

使用ANTLR4实现的C++转python翻译器

本项目使用 ANTLR4 实现了 C++ 转 python 翻译器。基于上次词法分析器和语法分析器的背景，通过将生成的 AST 树进行遍历后进行转换，生成与 C++ 语法相配的 Python 语法。翻译器通过词汇分析、解析和代码生成，将 C++ 程序转换为语义上等价的 Python 代码，同时维护 Python 的习惯用法和最佳实践。该系统成功地处理了基本的 C++ 构造，包括控制结构、函数、类和基本数据类型。

一、团队分工

@苏伟铭：搭建框架 @汪佳宇：基础翻译逻辑 @陈立心：完善翻译逻辑，进行相关测试

@王旭冉：完善翻译逻辑，进行相关测试 文档由大家统一完成。

二、实验内容

1、开发环境： Visual Studio Code , Windows 11

2、实验工具： ANTLR4

3、文件结构：

- src/tests -所有的测试用例，及生成的 Python 代码
- src/translation -从 AST 树翻译成 Python 的逻辑实现
- src/CPPLexer.g4 - 定义词法分析规则的 ANTLR4 语法文件
- src/CPPParser.g4 - 定义语法解析规则的 ANTLR4 语法文件
- src/main.py - 使用生成的翻译器的 Python 脚本
- src/cppParserBase.py - 解析器基类实现
- src/CppParserVisitor.py - 访问者模式实现，用于生成 AST
- src/CPPToPythonVisitor.py -访问者模式实现，用于遍历 AST 树
- src/cpp_to_python_transpiler.py -访问者模式实现，用于生成 Python 代码

4、难点与创新点

- 符号表的实现：通过使用字典存储符号信息，并采用作用域复制机制支持嵌套作用域，同时在不同转换器间共享符号表，实现了变量的声明、查找和类型检查，以及基本的符号检查和错误处理功能。

- 符号表的基本结构:

在代码中, 符号表主要通过字典(dictionary)实现, 具体体现在以下几个关键变量:

```
current_vars: Dict[str, str] # 变量名到类型的映射
custom_classes: List[str] # 自定义类名列表
current_functions: List[str] # 当前作用域内的函数名列表
```

- 作用域管理:

- 局部作用域:

```
def convert_compoundStatement(self, node: Node, current_vars: dict[str, str],
                                custom_classes: list[str], current_functions:
                                list[str]) -> list[str]:
    # 创建新的作用域
    scope_vars = current_vars.copy()
    # 处理作用域内的声明和语句
    ...
    # 更新父作用域中已存在的变量
    for var, type_ in scope_vars.items():
        if var in current_vars:
            current_vars[var] = type_
```

- 控制流作用域: 对于控制流语句 (如 `if-else`), 每个分支会创建独立的作用域。

```
def convert_selectionStatement(self, node: Node, current_vars: dict[str, str],
                                custom_classes: list[str], current_functions:
                                list[str]) -> list[str]:
    # if块的作用域
    if_vars = current_vars.copy()
    # else块的作用域
    else_vars = current_vars.copy()
```

- 符号表维护机制:

- 变量声明时的符号表更新:

```
current_vars[var_name] = var_type
loop_vars[var_name] = var_type
```

- 作用域链处理:

```
scope_vars = current_vars.copy() # 复制当前作用域的符号表
```

```
# 将修改过的变量更新回父作用域
for var, type_ in scope_vars.items():
    if var in current_vars:
        current_vars[var] = type_
```

○ 符号表的使用:

- 变量查找: 在ExpressionConverter中进行变量引用检查

```
def convert_variable(self, root_var: str, current_vars: dict[str, str]):
    if root_var not in current_vars:
        raise SyntaxError(f"variable {root_var} referenced before
declaration!")
```

- 类型检查: 通过符号表进行类型检查和转换

```
var_type = current_vars.get(var, 'str')
if var_type in ['int', 'float']:
    result.append(f"{var} = {var_type}(input())")
```

○ 错误处理:

实现了基本的符号检查和错误处理，能在语法错误时快速定位问题

通过自定义错误监听器，能够捕获并抛出详细的错误信息，从而提升调试效率。

```
class TranspilerErrorListener(ErrorListener):
    def syntaxError(self, recognizer, offendingSymbol, line, column, msg, e):
        raise TranspilerError(f"Syntax error at line {line}, column {column}:
{msg}")
```

- C++ 中的 ++ 与 -- 可以在 Python 中正确翻译

○ 实现方法:

1. **解析右侧表达式**: 首先，解析 ++ 或 -- 操作符后面的表达式。
2. **生成基本表达式**: 创建一个基本的 Python 表达式，如 `right := right`。
3. **处理自增 ++**: 如果是 ++，将表达式改为 `right := right + 1`。
4. **处理自减 --**: 如果是 --，将表达式改为 `right := right - 1`。

```

■ if expression_node.node_type == "unaryExpression":
    if expression_node.children[0].node_type == 'referenceOp':
        raise SyntaxError("pointer-related operations not supported
in expressions!")
    right =
self.convert_expression_online(expression_node.children[1],
current_vars, custom_classes, current_functions)
    if expression_node.children[0].node_type == 'NOT':
        return ' not ' + right
    py_statement = '(' + right + ' :=' + right
# ++/-- unaryExpression
    if expression_node.children[0].node_type == 'INCREMENT':
        py_statement += '+1)'
        return py_statement
    elif expression_node.children[0].node_type == 'DECREMENT':
        py_statement += '-1)'
        return py_statement

```

- 错误处理：在词法错误，语法错误，逻辑错误时均进行报错，并给出具体原因
 - 实现方法：在词法发生错误时，`Lexer` 进行报错；在有具体的语法错误时，由 `Parser` 进行报错；在实现有语义错误时，如预先使用未定义变量，会由翻译器进行报错。

- 例 1： `int b` 不在结尾加分号会报错

```

■ ERROR: Transpilation failed: Syntax error at line 1, column 5: no viable alternative at input 'intb'
Transpilation error: Syntax error at line 1, column 5: no viable alternative at input 'intb'

```

- 例 2：使用未定义变量时报错

```

ERROR: Transpilation failed: Convert_variable: variable a referenced before declaration! Current vars: {}
Transpilation error: Convert_variable: variable a referenced before declaration! Current vars: {}

```

```

int main() {
    std::cout << a;
}

```

- 例 3： `for` 循环时出现语法错误

```

ERROR: Transpilation failed: Syntax error at line 3, column 23: missing ';' at ')'
Transpilation error: Syntax error at line 3, column 23: missing ';' at ')'

```

```
#include <iostream>
int main() {
    for(int i = 0; i<10) {
        std::cout<< i ;
    }
}
```

- 实现了四则运算计算支持
 - 实现方法：通过支持栈结构，实现四则运算计算。

```
(base) PS E:\CODE\SEMESTER 5\complie\Compile\src> python tests/test_calc.py
Result: -35
```

- 实现了最主流的排序算法支持

```
(base) PS E:\CODE\SEMESTER 5\complie\Compile\src\tests> python test_bubbleSort.py
Input the array length: 6
Input integers: 2
5
7
4
8
3
Result: 2 3 4 5 7 8
```

```
• (base) PS E:\CODE\SEMESTER 5\complie\Compile\src\tests> python test_insertionSort.py
Original array: 12 11 13 5 6 7
Sorted array: 5 6 7 11 12 13
```

```
• (base) PS E:\CODE\SEMESTER 5\complie\Compile\src\tests> python test_mergeSort.py
Original array: 38 27 43 3 9 82 10
Sorted array: 3 9 10 27 38 43 82
```

```
• (base) PS E:\CODE\SEMESTER 5\complie\Compile\src\tests> python test_quickSort.py
Original array: 38 27 43 3 9 82 10 19
Sorted array: 3 9 10 19 27 38 43 82
```

5.特性支持

- 基本 C++ 语言语法
- 函数和类定义
- 基本数据类型和操作

三、测试用例

- **KMP字符串匹配算法** (`test_KMP.cpp`): 实现了经典的KMP字符串匹配算法, 包含部分匹配表 (next数组)的计算、字符串匹配过程、多重匹配支持和错误处理。
- **多种排序算法** (`test_bubbleSort.cpp` 等): 实现了主流的排序算法, 支持动态数组输入、整数排序和结果可视化输出。
- **回文检测算法** (`test_palindrome.cpp`): 实现了回文检测的算法, 支持动态字符串输入、忽略大小写和非字母数字字符, 并且在遇到不匹配时提前终止检测, 结果可视化输出。
- **四则运算算法** (`test_calc.cpp`): 实现了自定义栈结构的四则运算算法, 支持负数处理、中缀表达式处理和运算符优先级, 支持动态字符串输入。

注: 所有已生成的 `Python` 代码均可直接运行。

四、运行方法

详情请见 `readme.txt`

五、参考资料

- [ANTLR4文档](#)[Python文档](#)[C++文档](#)

附:关于AST和语法解析器等问题详见第一次提交报告