# PARP – informacje pomocnicze[1]

## Instalacja narzędzi do Prologa, Haskella i Smalltalka

Jeśli korzystasz z Ubuntu zainstaluj pakiety `swi-prolog`, `haskell-platform` i `gnu-smalltalk`:

```
$ sudo apt-get update
$ sudo apt-get install -y swi-prolog
$ sudo apt-get install -y haskell-platform
$ sudo apt-get install -y gnu-smalltalk
```

Jeśli korzystasz z innych dystrybucji Linuxa, znajdź i zainstaluj odpowiedniki tych pakietów.

Jeśli nie korzystasz z Linuxa, lub dla Twojej dystrybucji nie ma potrzebnych pakietów, możesz skorzystać z wirtualnej maszyny Ubuntu.

### Instalacja wirtualnej maszyny Ubuntu

*Uwaga: Wykonanie poniższej instrukcji spowoduje zajęcie ok. 4 GB miejsca na dysku.*

1. Pobierz i zainstaluj Ubuntu Multipass ze strony: https://multipass.run.
2. Uruchom terminal i wykonaj polecenie:

   ```
   multipass shell
   ```

   Zostanie pobrany obraz Ubuntu. Następnie zostanie utworzona i uruchomiona instancja Ubuntu o nazwie `primary`, a na koniec uruchomiona powłoka tej instancji.
3. Zainstaluj pakiety `swi-prolog`, `haskell-platform` i `gnu-smalltalk`:

```
ubuntu@primary:~$ sudo apt-get update
ubuntu@primary:~$ sudo apt-get install -y swi-prolog
ubuntu@primary:~$ sudo apt-get install -y haskell-platform
ubuntu@primary:~$ sudo apt-get install -y gnu-smalltalk
```

4. Katalog `~/Home` jest podmontowanym katalogiem domowym z systemu macierzystego. Utwórz w nim katalog roboczy:

   ```
   ubuntu@primary:~$ mkdir Home/PARP
   ```

   W tym katalogu możesz teraz tworzyć i edytować pliki w systemie macierzystym, a następnie korzystać z nich w powłoce Ubuntu.

5. Żeby opuścić powłokę i zatrzymać instancję `primary` wykonaj polecenia:

   ```
   ubuntu@primary:~$ logout
   multipass stop
   ```

   Nie zostawiaj działającej instancji (zużywa zasoby systemu macierzystego).

6. W celu ponownego uruchomienia instancji `primary` i przejścia do jej powłoki wykonaj polecenie:

   ```
   multipass shell
   ```

7. Na koniec semestru usuń instancję `primary` (i odinstaluj Ubuntu Multipass) :

   ```
   multipass delete primary
   multipass purge
   ```

## Praca z językiem SWI-Prolog

1. W katalogu roboczym utwórz plik tekstowy o rozszerzeniu `.pl` (np. `kb.pl`) z bazą wiedzy, np.:

```
woman(mia).
woman(jody).
woman(yolanda).
```

2. Przejdź do katalogu roboczego i uruchom środowisko Prologa:

```
ubuntu@primary:~$ cd ~/Home/PARP
ubuntu@primary:~/Home/PARP$ swipl
?-
```

3. Wczytaj plik z bazą wiedzy i wykonuj zapytania (pamiętaj o kropce na końcu):

```
?- [kb].
true.

?- woman(mia).
true.
```

4. Możesz edytować plik źródłowy. Po zapisaniu zmian uruchom polecenie:

```
?- make.
```

5. Zatrzymaj środowisko Prologa poleceniem:

```
?- halt.
```

# Praca z językiem Haskell

### Rozwijanie programu

1. W katalogu roboczym utwórz plik tekstowy o rozszerzeniu `.hs` (np. `main.hs`) z kodem, np.:

```
x = 5
increment n = n + 1
main = print (increment x)
```

2. Przejdź do katalogu roboczego i uruchom środowisko GHCi:

```
ubuntu@primary:~$ cd ~/Home/PARP
ubuntu@primary:~/Home/PARP$ ghci
Prelude>
```

3. Wczytaj plik z kodem:

```
Prelude> :l main.hs
*Main>
```

4. Sprawdź działanie definiowanych funkcji i akcji:

```
*Main> x
5
*Main> increment 0
1
*Main> main
6
```

5. Możesz edytować plik źródłowy. Po zapisaniu zmian wykonaj polecenie:

```
*Main> :r
```

6. Zatrzymaj środowisko GHCi poleceniem:

```
*Main> :q
```

### Uruchomienie skończonego programu

Skończony program możesz:

1. Uruchomić przy użyciu interpretera:

```
ubuntu@primary:~/Home/PARP$ runhaskell main.hs
```

2. Skompilować (plik musi posiadać akcję `main`) i uruchomić otrzymaną wersję binarną :

```
ubuntu@primary:~/Home/PARP$ ghc main.hs -O2 -o myprog
ubuntu@primary:~/Home/PARP$ ./myprog
```

## Praca z językiem GNU Smalltalk

### Rozwijanie programu

1. W katalogu roboczym utwórz plik tekstowy o rozszerzeniu `.st` (np. `account.st`) z kodem, np.:

```
Object subclass: Account [
    |balance|

    Account class >> new [
        |r|
        r := super new.
        r init.
        ^r
    ]
    init [
```

```
        balance := 0
    ]
    getBalance [
        ^balance
    ]
    deposit: amount [
        balance := balance + amount
    ]
]
```

2. Przejdź do katalogu roboczego i uruchom środowisko Smalltalka:

```
ubuntu@primary:~$ cd ~/Home/PARP
ubuntu@primary:~/Home/PARP$ gst
st>
```

3. Wczytaj plik z kodem (tj. złącz jego zawartość z aktualnym obrazem):

```
st> FileStream fileIn: 'account.st'
FileStream
```

4. Twórz obiekty, wysyłaj komunikaty:

```
st> a := Account new
an Account
st> a deposit: 100
an Account
st> a getBalance
100
st> a inspect
An instance of Account
  balance: 0
an Account
```

5. Możesz edytować plik źródłowy. Po zapisaniu zmian wczytaj ponownie plik z kodem.

6. Zatrzymaj środowisko GNU Smalltalk wciskając `Ctrl + D` lub wysyłając komunikat:

```
st> ObjectMemory quit
```

### Uruchomienie skończonego programu

Pliki z kodem możesz uruchomić przekazując je jako argumenty dla `gst`. Pliki są parsowane i wykonywane po kolei. Np.:

```
ubuntu@primary:~/Home/PARP$ gst account.st main.st
100
```

przy założeniu, że plik `main.st` miał postać:

```
a := Account new.
a deposit: 100.
Transcript show: a getBalance printString; cr.
```

---

1. Opracowanie: dr inż. Marcin Szlenk ↩