

---

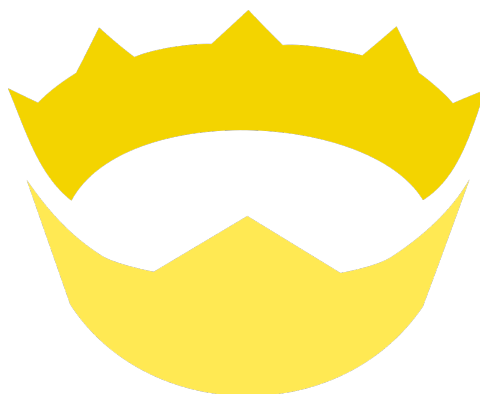
# Podstawy programowania w Nim

## podręcznik użytkownika

---

Namysław Tatarynowicz

5 czerwca 2023



## Spis treści

<b>1</b>	<b>Wprowadzenie do języka Nim</b>	<b>2</b>
1.1	Wstęp . . . . .	2
1.2	Poznajmy Nima . . . . .	2
1.3	Przygotowanie środowiska . . . . .	4
1.4	Kompilacja . . . . .	4
1.5	Pierwszy program . . . . .	6
<b>2</b>	<b>Zmienne i typy danych</b>	<b>9</b>
2.1	Zmienne . . . . .	9
2.2	Typy danych . . . . .	9
2.2.1	Typ tekstowy . . . . .	9
2.2.2	Typy liczbowe . . . . .	9
<b>3</b>	<b>Instrukcje warunkowe</b>	<b>10</b>
<b>4</b>	<b>Pętle</b>	<b>11</b>
4.1	Pętla for . . . . .	11
4.2	Pętla while . . . . .	11
4.3	Instrukcja break . . . . .	11
4.4	Instrukcja continue . . . . .	11

# 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++. Zatem, podczas kompilacji programu, najpierw powstaje plik C/C++ i dopiero ten plik jest kompilowany. Ostatecznie Nim daje możliwość kompilacji do C, C++ lub JavaScript.

Podobnie, jak 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.

**irc:**

`irc.libera.chat, #nim`

**Discord:**

`https://discord.gg/nim`

**Matrix:**

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

**Telegram:**

`https://t.me/nim\_lang`

**forum:**

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

**twitter:**

`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 Przygotowanie środowiska

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

- pakietu Nim
- kompilatora języka 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. 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:

<b>compile, c</b>	kompiluje projekt z wykorzystaniem domyślnego generatora kodu C
<b>r</b>	kompiluje do \$nimcache/projname i uruchamia z argumentami domyślnie używając kodu 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

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

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

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



Podobnie, jak w innych językach programowania, także i w Nim można łączyć teksty (ciągi znaków) — nazywa się to 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 `$`.

```
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 już 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

### 2.2 Typy danych

#### 2.2.1 Typ tekstowy

#### 2.2.2 Typy liczbowe

### **3 Instrukcje warunkowe**

## 4 Pętle

### 4.1 Pętla for

### 4.2 Pętla while

### 4.3 Instrukcja break

### 4.4 Instrukcja continue