

# Projekt 3. Programowanie Systemowe - Tłumaczenie Basha na C

## Systemy Operacyjne

Marcin Klimek, Kacper Kosmal

25 stycznia 2024



**Politechnika Krakowska**  
Wydział Inżynierii  
Elektrycznej i Komputerowej

# Spis treści

<b>1</b>	<b>Wstęp</b>	<b>1</b>
<b>2</b>	<b>Funkcjonalność Programu</b>	<b>1</b>
2.1	Ograniczenia . . . . .	1
<b>3</b>	<b>Kod programu</b>	<b>1</b>
<b>4</b>	<b>Działanie Programu</b>	<b>6</b>
4.1	Przykład działania . . . . .	6
<b>5</b>	<b>Podsumowanie</b>	<b>9</b>

# 1 Wstęp

Projekt prezentuje skrypt Bash, który konwertuje skrypt Bash na program w języku C. Celem tego projektu jest stworzenie narzędzia, które ułatwi przenoszenie prostych skryptów Bash do programów C.

## 2 Funkcjonalność Programu

Program jest w stanie przetłumaczyć podstawowe konstrukcje języka Bash, w tym:

- Deklaracje zmiennych
- Instrukcje warunkowe
- Pętle
- Instrukcje echo
- Proste operacje arytmetyczne

Skrypt jest przydatny do szybkiego prototypowania i przekształcania prostych skryptów Bash, które nie wykorzystują zaawansowanych funkcji lub zewnętrznych poleceń.

### 2.1 Ograniczenia

Skrypt ma kilka ograniczeń, nie obsługuje całości składni bash, w tym funkcji. W takim wypadku program wykrywa błąd, informuje o nim użytkownika, oraz kończy działanie

## 3 Kod programu

Skrypt składa się z kilku kluczowych części:

- Funkcje do tłumaczenia linii kodu Bash na C
- Obsługa specjalnych przypadków składni Bash, takich jak pętle, instrukcje warunkowe, echo i operacje arytmetyczne
- Główna pętla, która czyta skrypt Bash i tłumaczy go na C

Każda część skryptu skupia się na konkretnym zadaniu, co ułatwia zrozumienie i modyfikację kodu.

```
#!/bin/bash

# Zadeklaruj tablicę asocjacyjną do śledzenia zadeklarowanych zmiennych
declare -A declared_variables

# Funkcja do obsługi zastępowania zmiennych
variable_replacement() {
    local variable_name
    local variable_value
    local og_match

    # Wyrażenie regularne do dopasowywania przypisań zmiennych w bashu
    if [[ $line =~ ([[:alnum:]]+)([[:space:]]*)=([[:space:]]*)([^\;]+)(;|)$ ]]; then
        og_match="${BASH_REMATCH[0]}"
        variable_name="${BASH_REMATCH[1]}"
        variable_value="${BASH_REMATCH[4]}"
    else
        return
    fi
}
```

```

# Usuń ';' na końcu, jeśli jest obecne
variable_value="${variable_value%;}";

# Określ typ zmiennej na podstawie wartości
local variable_type=""
if [[ $variable_value =~ ^[+-]?[0-9]+\.[0-9]*$ ]]; then
    if [[ $variable_value =~ ^[+-]?[0-9]+$ ]]; then
        variable_type="int"
    else
        variable_type="double"
    fi
else
    variable_type="const char*"
fi

# Sprawdź, czy nazwa zmiennej jest pusta
if [[ -z "$variable_name" ]]; then
    echo "Error: nazwa zmiennej jest pusta"
    exit 1
fi

# Sprawdź, czy zmienna była wcześniej zadeklarowana
local was_there=${declared_variables[$variable_name]+_}
if [[ -n "$was_there" ]]; then
    line="${variable_name}=${variable_value};"
else
    declared_variables[$variable_name]=$variable_type
    substitute="${variable_type} ${variable_name}=${variable_value};"
    line="${line/$og_match/$substitute}"
fi
}

# Tabela przekierowań do formaterów typu printf
declare -A type_formatter_lookup
type_formatter_lookup["int"]="d"
type_formatter_lookup["double"]="f"
type_formatter_lookup["const char*"]="s"

# Flaga wskazująca zakończenie tłumaczenia echo
echo_done=false

# Funkcja do tłumaczenia instrukcji echo
echo_translation() {
    local echo_content

    # Wyrażenie regularne do dopasowywania instrukcji echo
    if [[ $line =~ ^echo[:space:]+(.*)$ ]]; then
        echo_content="${BASH_REMATCH[1]}"
        echo_content="${echo_content%\"}"
        echo_content="${echo_content#\"}"
    fi
}

