编译小组作业——代码高亮与提示程序

陈俊哲　2020010964　软件 01
梁　烨　2020080093　软件 01
田正祺　2020080095　软件 01

1　概述

此项目实现了 C++ 的词法分析器、语法分析器、以及代码高亮与提示程序。此项目的大部分逻辑实现于 Python 3，其中关于编译的算法与逻辑使用了 PLY。实现过程中主要参考的资料是 C++20 标准 N4860 版本。

2　开发环境

本次小组作业的成员使用的操作系统是 Windows 10，所使用的编译器为 Visual Studio Code。在实验的过程中, 我们使用了 Python Lex-Yacc 作为辅助工具进行代码高亮与提示程序的开发:

3　词法分析

3.1　实现

标准中的第五章中定义了 C++ 的词法小组使用了 Programming Languages —C++ Working Draft N4860 作为参考书和辅助工具书进行了词法分析部分的开发。其中，对于 C++20 里四个重要的 literals 类型（整数 Integer，浮点数 Float，字符 Characters（C++20 支持 MultiCharacters），字符串 String），小组进行了 tokens 的编写。根据不同的 tokens，用正则表达式和函数定义的方式编写了 tokens 的规则。

开发过程中发现每种记号的规则顺序与这些规则定义的顺序可能不一样，因此导致 `char c = u8't'` 中的 `u8` 被识别为标识符，而正确的是整个 `u8't'` 是一个字符常量。因此改变了 `myply/lex.py` 中的 `lex` 函数，让用户明确地定义记号类型的识别顺序。因此此项目 必须使用 `myply` 下的 PLY 包。

3.2　难点

阅读标准之前，我们以为 C++ 标识符可以简简单单地使用 `[_a-zA-Z][_a-zA-Z0-9]*` 的正则表达式匹配。但是 C++20 将可以作为标识符的符号大幅增加，甚至中文字与表情是合法的标识符字符。为了强调标准所允许的范围，以下显示的是符合标准的正则表达式：

```
((?![\u0300-\u036F\u1DC0-\u1DFF\u20D0-\u20FF\uFE20-\uFE2F])([_a-zA-Z]|[\u00A8\u00AA\u00AD\u00AF\u00B2-\u00B5\u00B7-\u00BA\u00BC-\u00BE\u00C0
    -\u00D6\u00D8-\u00F6\u00F8-\u00FF\u0100-\u167F\u1681-\u180D\u180F-\u1FFF\u200B-\u200D\u202A-\u202E\u203F-\u2040\u2054\u2060-\u206F\
    u2070-\u218F\u2460-\u24FF\u2776-\u2793\u2C00-\u2DFF\u2E80-\u2FFF\u3004-\u3007\u3021-\u302F\u3031-\uD7FF\uF900-\uFD3D\uFD40-\uFDCF\uFDF0
    -\uFE44\uFE47-\uFFFD\U00010000-\U0001FFFD\U00020000-\U0002FFFD\U00030000-\U0003FFFD\U00040000-\U0004FFFD\U00050000-\U0005FFFD\U00060000
    -\U0006FFFD\U00070000-\U0007FFFD\U00080000-\U0008FFFD\U00090000-\U0009FFFD\U000A0000-\U000AFFFD\U000B0000-\U000BFFFD\U000C0000-\
    U000CFFFD\U000D0000-\U000DFFFD\U000E0000-\U000EFFFD])(([_a-zA-Z]|[\u00A8\u00AA\u00AD\u00AF\u00B2-\u00B5\u00B7-\u00BA\u00BC-\u00BE\u00C0
    -\u00D6\u00D8-\u00F6\u00F8-\u00FF\u0100-\u167F\u1681-\u180D\u180F-\u1FFF\u200B-\u200D\u202A-\u202E\u203F-\u2040\u2054\u2060-\u206F\
    u2070-\u218F\u2460-\u24FF\u2776-\u2793\u2C00-\u2DFF\u2E80-\u2FFF\u3004-\u3007\u3021-\u302F\u3031-\uD7FF\uF900-\uFD3D\uFD40-\uFDCF\uFDF0
    -\uFE44\uFE47-\uFFFD\U00010000-\U0001FFFD\U00020000-\U0002FFFD\U00030000-\U0003FFFD\U00040000-\U0004FFFD\U00050000-\U0005FFFD\U00060000
    -\U0006FFFD\U00070000-\U0007FFFD\U00080000-\U0008FFFD\U00090000-\U0009FFFD\U000A0000-\U000AFFFD\U000B0000-\U000BFFFD\U000C0000-\
    U000CFFFD\U000D0000-\U000DFFFD\U000E0000-\U000EFFFD])|[0-9])*)
```

