# Translacja z Pythona – dokumentacja wstępna

*Autor: Jakub Ficek*

## Temat projektu

Celem projektu jest stworzenie translatora skrośnego dla wybranego podzbioru języka Python do języka C++. Program będzie napisany w języku Python3.

## Założenia

- Obsługiwanymi typami są int, float, bool
- Wszystkie zmienne są statycznie typowane (zmienna nie może być najpierw typu int a potem float)
- Obsługa własnych funkcji z adnotowanymi typami argumentów i zwracanej wartości
- Obsługa instrukcji warunkowych if elif else
- Obsługa pętli while
- Obsługa wyrażeń matematycznych i logicznych
- Obsługa wyjścia przy pomocy funkcji print

## Przykłady

```
x = 5
while x > 0:
        if x % 3 == 0:
                print(0)
        elif x % 3 == 1:
                print(1)
        else:
                print(2)
        x = x - 1
```

```
#include <iostream>


using namespace std;

int main()
{
        int x;
        x = 5;
        while(x > 0)
        {
                if(x % 3 == 0)
                {
                        cout << 0 << endl;
                }
                else if(x % 3 == 1)
                {
                        cout << 1 << endl;
                else
                {
                        cout << 0 << endl;
                }
                x = x - 1;
        }
        return 0;
}
```

```
def sum(a: int, b: int) -> int:
        return a + b

x = 2

while x > 0 and 1 == 1:
        x = x - 1
        print(x)
y = 3 + 2 * 2
print(sum(x, y))
```

```
#include <iostream>

using namespace std;

int sum(int a, int b)
{
        return a + b;
}

int main()
{
        int x;
        int y;
        x = 2;
        while(x > 0 && 1 == 1)
        {
                x = x - 1;
                cout << x << endl;
        }
        y = 3 + 2 * 2;
        cout << sum(x, y) << endl;
        return 0;
}
```

```
x=2
x++
```

```
error: line 2
```

## Lista zdefiniowanych tokenów

'def' - DEF,
'if' - IF,
'elif' - ELIF,
'else' - ELSE,
'while' - WHILE,
'None' - NONE,
'int' - INT,
'float' - FLOAT,
'bool' - BOOL,
'return' - RETURN,
'print' - PRINT,
':' - COLON,
',' - COMMA,
'->' - RETURN_TYPE,
'\+' - PLUS,
'-' - MINUS,
'*' - MULTIPLY,
'/' - DIVIDE,
'\(' - LP,
'\)' - RP,
'%' - MODULO,
'==' - ISEQUAL,

'!=' - ISNOTEQUAL,
'<=' - ISEQUALLESS,
'<' - ISLESS,
'>=' - ISEQUALLESS,
'>' - ISMORE,
'=' - EQUALS,
'and' - AND,
'or' - OR,
'not' - NOT,
'\d+\.\d+' - VALUE_FLOAT,
'\d+' - VALUE_INT,
'True|False' - VALUE_BOOL,
'[a-zA-Z_][a-zA-Z0-9_]*' - IDENTIFIER
wzrost liczby tabulacji w nowej linii – INDENT
zmniejszenie liczby tabulacji w nowej linii – DEDENT
taka sama liczba tabulacji w nowej linii – NEWLINE

## Gramatyka

**program** = statements

**statements** = statement | statement statements

**statement** = assignment_statement | function_statement | return_statement | while_statement | if_statement | print_statement | expression_statement | NEWLINE | INDENT | DETEND

**assignment_statement** = IDENTIFIER ISEQUAL expression_statement

**function_statement** = DEF IDENTIFIER LP [IDENTIFIER COLON type] { COMMA IDENTIFIER' COLON type } RP RETURN_TYPE (type | NONE)  COLON INDENT

