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 na wejściu dostaje plik z kodem napisanym w języku Python, jako wyjście program generuje plik z analogicznym kodem napisanym w C++ albo wypisuje komunikat o odpowiednim błędzie. Program będzie napisany w języku Python.

Założenia

- Obsługiwanymi typami są int, float, bool, str
- Wszystkie zmienne są statycznie typowane (zmienna nie może być najpierw typu int a potem str)
- 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

Przykład

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

x = 2
z = "a"

while x > 0:
        x = x - 1
        print(z)
y = 3
print(sum(x,y))
```

```
#include <iostream>
#include <string>

using namespace std;

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

int main()
{
        int x = 2;
        string z = "a";
        while(x > 0)
        {
                x = x - 1;
                cout << z << endl;
        }
        int y = 3;
        cout << sum(x, y) << endl;
        return 0;
}
```

Gramatyka

program = statements

statements = statement | statement statements

statement_block = indent statements dedent

statement = assignment_statement | function_statement | return_statement | while_statement | if_statement | print_statement | expression_statement | EOL

assignment_statement = identifier '=' expression_statement

function_statement = 'def' identifier '(' [identifier ':' type] { ',' identifier ':' type } ')' '->' (type | 'None') ':' EOL statement_block

return_statement = 'return' expression_statement

while_statement = 'while' expression_statement ':' EOL statement_block

if_statement = 'if' expression_statement ':' EOL statement_block [el_statement]

el_statement = else_statement | elif_statement

elif_statement = 'elif' expression_statement ':' EOL statement_block [el_statement]

else_statement = 'else' ':' EOL statement_block

print_statement = 'print' '(' expression_statement ')'

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 '(' [identifier | value] { ',' (identifier | value) } ')'

unary _op = '+' | '-'

unary_logic_op = 'not'

binary_op = '+' | '-' | '*' | '/' | '%'

binary_logic_op = 'and' | 'or'

comparison_op = '==' | '!=' | '<' | '<=' | '>=' | '>'

type = 'int' |'float' | 'bool' | 'str'

value = int |float |bool | str

identifier = [a-zA-Z_][a-zA-Z0-9_]*

intend – wzrost liczba początkowych tabulacji o 1 względem poprzedniej linii

dedend – spadek liczby początkowych tabulacji o 1 względem poprzedniej linii