

# Podstawy programowania w Nim

Namysław Tatarynowicz

1 czerwca 2023

## Spis treści

<b>1</b>	<b>Wprowadzenie do języka Nim</b>	<b>2</b>
1.1	Wstęp . . . . .	2
1.2	Kilka słów o języku . . . . .	2
1.3	Instalacja środowiska . . . . .	3
1.4	Kompilacja . . . . .	3
1.5	Pierwszy program . . . . .	4
<b>2</b>	<b>Zmienne i typy danych</b>	<b>7</b>
2.1	Zmienne . . . . .	7
2.2	Typy danych . . . . .	7
<b>3</b>	<b>Instrukcje warunkowe</b>	<b>8</b>
<b>4</b>	<b>Pętle</b>	<b>9</b>
4.1	Pętla for . . . . .	9
4.2	Pętla while . . . . .	9
4.3	Instrukcja break . . . . .	9
4.4	Instrukcja continue . . . . .	9

# 1 Wprowadzenie do języka Nim

## 1.1 Wstęp

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.

W przypadku zauważenia błędu lub w przypadku jakiegokolwiek sugestii proszę o kontakt na adres mailowy: [n.tatarynowicz@gmail.com](mailto: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 Kilka słów o języku

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. Zatem, podczas kompilacji programu, Nim najpierw tworzy plik C i dopiero ten plik jest kompilowany.

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 takich miejsc.

irc: [#nim](https://libera.chat)

Discord: <https://discord.gg/nim>

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

Telegram: [https://t.me/nim\\_lang](https://t.me/nim_lang)

forum: <https://forum.nim-lang.org/>

twitter: [https://twitter.com/nim\\_lang](https://twitter.com/nim_lang)

blog: <https://nim-lang.org/blog.html>

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 Instalacja środowiska

Instalacja...

### 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:

<b>compile, c</b>	kompiluje projekt z wykorzystaniem domyślnego generatora kodu C
<b>r</b>	compile to \$nimcache/projname, run with [arguments] using backend specified by <code>-backend</code> (default: c)
<b>doc</b>	generuje dokumentację dla pliku wejściowego

`opcje` mogą przyjmować następujące wartości:

<code>-p, --path:PATH</code>	dodaj ścieżkę do ścieżek wyszukiwania
<code>-d, --define:SYMBOL(:VAL)</code>	define a conditional symbol (Optionally: Define the value for that symbol, see: "compile time define pragmas")
<code>-u, --undef:SYMBOL</code>	undefine a conditional symbol
<code>-f, --forceBuild:on off</code>	wymuś przebudowę wszystkich modułów
<code>--stackTrace:on off</code>	włącz/wyłącz ślad stosu
<code>--threads:on off</code>	włącz/wyłącz obsługę wielowątkowości
<code>-x, --checks:on off</code>	włącz/wyłącz wszystkie kontrole środowiska uruchomieniowego
<code>-a, --assertions:on off</code>	włącz/wyłącz asercje
<code>--opt:none speed size</code>	nie optymalizuj, optymalizuj pod względem szybkości lub czasu
<code>--debugger:native</code>	użyj natywnego debugera (gdb)
<code>--app:console gui lib staticlib</code>	wygeneruj aplikację konsolową, graficzną, DLL lub bibliotekę statyczną
<code>-r, --run</code>	uruchom skompilowany program z podanymi argumentami
<code>--eval:cmd</code>	evaluate nim code directly; e.g.: nim -eval:"echo 1" defaults to e (nimscrip) but customizable: nim r -eval:'for a in stdin.lines: echo a'
<code>--fullhelp</code>	pokaż pełny podręcznik pomocy
<code>-h, --help</code>	pokaż podręcznik pomocy
<code>-v, --version</code>	pokaż szczegóły dot. zainstalowanej wersji

## 1.5 Pierwszy program

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 ":)"
"Hej!".echo
"Hej! Hej!".echo()
"Hej! ".echo "Ho!"
echo("Hello world")
("Hello world!").echo
```

```
:)
Hej!
Hej! Hej!
Hej! Ho!
Hello world
Hello world!
-
```

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"
```

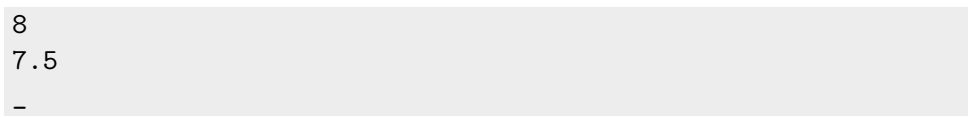
```
Hello world. Bye!
```

```
-
```

Można oczywiście wyświetlać liczby lub wyniki operacji arytmetycznych. Liczby zmiennoprzecinkowe mają oddzielną 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
```



## 2 Zmienne i typy danych

### 2.1 Zmienne

### 2.2 Typy danych



### 3 Instrukcje warunkowe

## 4 Pętle

### 4.1 Pętla for

### 4.2 Pętla while

### 4.3 Instrukcja break

### 4.4 Instrukcja continue