Podstawy programowania w Nim

Namysław Tatarynowicz

2 czerwca 2023

Spis treści

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 jakiejkolwiek sugestii 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 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 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 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:

`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)`	define a conditional symbol (Optionally: Define the value for that symbol, see: "compile time define pragmas")
`-u, --undef:SYMBOL`	undefine a conditional symbol
`-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`	evaluate nim code directly; e.g.: nim –eval:"echo 1" defaults to e (nimscript) but customizable: nim r –eval:'for a in stdin.lines: echo a'
`--fullhelp`	pokaż pełny podręcznik pomocy
`-h, --help`	pokaż podręcznik pomocy
`-v, --version`	pokaż szczegóły dot. zainstalowanej wersji

Przykład konstrukcji 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 `]#`. Poniższy przykład przedstawia oba te sposoby.

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

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

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