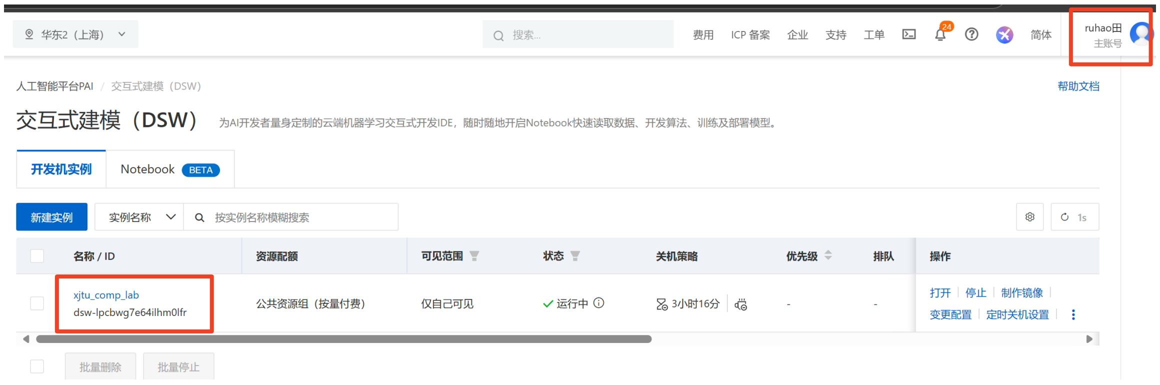
编译器设计专题实验二：词法分析器实现

计算机 2101 田濡豪 2203113234

1 目录

2 环境配置

本人选择使用 Visual Studio Code 配合阿里云 PAI-DSW 完成实验，在阿里云中创建的 DSW 实例如下：



交互式建模（DSW） 为AI开发者量身定制的云端机器学习交互式开发IDE，随时随地开启Notebook快速读取数据、开发算法、训练及部署模型。

名称 / ID	资源配额	可见范围	状态	关机策略	优先级	排队	操作
xjtu_comp_lab dsw-lpcbwg7e64ilhm0lfr	公共资源组（按量付费）	仅自己可见	✓ 运行中	3小时16分	-	-	打开 | 停止 | 制作镜像 | 变更配置 | 定时关机设置

ID 作为本人凭据。

由于校园网带宽有限，随着代码库增大远程开发的延迟和响应性显著降低，本次实验主要在本地完成，**但是在运行测试程序时会一同展示本人的阿里云控制台界面作为凭据。**

3 实验内容

3.1 实验要求（需求分析）

要求：

基于两种不同后端的词法分析单元

功能：

给定类似 C 语言的文法配置和一个输入串，识别串中的各个词法单元，并输出每个词法单元的类型和内容。

后端要求 1：使用 DFA 定义文法

- 输入一个词法记号类别，包括名称、优先级
- 输入一系列 DFA 定义，其中每个状态机对应一个词法记号类别
- 能够识别一个输入串中的所有词法记号，并输出每个词法记号的类型和内容

