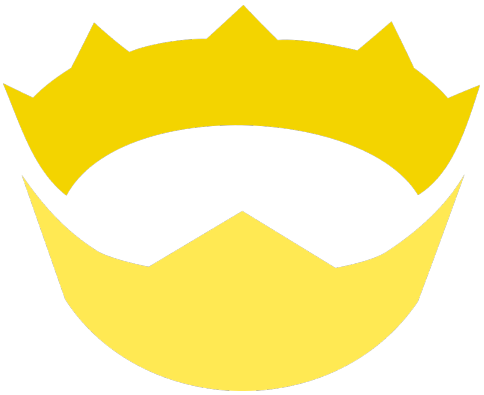
Podstawy programowania w Nim
podręcznik użytkownika

Namysław Tatarynowicz

6 czerwca 2023



Spis treści

1 Wprowadzenie do języka Nim

1.1 Wstęp

Uwaga

Ten podręcznik nie jest oficjalnym dokumentem i nie jest powiązany z `nim-lang.org`.

Tworząc ten dokument przyjąłem sobie za cel przedstawienie podstaw programowania w Nim ale również jest to sposób na moją naukę tego języka. Zawartość tego dokumentu będzie ewoluowała wraz ze wzrostem mojej świadomości programowania w Nim.

Niniejszy podręcznik jest w bardzo wczesnej fazie tworzenia. Posiada wiele elementów do poprawy, o których wiem, ale też mnóstwo, z których nie zdaję sobie jeszcze sprawy. Gdy zauważysz merytoryczny błąd lub masz sugestię, proszę o kontakt na adres mailowy: `n.tatarynowicz@gmail.com`.

Opracowując ten dokument, korzystam z następującego oprogramowania:

```
system operacyjny: antiX 22
język            : Nim 1.6.12
kompilator       : glibc 2.28
edytor kodu      : Geany 1.33
```

1.2 Poznajmy Nima

Nim jest językiem programowania o składni zbliżonej do języka Python, ale kompilowanym. Dzięki temu umożliwia tworzenie programów o podobnej wydajności jak programy napisane w języku C. Nim do kompilacji kodu wykorzystuje jedną z dostępnych bibliotek języka C/C++ (domyślnie C). Zatem, podczas kompilacji programu, najpierw powstaje plik C i dopiero ten plik jest kompilowany. Ostatecznie Nim daje możliwość kompilacji do C, C++ lub JavaScript.

Podobnie, jak w języku Python, w Nim także możemy korzystać z gotowych modułów, które importujemy tak samo w obu językach. Trzeba zauważyć, że Nim nie posiada tak dużej ich liczby, jak Python.

Najnowszą wersję Nima znajdziemy na stronie `https://nim-lang.org/`. W momencie tworzenia tego dokumentu, na stronie twórców, Nim jest dostępny w wersji 1.6.12.

Znajduje się tam również środowisko, dzięki któremu będziemy mogli skompilować i uruchomić nasz kod on-line oraz wygenerować odnośnik i go udostępnić: `https://play.nim-lang.org/`.

Język Nim jest wolnym oprogramowaniem — jest rozprowadzany na licencji MIT.

Jeśli napotkamy na jakiś programistyczny problem, to dzięki społeczności Nima możemy uzyskać pomoc w jego rozwiązaniu. Poniżej umieszczam listę miejsc, które możemy odwiedzić. W momencie przygotowywania tego podręcznika nie spotkałem polskojęzycznych kanałów społecznościowych przeznaczonych użytkownikom Nima.

blog:
`https://nim-lang.org/blog.html`

forum:
`https://forum.nim-lang.org/`

IRC:
`irc.libera.chat, #nim`

Discord:
`https://discord.gg/nim`

Matrix:
`https://matrix.to/#/#nim-lang:matrix.org`

Telegram:
`https://t.me/nim_lang`

Twitter:
`https://twitter.com/nim_lang`

Reddit:
`https://www.reddit.com/r/nim`

StackOverflow:
`https://stackoverflow.com/questions/tagged/nim-lang`

Wiki:
`https://github.com/nim-lang/Nim/wiki`

1.3 Przygotowanie środowiska

Abyśmy mogli zacząć pisać i kompilować programy na naszym komputerze, będziemy potrzebowali:

- pakietu Nim
- kompilatora języka C/C++
- dowolnego/ulubionego edytora kodu

Nim nie dostarcza zintegrowanego środowiska programistycznego, dlatego wszystkie powyższe elementy będziemy musieli pobrać, zainstalować i skonfigurować samodzielnie.

