
AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

# Opracowanie i implementacja języka programowania z wbudowanym mechanizmem wzorca obserwatora

**Robert Cebula**
**Wydział Fizyki i Informatyki Stosowanej**

**Kraków, Styczeń 2017**

# Cele oraz motywy pracy

- Stworzenie w pełni funkcjonalnego języka programowania
- Zaprojektowanie i napisanie wszystkiego od zera (m.in. lekser, parser)
- Lepsze poznanie codziennego narzędzia pracy programisty
- Który informatyk nie chciałby stworzyć własnego języka programowania?

# Główne założenia języka

- Nazwa języka to BIO (built-in observer)
- Wbudowany mechanizm pozwalający implementować programy oparte o wzorzec obserwatora
- Dynamicznie typowany
- Syntaktycznie wszystko jest funkcją (także konstrukty takie jak pętla FOR czy instrukcja warunkowa IF)
- Zorientowany na programowanie proceduralne (możliwość definiowania własnych funkcji)
- Błędy są wartościami

# Główne założenia implementacji

- Podział na dwa programy: kompilator i interpreter
- Napisane w Javie w wersji 1.8
- Skierowana głównie na system operacyjny Linux. Powinno działać także pod Windowsem i Mac Osem (dzięki wykorzystaniu Javy)

# Podstawowa semantyka

- Przypisanie

```
AS_LOC(a, 10)
AS_GLOB(a, "text")
PRINTLN(a)
PRINTLN(GET_GLOB(a))
```

- Instrukcja warunkowa IF

```
IF(flag, PRINTLN("true"), PRINTLN("false"))
```

- Pętla FOR

```
FOR
(
  AS_LOC(i, 0),
  LS(i, 10),
  PRINTLN(i),
  INC(i)
)
```

- Definicja funkcji

```
def fun_name(arg1, arg2, arg3=10)
   PRINTLN(arg1)
   PRINTLN(ADD(arg2, arg3))
end
```

# Podstawowa semantyka (2)

- Wywołanie funkcji

```
fun_name("sum: ", arg3=10, arg2=-20.5)
```

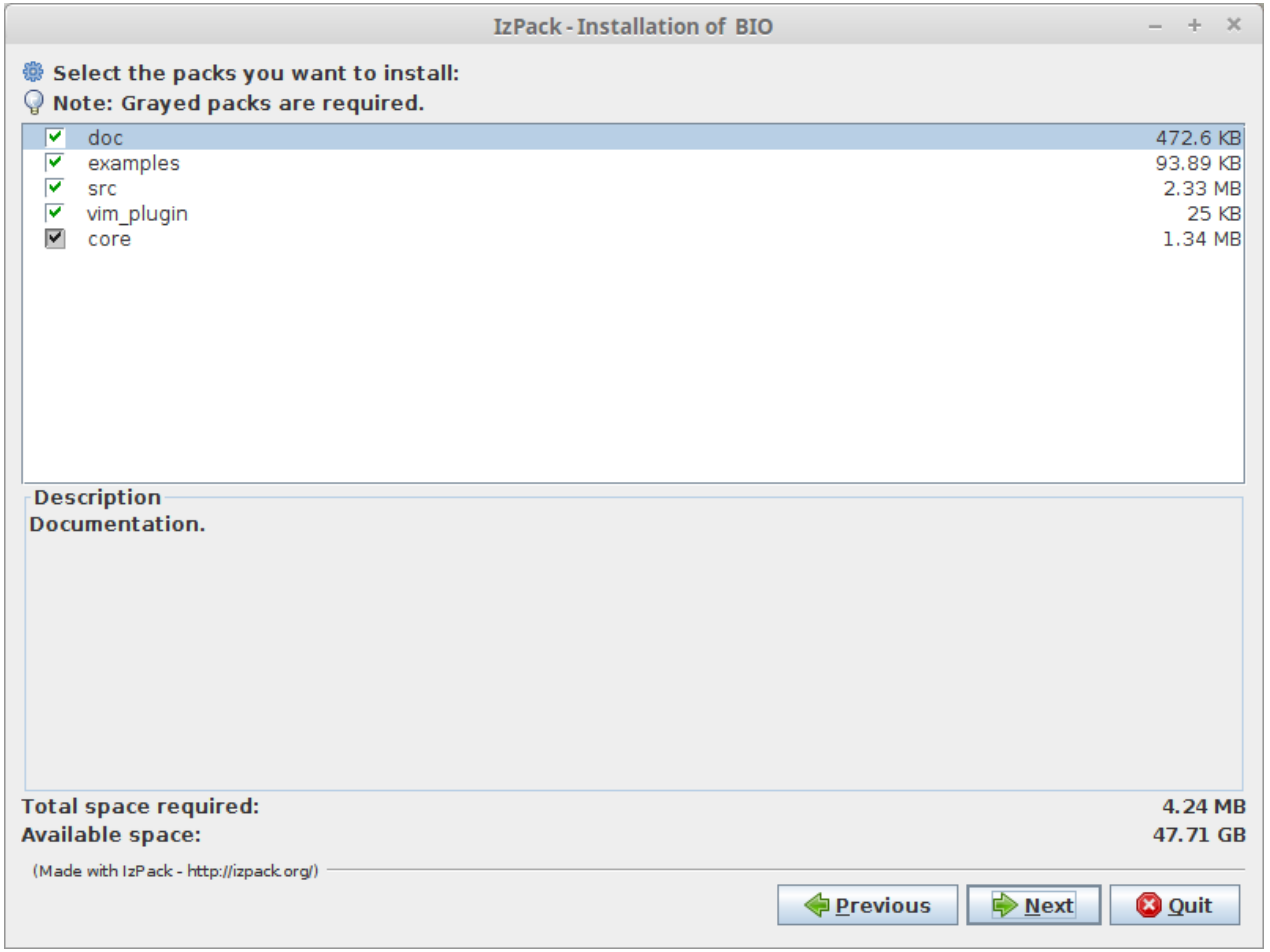
- Wzorzec obserwatora

```
def some_event()
   DN()
end

def on_some_event()
   DN()
end

def onSTART()

   ATTACH_TO_EVENT(some_event, on_some_event)
   IS_ATTACHED(some_event, on_some_event)
   DETACH_FROM_EVENT(some_event, on_some_event)

end
```