后端要求 2：使用 FLEX 工具和正则表达式定义文法

- 输入一个词法记号类别，包括名称、优先级
- 输入一系列正则表达式定义，其中每个正则表达式对应一个词法记号类别

- 调用 FLEX 工具对输入串进行此法分析

前端要求：

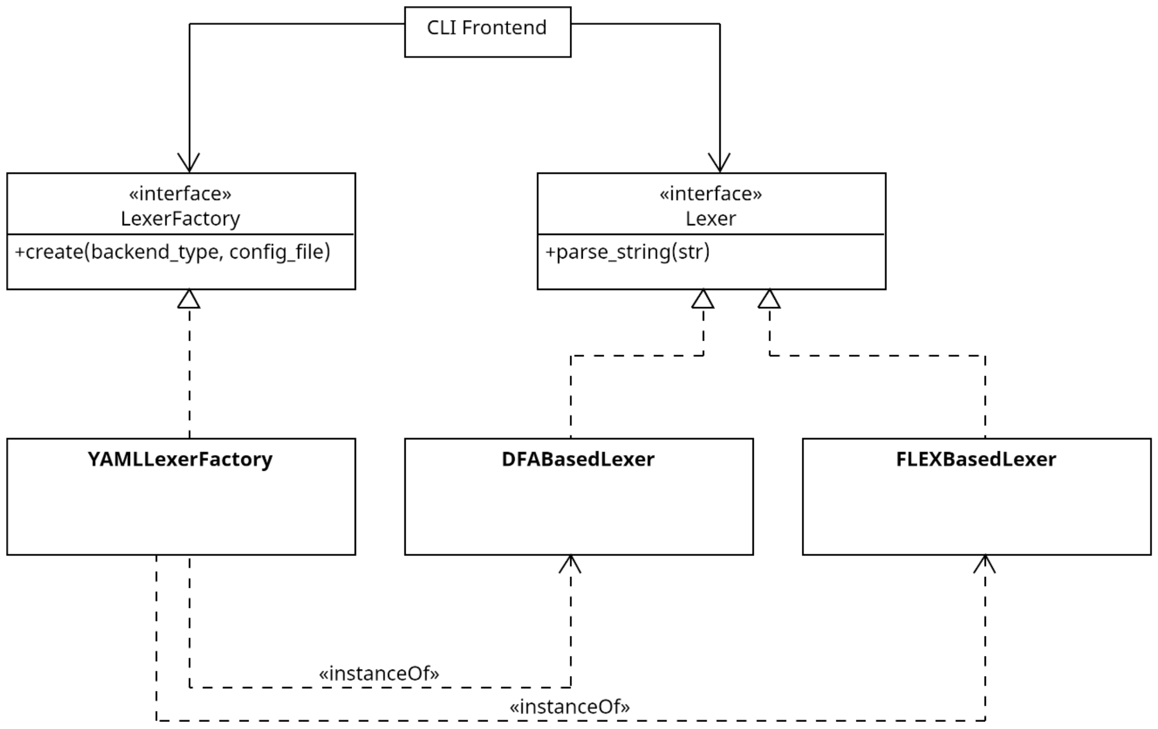
- 允许通过 CLI 选择不同的后端
- 输入配置文件和要匹配的字符串
- 将词法分析结果格式化输出到文件中

3.2 实验设计

3.2.1 总体架构

显然，两个后端对前端实现的具体功能是相同的，因此前后端之间应该通过一个统一接口进行依赖倒置。但是，注意到这两个后端的数据依赖并不相同。DFA 在初始化时需要一个状态转换表，而 FLEX 需要一个正则表达式定义的文法。因此，前端需要根据后端的不同选择不同的配置文件格式，破坏了封装性。对于这种易变的依赖关系，一个典型的方案是使用工厂模式来解决。即前端除了词法分析器接口还拥有一个工厂接口，工厂会根据配置文件和后端选择创建加载好 DFA 或正则表达式定义的词法分析器实例。这样，前端只需要依赖于词法分析器接口，而不需要依赖于具体的实现类。

整体架构如下图所示：



3.2.2 词法分析器接口

定义一个词法分析器实例的功能就是在其生存周期内匹配同一种文法的字符串，所以其必须包括的接口方法很简单：

- 输入一个字符串，返回匹配有关的结果信息。

综合考虑错误处理等因素，定义一个回传的数据模型：

单个词法单元匹配结果

词法类别：使用 std::string 表示

词法内容：使用 std::string 表示

匹配结果列表

一个 std::vector，包含多个此法单元匹配结果，每个结果的位置表示其匹配的顺序

返回结果

一个数据类，包含一个匹配结果列表，匹配是否成功的标志，错误信息

3.2.3　基于 DFA 的后端设计

基于 DFA 的词法分析器运作的大致过程如下：给定一个前缀，使用不同词法类别的 DFA 进行匹配，对匹配成功的类别选取最高优先级作为其最终字符串类别。对一个字符串，每次寻找前缀的时候使用最长匹配原则，直到没有可匹配的前缀为止。