W przypadku systemu operacyjnego Windows ze strony projektu należy pobrać archiwum zip Nima w wersji odpowiedniej do posiadanej architektury komputera. Analogicznie należy postąpić ze znajdującym się na stronie archiwum kompilatora języka C (mingw).

Chcąc zainstalować Nima na jednej z dystrybucji GNU/Linux można to zrobić na dwa sposoby. Możemy zainstalować pakiet Nim z repozytorium danej dystrybucji lub pobrać go ze strony projektu. W obu przypadkach dodatkowo będzie potrzebny kompilator C/C++. Warto wówczas zainstalować metapakiet o nazwie `build-essentials` zawierający niezbędne biblioteki wykorzystywane do pracy z językami C/C++.

1.4 Kompilacja

Najszybszym sposobem skompilowania przygotowanego pliku jest wykonanie polecenia:

```
nim c nazwa_pliku.nim
```

Po poprawnym skompilowaniu otrzymamy wykonywalny plik o takiej samej nazwie.

Ogólna składnia kompilacji programu jest następująca:
`nim polecenie [opcje] [nazwa_pliku] [argumenty]`

`polecenie` może przyjmować następujące wartości:

`compile, c`	kompiluje projekt z wykorzystaniem domyślnego generatora kodu C
`r`	kompiluje do $nimcache/projname i uruchamia z argumentami domyślnie używając kodu C
`doc`	generuje dokumentację dla pliku wejściowego

`opcje` mogą przyjmować następujące wartości:

`-p, --path:PATH`	dodaj ścieżkę do ścieżek wyszukiwania
`-d, --define:SYMBOL(:VAL)`	zdefiniuj symbol warunkowy
`-u, --undef:SYMBOL`	usuń definicję symbolu warunkowego
`-f, --forceBuild:on|off`	wymuś przebudowę wszystkich modułów
`--stackTrace:on|off`	włącz/wyłącz ślad stosu
`--threads:on|off`	włącz/wyłącz obsługę wielowątkowości
`-x, --checks:on|off`	włącz/wyłącz wszystkie kontrole środowiska uruchomieniowego
`-a, --assertions:on|off`	włącz/wyłącz asercje
`--opt:none|speed|size`	nie optymalizuj, optymalizuj pod względem szybkości lub czasu
`--debugger:native`	użyj natywnego debugera (gdb)
`--app:console|gui|lib|staticlib`	wygeneruj aplikację konsolową, graficzną, DLL lub bibliotekę statyczną
`-r, --run`	uruchom skompilowany program z podanymi argumentami
`--eval:cmd`	bezpośrednia ewaluacja kodu nim; np.: `nim --eval:"echo 1"`
`--fullhelp`	pokaż pełny podręcznik pomocy
`-h, --help`	pokaż podręcznik pomocy
`-v, --version`	pokaż szczegóły dot. zainstalowanej wersji

Przykładowa konstrukcja polecenia do skompilowania kodu zawartego w pliku `main.nim`:

```
nim c -r --opt:speed main.nim
```

Dzięki czemu kod (zapisany w pliku `main.nim`) najpierw zostanie skompilowany (`c`) z optymalizacją pod względem szybkości działania (`--opt:speed`), a następnie uruchomiony (`-r`).

1.5 Pierwszy program

Zazwyczaj podczas nauki programowania nowego języka, jako pierwszy, pisze się program wyświetlający napis `Hello world`. Więc tradycji niech stanie się zadość.

Do wyświetlania tekstu na ekranie służy polecenie `echo`. Konsekwencją jego działania, poza wyświetlaniem tekstu, jest przejście kursora do nowej linii.

```
echo "Hello world"
```

```
Hello world
_
```

Istnieją różne sposoby wyświetlania tekstu w konsoli:

```
echo "Hello world"
echo("Hello world")
"Hello world".echo()
"Hello world".echo
"Hello ".echo("world")
"Hello".echo " world"
```

Wszystkie powyższe przykłady wyświetlą napis `Hello world`, po czym kursor przejdzie do nowej linii.