**return_statement** = RETURN expression_statement

**while_statement** = WHILE expression_statement COLON INDENT

**if_statement** = IF expression_statement COLON INDENT [el_statement]

**el_statement** = else_statement | elif_statement

**elif_statement** = ELIF expression_statement COLON INDENT [el_statement]

**else_statement** = ELSE COLON INDENT statement_block

**print_statement** = PRINT LP [expression_statement] {COMMA expression_statement } RP

**expression_statement** = func_call | operation

**operation** = { unary_op | unary_logic_op } expression_statement { binary_op | binary_logic_op |comparison_op expression_statement }

**func_call** = IDENTIFIER LP [identifier | value] { COMMA (identifier | value) } RP

**unary _op** = PLUS | MINUS

**unary_logic_op** = NOT

**binary_op** = PLUS | MINUS | MULTIPLY | DIVIDE | MODULO

**binary_logic_op** = AND | OR

**comparison_op** = ISEQUAL | ISNOTEQUAL | ISLESS | ISEQUALLESS | ISMORE | ISEQUALMORE

**type** = INT |FLOAT | BOOL

**value** = VALUE_INT | VALUE_FLOAT | VALUE_BOOL

# Sposób uruchomienia

Program z kodem napisanym w języku Python, jako wyjście program wypisuje lub tworzy plik z analogicznym kodem napisanym w C++ albo wypisuje komunikat o odpowiednim błędzie (błąd może wystąpić na etapie analizy leksykalnej lub składniowej)

# Moduły

- **Moduł obsługi plików** – odpowiedzialny za wczytywanie tekstu z pliku i tworzenie pliku wyjściowego
- **Lekser** – odpowiedzialny za rozbicie tekstu wczytanego z pliku na tokeny, które trafiają do analizatora składniowego.
- **Parser** – odpowiedzialny za sprawdzenie czy wczytany ciąg tokenów jest zgodny z gramatyką zdefiniowanego podzbioru oraz zbudowanie drzewa rozbioru.
- **Generator kodu** – odpowiedzialny za konstrukcję kolejnych instrukcji w C++, przechodząc drzewo rozbioru oraz korzystając z tablicy symboli, aby zainicjalizować wszystkie zmienne
- **Moduł obsługi błędów** – odbiera błędy z leksera, parsera oraz prezentuje błędy w czytelny sposób (numer linii)

  Dodatkowe struktury danych:

- **Tablica symboli** – zawierająca używane w kodzie wejściowym identyfikatory oraz informacje o ich typie i zasięgu, uzupełniana przez parser
- **Tablica akceptowalnych tokenów** – predefiniowana tablica wszystkich zdefiniowanych symboli, używana przez lekser

# Reguły translacji

- Stała struktura kodu w C++, pojawiająca się zawsze:

```
#include <iostream>

using namespace std;

//definicje funkcji

int main()

{

    // inicjalizacja zmiennych o zasięgu globalnym

    // translacja pythonowych instrukcji
```

```
    return 0;
}
```

- dla DETEND i NEWLINE dodawany jest średnik i znak nowej linii
- Na początku funkcji inicjowane są wszystkie zmienne lokalne
- Zamiana INT, FLOAT, BOOL, NONE na kolejno ‘int’, ‘float’, ‘bool’, ‘void’
- Zamiana funkcji postaci ‘def func(arg1: type1, arg2:type2) -> type3:’ na ‘type3 func(type1 arg1, type2 arg2)’
- INDENT oznaczane przez ‘{‘, a DEDENT przez ‘}’
- Zamiana ‘if expression:’ na ‘if(expression)’
- Zamiana ‘elif expression:’ na ‘else if(expression)’
- Zamiana ‘else:’ na ‘else’
- Zamiana ‘while expression:’ na ‘while(expression)’
- Operatory porównania, przypisania oraz arytmetyczne pozostają bez zmian
- Zamiana operatorów logicznych ‘not’, ‘and’, ‘or’ na kolejno ‘!’, ‘&&’, ‘||’
- Zamiana ‘print(expression1, expression2)’ na ‘cout << expression1 << expression2 << endl’

# Testowanie

Testowanie będzie odbywać się przy użyciu krótkich przykładów testujących, porównując uzyskany przez program kod z oczekiwanym.