除了标识符以外，标准定义了

难点一：Literals 中的各种符号问题以及各种括号问题正则表达式虽然是很直观的东西且编写时的思路是较为清晰的，我们在编写 literals 的时候，速度还算是较快的。但是由于正则表达式对于细节的把控，尤其是在针对括号的地方，导致经常会出现括号对不上号，或者是有的 [] 需要在外面加一层括号才能正常使用的情况，对小组成员造成了一定的影响。同时，因为转义字符的原因，对于一些正则表达式中不常见的但因为编译语言或者是编译器的原因需要转义字符才能正常表达的符号，需要在不断的调试过程中才能找出这些符号并加以处理。难度二：C++20 的标准非常长 C++20 是 C++ 非常新的版本，其中它对很多标准都做了更新，比如 char 一般只是一个字符，但在 C++20 中却可以有多个字符（类似于一个 string，但又不完全一样），导致在编写的想要完全符合 C++20 的标准非常困难。

3.3 创新点

小组实现了一个可视化的网页界面，当把鼠标光标放到词上的时候，会自动显示它的 tokens。整体的页面非常的间接且清晰明了。

4 文法分析

4.1 难点

C++20 标准非常复杂，工作量很大。和词法分析部分类似，因为 C++20 是 C++ 非常新的版本，它对很多标准都做了更新和扩充，使得它的各种标准非常多，比如 **`char`** 一般只是一个字符，但在 C++20 中却可以有多个字符（类似于一个 string，但又不完全一样），导致在编写的时候面对的工作量非常大。不管是词法分析还是语法分析的大作业部分，本小组一开始都是严格按照 C++ 标准去进行实现的。在一步一步按照 C++20 标准实现语法分析部分的时候，最长的 declarations 文法足足写了有 800 多行代码。而在我们实现了那十个部分的文法后，发现工作量大带来的问题便是在调试的过程中会遇到不计其数的问题。首先是修改起来最简单的 typo 问题，有显性的，比如打错了字，也有隐性的，比如某一层文法或者产生式中多写了一个分号。在之后，便是各种文法冲突的问题。在试图解决文法冲突，并苦苦挣扎了将近一整个周末后，我们不得已选择抛弃掉部分 C++20 标准，选择按照大作业文档中的要求，保留基础文法，最终实现了语法高亮程序需要用的语法分析部分。

5 代码高亮与提示

5.1 前端

代码高亮与提示的可视化是通过网页在 `editor/` 下的 HTML, JS 与 CSS 文件实现的。其中使用了 Bootstrap 做格式化，与定义了一个 HTML 元素、提供了文本输入的基本框架与格式的 `code-input` 。此页面中包括文本编译器的几个基本功能，其描述以及实现方法如下：

1. 代码高亮：`code-input` 提供了一个 JS 接口，让开发者定义一个函数，其参数为包含用户代码输入的文本框。用户每次改变文本框的内容会调用这个函数，并由这个函数根据用户的输入而提供包含高亮的 HTML 代码。此函数在 `editor/editor.js` 中的 `handle_update` 函数实现。它将用

户的输入发给后端，并接收后端的响应而更新网页上的高亮状态。后端返回的 HTML 中包括每个记号本身与其类名。`editor/editor.css` 根据类名进行高亮。高亮的颜色与 VSCode 的 Microsoft C/C++ 插件的颜色一致，即关键字为深蓝色、标识符为浅蓝色、数值型常量为浅绿色、字符（串）常量为橘色、运算符为白色、类名为绿色、引入语句为紫色、函数名及其调用为黄色、注释为深绿色。

2. 代码提示：前端发给后端的信息中包括用户的光标目前所在的位置。后端根据光标的位置与已有的代码提供若干个提示，由 `handle_update` 显示这些提示。当用户通过点击或者使用快捷键选择某个提示后，`editor/editor.js` 中的 `autocomplete` 函数将进行补全。此外当用户输入 `' " ( [ {` 中的一个字符后，它对应的字符将自动补全。

3. 快捷键：用户可以使用制表键插入四个空格符号。使用 `ctrl + n` 使用序号为 `n` 的提示进行代码补全。

5.2 后端

由于输入到语法分析器中的记号中不能包含注释，但是代码高亮中需要展示注释，所以传给语法分析器的词法分析器是继承 `lex.Lexer` 的 `NoCommentsLexer`。除了在调用 `token` 函数时它会跳过所有注释，`NoCommentsLexer` 与 `lex.Lexer` 的功能完全一样。

6 小组分工

陈俊哲：
梁烨：
田正祺：代码高亮与提示前端、

7 辅助说明