Jeśli chcemy, aby po wyświetleniu tekstu kursor pozostał w tej samej linii, skorzystamy z `stdout.write`. Możemy i tu wymusić przejście do nowej linii używając `\n`

```
stdout.write "Hello world. "
stdout.write "Bye!\n"
```

```
Hello world. Bye!
_
```

Można oczywiście wyświetlać liczby lub wyniki operacji arytmetycznych. Liczby zmiennoprzecinkowe mają oddzieloną część całkowitą od dziesiętnej kropką.

```
echo 20
echo 3.14
echo 5+5
echo 6-2
echo 2*4
echo 15/2
```

```
20
3.14
10
4
8
7.5
```

W kodzie źródłowym można stosować komentarze. W Nim mamy do dyspozycji ich dwa rodzaje — jednolinijkowe, zaczynające się symbolem hash `#` i wielolinijkowe — tekst zapisany pomiędzy `#[` a `]#` lub pomiędzy `discard """` a `"""`. Poniższy przykład przedstawia wyżej opisane sposoby.

```
# To jest kometarz jednolinijkowy
echo "Hej!" # można też tak komentować
#[
    To jest
    kometarz
    wielolinijkowy
]#
discard """
    To także jest
    komentarz
    wielolinijkowy
"""
```

Po kompilacji i uruchomieniu pliku zostanie wyświetlony tylko napis "`Hej!`".

Edytory kodu zwykle posiadają własną opcję wstawiania komentarzy, np. generują znaki komentowania jednolinijkowego ale dla zaznaczonej części kodu. Taki sposób wydaje się najbardziej wygodny i szybki.

Podobnie, jak w innych językach programowania, także i w Nim można łączyć teksty (ciągi znaków). Takie łączenie nazywa się konkatenacją.

Gdy mamy przygotowane już napisy do połączenia, wówczas możemy użyć znaku `&`.

```
echo "Ala " & "ma " & "kota."
```

```
Ala ma kota.
_
```

Bezpośrednio można łączyć ze sobą tylko ciągi znaków. Jeśli mamy do złączenia ciąg znaków i liczbę, wówczas trzeba tę liczbę zrzutować na ciąg znaków. Do rzutowania danego typu danych na ciąg znaków użyjemy symbolu `$`, który wstawimy przed liczbą.

```
echo "Ala " & "ma " & $20 & " lat."
```

```
Ala ma 20 lat.
_
```

Można też to zrobić trochę szybciej, używając znaku `,`, który automatycznie zamienia wszystkie argumenty na ciągi znaków.

```
echo "Ala ", "ma ", 20, " lat."
```

```
Ala ma 20 lat.
_
```

Na tym etapie nauki mogę zasugerować przyjęcie jednego, wygodnego sposobu zarówno wyświetlania tekstów, tworzenia komentarzy, zwłaszcza wielolinijkowych, jak i łączenia tekstów. Pomoże to w utrzymaniu porządku w naszym kodzie i sprawi, że będzie on bardziej czytelny.

2 Zmienne i typy danych

2.1 Zmienne

Zmienne w programowaniu często porównuje się do pojemników, które przechowują różne wartości. Tymi wartościami może być pojedynczy znak, ciąg znaków, różnego rodzaju liczby, a nawet wartości logiczne — prawda, czy fałsz. Odwołując się do tych pojemników, możemy użyć wartości w nich przechowywanych.

W Nim przed użyciem zmiennej trzeba ją wcześniej zadeklarować. Do deklaracji zmiennej służy `var`. Spójrzmy na przykład:

```
var name = "Adam"
echo "Cześć! Mam na imię ", name, "."
```

```
Cześć! Mam na imię Adam.
_
```

Deklaracja zmiennej polega na określeniu nazwy, pod jaką będzie ona przechowywała wartości oraz typu tych danych. Nim „potrafi" rozpoznać, jakiego typu jest zmienna, którą przechowujemy. Jednak, gdy chcemy zadeklarować zmienną bez podawania jej wartości, musimy sami określić jej typ.

```
var age: int
```

Właśnie zadeklarowaliśmy zmienną o nazwie `age`, która będzie przechowywał liczby całkowite (`int`).

