1, jeżeli nie istnieje – wykonaj operację 2. Można zadać więcej warunków poprzez `elif [ warunek ]`. Instrukcje `elif` i `else` są opcjonalne.
Wszystkie możliwe warunki można sprawdzić w dokumentacji polecenia `test` (`man test`).

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
student@wftlab-180:~$ echo $b
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
student@wftlab-180:~$ echo $b
4
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
student@wftlab-180:~$ echo $b
4
student@wftlab-180:~$ expr 5 + 5
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
student@wftlab-180:~$ echo $b
4
student@wftlab-180:~$ expr 5 + 5
10
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
student@wftlab-180:~$ echo $b
4
student@wftlab-180:~$ expr 5 + 5
10
student@wftlab-180:~$ c=‘expr 5 + 5‘
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
student@wftlab-180:~$ echo $b
4
student@wftlab-180:~$ expr 5 + 5
10
student@wftlab-180:~$ c=‘expr 5 + 5‘
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
student@wftlab-180:~$ echo $b
4
student@wftlab-180:~$ expr 5 + 5
10
student@wftlab-180:~$ c=‘expr 5 + 5‘
student@wftlab-180:~$ echo $c
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
student@wftlab-180:~$ echo $b
4
student@wftlab-180:~$ expr 5 + 5
10
student@wftlab-180:~$ c=‘expr 5 + 5‘
student@wftlab-180:~$ echo $c
10
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
student@wftlab-180:~$ echo $b
4
student@wftlab-180:~$ expr 5 + 5
10
student@wftlab-180:~$ c=‘expr 5 + 5‘
student@wftlab-180:~$ echo $c
10
student@wftlab-180:~$ let c=‘expr $c + 5‘
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ a=1
student@wftlab-180:~$ echo $a
1
student@wftlab-180:~$ echo $((2+2))
4
student@wftlab-180:~$ b=2
student@wftlab-180:~$ let b=$b+2
student@wftlab-180:~$ echo $b
4
student@wftlab-180:~$ expr 5 + 5
10
student@wftlab-180:~$ c=‘expr 5 + 5‘
student@wftlab-180:~$ echo $c
10
student@wftlab-180:~$ let c=‘expr $c + 5‘; echo $c
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$ ((d = $d + 10))
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$ ((d = $d + 10))
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$ ((d = $d + 10))
student@wftlab-180:~$ echo $d
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$ ((d = $d + 10))
student@wftlab-180:~$ echo $d
30
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$ ((d = $d + 10))
student@wftlab-180:~$ echo $d
30
student@wftlab-180:~$ ((d++))
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$ ((d = $d + 10))
student@wftlab-180:~$ echo $d
30
student@wftlab-180:~$ ((d++)); echo $d
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$ ((d = $d + 10))
student@wftlab-180:~$ echo $d
30
student@wftlab-180:~$ ((d++)); echo $d
31
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$ ((d = $d + 10))
student@wftlab-180:~$ echo $d
30
student@wftlab-180:~$ ((d++)); echo $d
31
student@wftlab-180:~$
```

Możliwa jest prosta arytmetyka w powłoce Bash, ale możemy też użyć zewnętrznego narzędzia do wykonywania operacji matematycznych, np. `bc` (więcej informacji: `man bc`).

```
student@wftlab-180:~$ echo 13/2 | bc
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$ ((d = $d + 10))
student@wftlab-180:~$ echo $d
30
student@wftlab-180:~$ ((d++)); echo $d
31
student@wftlab-180:~$
```

Możliwa jest prosta arytmetyka w powłoce Bash, ale możemy też użyć zewnętrznego narzędzia do wykonywania operacji matematycznych, np. `bc` (więcej informacji: `man bc`).

```
student@wftlab-180:~$ echo 13/2 | bc
6
student@wftlab-180:~$
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$ ((d = $d + 10))
student@wftlab-180:~$ echo $d
30
student@wftlab-180:~$ ((d++)); echo $d
31
student@wftlab-180:~$
```

Możliwa jest prosta arytmetyka w powłoce Bash, ale możemy też użyć zewnętrznego narzędzia do wykonywania operacji matematycznych, np. `bc` (więcej informacji: `man bc`).