```

```

local printf_command='printf("'
local printf_arguments=()

# Przetwarzanie każdej zmiennej w instrukcji echo
while [[ $echo_content =~ (\${a-zA-Z_}[a-zA-Z0-9_]*) ]]; do
    local var_name="${BASH_REMATCH[1]}"
    local var_name_stripped="${var_name:1}"
    local var_type=${declared_variables[$var_name_stripped]}

    printf_command+="${echo_content%%"$var_name"*}"
    printf_command+="${type_formatter_lookup[$var_type]}"
    printf_arguments+=("$var_name_stripped")
    echo_content="${echo_content#"${var_name}"}"
done

printf_command+='\n'

if [[ ${#printf_arguments[@]} -gt 0 ]]; then
    IFS=","
    printf_command+=", ${printf_arguments[*]}"
    unset IFS
fi

printf_command+=");"
line="$printf_command"
echo_done=true
fi
}

# Funkcja do tłumaczenia pętli for-range
range_for_translation() {
    local variable_name
    local start
    local end

    # Wyrażenie regularne do dopasowywania pętli for-range
    if [[ $line =~ for[[:space:]]+([[:alnum:]]_)+[[:space:]]+in[[:space:]]+
        +\{([0-9]+\.\.([0-9]+\)\}\);([[:space:]]*)(\{)? ]]; then
        variable_name="${BASH_REMATCH[1]}"
        start="${BASH_REMATCH[2]}"
        end="${BASH_REMATCH[3]}"
        bracket="${BASH_REMATCH[5]}"

        line="for (int ${variable_name} = ${start}; ${variable_name} <= ${end};
            ${variable_name}++) ${bracket}"
    fi
}

# Funkcja do tłumaczenia linii skryptu bash na C
translate_line() {
    line="$1"

```

```

# Obsłuż komentarz: zamień # na // i nie tłumacz niczego po #
if [[ "$line" =~ ^# ]]; then
    line=${line/#\#//}
    return
fi

# Sprawdź czy linia jest wspierana: jeśli jest to funkcja (wykrywana po słowie
# kluczowym function oraz nawiasach klamrowych), to napisz że skrypt nie jest
# wspierany
if [[ "$line" =~ function ]]; then
    echo "Error: skrypt zawiera funkcje, które nie są wspierane"
    exit 1
fi

# Wykryj () w nazwie funkcji
if [[ "$line" =~ [a-zA-Z_][a-zA-Z0-9_]*\(\) ]]; then
    echo "Error: skrypt zawiera funkcje, które nie są wspierane"
    exit 1
fi

# Tłumaczenie instrukcji echo
echo_translation

# Zamienia 'done' na '}'
line=$(echo "$line" | sed -E 's/^done$/}/')
# Zamienia 'do' na '{'
line=$(echo "$line" | sed -E 's/^do$/{/')
# Przetwarza pętle for, zamieniając składnię Bash na C
line=$(echo "$line" | sed -E 's/^for \(\((.*)\)\);?/for (\1)/')

# Arytmetyka: let "VAR=wyrażenie"
# Zamienia składnię let na przypisanie w C
line=$(echo "$line" | sed -E 's/^let "(\w+)=(.*)"$/\1 = \2;/')

# Przypisanie zmiennej: VAR=wartość
# tylko działa, jeśli echo_done jest fałszywe
if [[ $echo_done == false ]]; then
    variable_replacement
fi
echo_done=false

# Arytmetyka: $(wyrażenie)
# Zamienia składnię $(wyrażenie) na (wyrażenie); dla wyrażeń
# zakończonych średnikiem
line=$(echo "$line" | sed -E 's/\$\(\((.*)\)\)\$/(\1);/')
# Zamienia składnię $(wyrażenie) na (wyrażenie) dla pozostałych przypadków
line=$(echo "$line" | sed -E 's/\$\(\((.*)\)\)\$/(\1)/')
# Zamienia 'if [ warunek ]; then' na 'if (warunek) {'
line=$(echo "$line" | sed -E 's/^if \[ (.*) \]; then$/if (\1) {/')
# Zamienia 'fi' na '}'
line=$(echo "$line" | sed -E 's/^fi$/}/')