# Instalacja

- Gotowy instalator
- Dla systemu linux kopiuje pliki do wybranego przez użytkownika katalogu i tworzy skrypty startowe w katalogu /usr/local/bin/
- Dodatkowo możemy wybrać opcję zainstalowania pluginu, który umożliwia kolorowanie składni języka BIO dla programu vim

# Instalacja (2)



# Przykład użycia

## Server.bio:

```
#IMPORT("tcp")

def msg_recv(msg)
   DN()
end

def on_msg_recv_1(msg)
   PRINTLN({ "on_msg_recv_1: " + msg })
end

def on_msg_recv_2(msg)
   PRINTLN({ "on_msg_recv_2: " + msg })
end

def onSTART()
   @ przypisz funkcje obserwatorów do funckji msg_recv
   ATTACH_TO_EVENT(msg_recv, on_msg_recv_1, msg_recv, on_msg_recv_2)
   @ nasłuchuj połączenia
   AS_LOC(conn, TCP_LISTEN(5500))
   @ odbierz stringa
   msg_recv(TCP_RECV_STR(conn))
   @ zamknij połączenie
   TCP_CLOSE(conn)
end
```

## Client.bio:

```
#IMPORT("tcp")

def onSTART()
   @ połącz się z serwerem
   AS_LOC(conn, TCP_CONNECT("127.0.0.1", 5500))
   @ wyślij mu stringa wczytanego z konsoli
   TCP_SEND(conn, INPUT("Message to server: "))
   @ zamknij połączenie
   TCP_CLOSE(conn)
end
```

# Przykład użycia (2)

Server:

```
robert@robert-desktop ~/programowanie/BIO/bsc_thesis/presentation_example $ bioc server.bio -o server.cbio
robert@robert-desktop ~/programowanie/BIO/bsc_thesis/presentation_example $ bio server.cbio
on_msg_recv_1: some text 1234.24
on_msg_recv_2: some text 1234.24
robert@robert-desktop ~/programowanie/BIO/bsc_thesis/presentation_example $
```

Client:

```
robert@robert-desktop ~/programowanie/BIO/bsc_thesis/presentation_example $ bioc client.bio -o client.cbio
robert@robert-desktop ~/programowanie/BIO/bsc_thesis/presentation_example $ bio client.cbio
Message to server: some text 1234.24
robert@robert-desktop ~/programowanie/BIO/bsc_thesis/presentation_example $
```

# Co udało się zrealizować

- Wszystkie podstawowe założenia (wbudowany obserwator, dynamicznie typowany, syntaktycznie wszystko jest funkcją, definiowanie własnych funkcji, błędy są wartościami)
- Preprocesor z dwoma dyrektywami (#INCLUDE i #DEFINE)
- Wbudowany parser matematyczno-logiczny
- Dynamiczne struktury (jak w języku MATLAB)
- Argumenty domyślne i nazywane funkcji
- Optymalizacja kodu pośredniego
- Rozbudowana, dobrze udokumentowana biblioteka standardowa
- I wiele więcej