```
student@wftlab-180:~$ echo 13/2 | bc
6
student@wftlab-180:~$ echo 13/2 | bc -l
```

Arytmetyka w Bashu

```
student@wftlab-180:~$ d=20
student@wftlab-180:~$ ((d = $d + 10))
student@wftlab-180:~$ echo $d
30
student@wftlab-180:~$ ((d++)); echo $d
31
student@wftlab-180:~$
```

Możliwa jest prosta arytmetyka w powłoce Bash, ale możemy też użyć zewnętrznego narzędzia do wykonywania operacji matematycznych, np. `bc` (więcej informacji: `man bc`).

```
student@wftlab-180:~$ echo 13/2 | bc
6
student@wftlab-180:~$ echo 13/2 | bc -l
6.50000000000000000000
student@wftlab-180:~$
```

Przykładowe zadanie

Zadanie z zajęć w pracowni:

a utworzyć katalog o nazwie *zadanie1*, a w nim (za pomocą skryptu) wygenerować 1366 plików o nazwach zawierających liczbę losową (przed utworzeniem upewnić się, że plik o zadanej nazwie nie istnieje już w bieżącym katalogu);

b zmodyfikować skrypt w taki sposób, żeby pliki zawierające w nazwie liczbę parzystą/nieparzystą były tworzone w oddzielnych podkatalogach.

Przykładowe zadanie

Zadanie z zajęć w pracowni:

a utworzyć katalog o nazwie *zadanie1*, a w nim (za pomocą skryptu) wygenerować 1366 plików o nazwach zawierających liczbę losową (przed utworzeniem upewnić się, że plik o zadanej nazwie nie istnieje już w bieżącym katalogu);

b zmodyfikować skrypt w taki sposób, żeby pliki zawierające w nazwie liczbę parzystą/nieparzystą były tworzone w oddzielnych podkatalogach.

Przydatne informacje:

- polecenie `touch` – zmiana czasu dostępu/modyfikacji pliku, ale pozwala też utworzyć pusty plik (więcej: `man touch`),
- zmienna RANDOM: `echo $RANDOM` zwraca liczbę pseudolosową,
- polecenia: `mkdir` (tworzenie katalogów), `rm` (usuwanie plików).

Przydatne polecenia

- `ls` – lista plików w bież. katalogu,
- `mkdir nazwa` – utwórz katalog,
- `cd nazwa` – przejdź do katalogu,
- `cd ..` – przejdź do kat. *wyżej*,
- `pwd` – ścieżka dost. do bież. katalogu,
- `rm *` – usuń wszystkie pliki w bieżącym katalogu,
- `rm -R *` – usuń wszystkie pliki i katalogi w bieżącym katalogu,
- `touch plik` – zmienia czas modyf. / tworzy nieistniejący plik,
- `cat nazwa` – wyświetla zawartość pliku *nazwa*,
- `cp nazwa1 nazwa2` – skopiuj *nazwa1*→*nazwa2*,
- `mv nazwa1 nazwa2` – przenieś / zmień nazwę,
- `grep wzorzec plik` – zwraca wiersze (z pliku) zawierające *wzorzec*,
- `less plik` – przeglądaj zawartość pliku (wyjście: `q`),

- `[TAB]` –uzupełnianie składni,
- `[↑]` – poprzednie polecenie,
- `nazwa (./nazwa)` – uruchamia program,
- `.` – symbol bieżącego katalogu,
- `..` – katalog *wyżej* w systemie plików.

Ćwiczenia

- Typy zmiennych (`declare` i `readonly`), zmienne środowiskowe (`env`, `export`) i specjalne,

Ćwiczenia

- Typy zmiennych (`declare` i `readonly`), zmienne środowiskowe (`env`, `export`) i specjalne,
- przekazywanie argumentów do skryptu, proste wyrażenia regularne i `grep`,
- funkcje, wartości zwracane przez funkcje i skrypty, (...)

Podsumowanie

Pytania i dyskusja