```

```

# Zamienia '-eq' na '=='
line=$(echo "$line" | sed 's/-eq==/g')
# Zamienia '-ne' na '!='
line=$(echo "$line" | sed 's/-ne!=/g')
# Zamienia '-lt' na '<'
line=$(echo "$line" | sed 's/-lt/</g')
# Zamienia '-le' na '<='
line=$(echo "$line" | sed 's/-le/<=/g')
# Zamienia '-gt' na '>'
line=$(echo "$line" | sed 's/-gt/>/g')
# Zamienia '-ge' na '>='
line=$(echo "$line" | sed 's/-ge/>=/g')
range_for_translation
# Zamienia 'while [ warunek ];' na 'while (warunek)'
line=$(echo "$line" | sed -E 's/while \[ (.*) \];?/while (\1)/')
# Usuwa '$' przed nazwą zmiennej
line=$(echo "$line" | sed -E 's/\$(\w+)/\1/g')

# Wypisz przetłumaczoną linię
echo "$line" >>"$2"
}

input_file="$1"
output_file="$2"

# Wyczyść plik wynikowy
>"$output_file"

echo "#include <stdio.h>" >>"$output_file"
echo "int main() {" >>"$output_file"

# pętli przetwarzaj każdą linię i przetłumacz ją
while read -r line; do
    # Jeśli linia jest shebang, zignoruj ją
    if [[ "$line" =~ ^#! ]]; then
        continue
    fi

    # Przetłumacz linię i dodaj ją do pliku wynikowego
    translate_line "$line" "$output_file"
done <"$input_file"

echo "}" >>"$output_file"

```

Kod używa wyrażeń Regex aby dokonywać tłumaczenia. Typy zmiennych są pamiętane w tablicy asocjacyjnej, aby zapewnić poprawne tłumaczenie zmiennych. W programie można wyróżnić następujące funkcje:

- `variable_replacement` - funkcja do obsługi zastępowania zmiennych
- `echo_translation` - funkcja do tłumaczenia instrukcji `echo`
- `range_for_translation` - funkcja do tłumaczenia pętli `for-range` (`for i in {1..5}; do`)
- `translate_line` - funkcja do tłumaczenia linii skryptu `bash` na `C`, wywoływana dla każdej linii

skryptu

Plik wejściowy jest odczytywany linia po linii, a następnie przekazywany do funkcji `translate_line`, która przetwarza linię i zapisuje ją do pliku wyjściowego.

## 4 Działanie Programu

Program można uruchomić z linii poleceń, podając jako argumenty ścieżki do plików wejściowego i wyjściowego.

```
./bash_to_c.sh input.sh output.c
```

### 4.1 Przykład działania

Dla testu podany zostaje plik `in.sh`:

```
#!/bin/bash

# Inicjalizacja zmiennych
limit=10
suma=0
i=0

echo "Obliczanie sumy liczb parzystych do $limit..."

# Używanie pętli while do sumowania liczb parzystych
while [ $i -le $limit ]; do
    # Wyrażenie matematyczne do sprawdzania, czy liczba jest parzysta
    if [  $((i \% 2)) -eq 0$  ]; then
        # Wyrażenie matematyczne do sumowania
        suma=$((suma + i))
        echo "Dodano $i do sumy, aktualna suma: $suma"
    fi
    i=$((i + 1))
done

echo "Suma liczb parzystych do $limit wynosi: $suma"

echo "Wyświetlanie sekwencji od 1 do 5 za pomocą pętli for..."

# Używanie pętli for do wyświetlania sekwencji
for ((j = 1; j <= 5; j++)); do
    echo "Numer sekwencji: $j"
done

echo "Wyświetlanie zakresu od 1 do 5 za pomocą pętli for range..."

# Zakres for
for i in {1..5}; do
    echo "Numer zakresu: $i"
done
```



```

echo "Demonstracja obsługi liczb niecałkowitych..."

# Testowanie liczb
a=1
b=2
c=3
echo "a=$a, b=$b, c=$c"

suma=$((a + b + c))
echo "Próba sumowania $a, $b i $c (Uwaga: Bash nie obsługuje liczb
      zmiennoprzecinkowych): $suma"

echo "Demonstracja obsługi łańcuchów znaków..."

znaki="test"
echo "znaki=$znaki"