Nic nie stoi na przeszkodzie, żeby oprócz przypisania wartości do zmiennej, określij również jej typ:

```
var age: int = 20
```

Po deklaracji zmiennej, można zmieniać jej wartość, ale pamiętajmy, że nowe wartości muszą być tego samego typu.

```
var age: int = 20
echo age
age = 17
echo age
```

```
20
17
_
```

Jeśli deklarujemy kilka zmiennych, możemy uprościć zapis, stosując wcięcia (tabulator lub spacje):

```
var
    x = 10
    y = 3.5
    z = "Hello!"
```

Oprócz zmiennych w Nim możemy używać stałych. Służą one do przechowywania wartości, których, gdy zostaną już określone, nie będzie możliwości dokonania ich zmiany.

```
const pi = 3.14159
```

Proszę zwrócić uwagę, że **każda stała zadeklarowana w ten sposób musi mieć ustaloną wartość już podczas kompilacji programu.**

Podobnie, jak w przypadku zmiennych, dla deklaracji stałych także możemy przygotować deklarację blokową:

```
const
    pi = 3.14159
    c = 300_000_000
    text = "Hello world"
```

W powyższym przykładzie został użyty trochę inny, niż do tej pory sposób zapis liczb: `300_000_000`. Ułatwia on poprawne wpisywanie bardzo dużych (lub bardzo małych) wartości, ale nie zmienia on interpretacji tej liczby. Przykład:

```
echo 300_000_000
```

```
300000000
_
```

W Nim można również deklarować stałe, które nie muszą mieć ustalonych wartości podczas kompilacji. Deklarujemy je, używając `let`:

```
stdout.write("Podaj swoje imię: ")
let name = stdin.readLine()
echo name
```

W powyższym przykładzie użytkownik zostanie poproszony o podanie imienia, a następnie to imię wyświetli się na ekranie.

W Nim możemy tworzyć zmienne lub stałe, których nazwy składają się z kilku słów, pomiędzy, którymi może być wstawiona spacja. Musimy wówczas nazwę zmiennej zamknąć pomiędzy ` a `. Co więcej, odwołując się do nazw zmiennych lub stałych Nim nie uwzględnia wielkości znaków poza pierwszym znakiem. Proszę zwrócić uwagę na poniższy przykład.

```
let `my own const` = 12
echo `my own const`
echo myownconst
echo myOwnConst
```

```
12
12
12
_
```

Gdybyśmy jednak próbowali odwołać się do stałej o nazwie `MyOwnConst` otrzymalibyśmy błąd: `undeclared identifier: 'MyOwnConst'`.

2.2 Typy danych

W Nim możemy korzystać z następujących typów danych:

```
var c: char = 'y'          # pojedynczy znak
var name: string = "Adam" # ciąg znaków
var age: int = 10          # liczba całkowita
var pi: float = 3.14159    # liczba zmiennoprzecinkowa
var b: bool = true         # wartość logiczna
```

Podczas deklaracji zmiennej lub stałej, która będzie przechowywała pojedynczy znak, zamiast cudzysłowów używamy apostrofów.

Żeby sprawdzić typ danej zmiennej lub stałej korzystamy z atrybutu `type`. Oto typy danych dla zmiennych z powyższego przykładu:

```
echo c.type
echo name.type
echo age.type
echo pi.type
echo b.type
```

```
char
string
int
float
bool
_
```

W przypadku liczb całkowitych możemy dodatkowo określić zakres. Mamy do dyspozycji: `int64`, `int32`, `int16`, `int8`. Zatem możemy ustalić zakresy liczb całkowitych, kolejno: 64 bitowe, 32 bitowe, 16 bitowe i 8 bitowe.

Analogicznie mamy do dyspozycji liczby całkowite bez znaku (liczby dodatnie i 0): `uint64`, `uint32`, `uint16`, `uint8`.

Możemy sprawdzić, jak dużo miejsca w pamięci zajmuje stała lub zmienne. Wykorzystamy do tego operator `sizeof`. Wartość ta jest podawana w bajtach.

```
var a: uint64 = 10
var b: uint32 = 10
var c: uint16 = 10
var d: uint8 = 10

echo a.sizeof
echo b.sizeof
echo c.sizeof
echo d.sizeof
```

```
8
4
2
1
_
```

3 Praca z ciągami znaków