```

Jego uruchomienie daje następujący wynik:

```

Obliczanie sumy liczb parzystych do 10...
Dodano 0 do sumy, aktualna suma: 0
Dodano 2 do sumy, aktualna suma: 2
Dodano 4 do sumy, aktualna suma: 6
Dodano 6 do sumy, aktualna suma: 12
Dodano 8 do sumy, aktualna suma: 20
Dodano 10 do sumy, aktualna suma: 30
Suma liczb parzystych do 10 wynosi: 30
Wyświetlanie sekwencji od 1 do 5 za pomocą pętli for...
Numer sekwencji: 1
Numer sekwencji: 2
Numer sekwencji: 3
Numer sekwencji: 4
Numer sekwencji: 5
Wyświetlanie zakresu od 1 do 5 za pomocą pętli for range...
Numer zakresu: 1
Numer zakresu: 2
Numer zakresu: 3
Numer zakresu: 4
Numer zakresu: 5
Demonstracja obsługi liczb niecałkowitych...
a=1, b=2, c=3
Próba sumowania 1, 2 i 3 (Uwaga: Bash nie obsługuje liczb zmiennoprzecinkowych): 6
Demonstracja obsługi łańcuchów znaków...
znaki=test

```

Po uruchomieniu skryptu `bash_to_c.sh` z plikiem `in.sh` jako argumentem wejściowym, zostanie wygenerowany plik `out.c` (plik został sformatowany dla czytelności):

```

#include <stdio.h>
int main() {
    int limit = 10;
    int suma = 0;
    int i = 0;

```

```

printf("Obliczanie sumy liczb parzystych do %d\n", limit);

while (i <= limit) {
    if ((i % 2) == 0) {
        suma = (suma + i);
        printf("Dodano %d do sumy, aktualna suma: %d\n", i, suma);
    }
    i = (i + 1);
}

printf("Suma liczb parzystych do %d wynosi: %d\n", limit, suma);

for (int j = 1; j <= 5; j++) {
    printf("Numer sekwencji: %d\n", j);
}

for (int i = 1; i <= 5; i++) {
    printf("Numer zakresu: %d\n", i);
}

int a = 1;
int b = 2;
int c = 3;
printf("a=%d, b=%d, c=%d\n", a, b, c);

suma = (a + b + c);
printf(
    "Próba sumowania %d, %d i %d (Uwaga: Bash nie obsługuje liczb "
    "zmiennoprzecinkowych): %d\n",
    a, b, c, suma);

const char* znaki = "test";
printf("znaki=%s\n", znaki);
}

```

Co daje następujący wynik:

```

Obliczanie sumy liczb parzystych do 10
Dodano 0 do sumy, aktualna suma: 0
Dodano 2 do sumy, aktualna suma: 2
Dodano 4 do sumy, aktualna suma: 6
Dodano 6 do sumy, aktualna suma: 12
Dodano 8 do sumy, aktualna suma: 20
Dodano 10 do sumy, aktualna suma: 30
Suma liczb parzystych do 10 wynosi: 30
Numer sekwencji: 1
Numer sekwencji: 2
Numer sekwencji: 3

```

```
Numer sekwencji: 4
Numer sekwencji: 5
Numer zakresu: 1
Numer zakresu: 2
Numer zakresu: 3
Numer zakresu: 4
Numer zakresu: 5
a=1, b=2, c=3
Próba sumowania 1, 2 i 3 (Uwaga: Bash nie obsługuje liczb zmiennoprzecinkowych): 6
znaki=test
```

Jak widać, program działa poprawnie, tłumacząc skrypt Bash na program C.

## 5 Podsumowanie

Projekt ten demonstruje, jak można użyć Bash i wyrażeń regularnych do stworzenia narzędzia do konwersji kodu. Jest to przykład prostego transpilatora, który może być używany do szybkiego prototypowania i przekształcania prostych skryptów Bash w programy C.