可见其中主要有两部分：DFA 判断程序和前缀匹配程序。其功能是分离的。DFA 判断程序只负责判断一个字符串符不符合某个 DFA 的定义，而前缀匹配程序负责在一个字符串中寻找所有符合某个 DFA 的前缀。前缀匹配程序需要使用 DFA 判断程序来判断每个前缀是否符合 DFA 定义。

3.2.3.1　DFA 判断程序

首先考虑 DFA 判断程序，上个实验中已经给出了一个 DFA 数据模型。

```cpp
struct DFA {
    std::unordered_set<char> character_set;                          // 字符集
    std::unordered_set<std::string> states_set;                      // 状态集
    std::string initial_state;                                       // 开始状态
    std::unordered_set<std::string> accepting_states;                // 接受状态集
    std::unordered_map<std::string, std::unordered_map<char, std::string>> transitions; // 状态转换表
};
```

并且同时实现了能够判断字符串是否属于单个 DFA 的数据类。

```cpp
class DFASimulator {
public:
    virtual ~DFASimulator() = default;
    // 更新 DFA 设定
    virtual bool UpdateDFA(const DFA& dfa) = 0;
    // 模拟单一字符串
    virtual bool SimulateString(const std::string& input, std::string& simulation_log) const = 0;
    // 生成所有符合规则的字符串
    virtual std::set<std::string> GenerateAcceptedStrings(int max_length) const = 0;
};
```

因此，本次实验的 DFA 程序只要对其进行简单封装。

- 维护一个 MAPPING 表，key 为词法类别，value 为 DFA 实例
- 提供一个词法类别判断方法，输入字符串和要匹配的词法类别，返回是否匹配成功
- 提供错误处理，比如 key 不存在，DFA 匹配出现错误等

3.2.3.2　前缀匹配程序

前缀匹配程序的主要功能是对一个字符串以最长匹配原则和最高优先级原则进行匹配。其中最长匹配原则是占主导的，而最高优先级原则只在多个词法类别都匹配成功时才会使用。整个算法的自然语言描述如下：

- 维护一个空的中间状态列表。用于记录字符串中所有可能匹配成功的前缀。
- 把当前字符串初始化为整个字符串。
- 从头开始迭代当前字符串的前缀。
- 对当前前缀，把所有词法类别输入 DFA 匹配程序。如果有一个或多个匹配成功，取其中优先级最高的作为当前前缀的此法类别，并将（词法类别，词法内容，前缀长度）加入中间状态列表。如果没有，那么无需加入中间状态列表。
- 把下一个字符加入当前前缀，使用上一个步骤继续迭代，直到字符串结束。
- 回顾中间状态列表，选取其中最长的前缀作为最终的匹配结果，并将其加入最终结果列表。
- 将当前字符串更新为去掉前缀后的字符串，重复上面的步骤，直到没有可匹配的前缀为止。
- 返回最终结果列表。

因此可以从中抽象出几个基本的数据模型，每个包装在一个类中。

词法类别

名称：使用 std::string 表示

优先级：使用 int 表示，越小优先级越高

是否“多符一种”: 使用 bool 表示，即词法内容是否有意义，如 ID 的内容是变量名，IF 的内容没有意义

比较方法：按照优先级大小比较（重载运算符）

词法类别列表

一个 std::set，包含多个词法类别

中间状态

词法类别：使用 std::string 表示

词法内容：使用 std::string 表示

前缀长度：使用 int 表示

前缀优先级：使用 int 表示，越小优先级越高

比较方法：按照前缀长度大小比较（重载运算符）

中间状态列表

一个 std::set，包含多个中间状态

匹配结果

词法类别：使用 std::string 表示

词法内容：使用 std::string 表示

最终结果

一个 std::vector，包含多个匹配结果

此处最终结果和要求返回的数据类中的匹配结果列表是相同的，因此可以直接使用最终结果作为返回结果。有很多可以优化的部分，但考虑到可读性和程序本身轻量级，暂时不做优化。

3.2.3.3 初始化

根据以上需求，定义一个专门用于初始化的数据类：

```
struct ConstructionInfo
{
    std::vector<TokenType> token_types; // List of token types
    std::vector<DFA> dfa_configurations; // List of DFA configurations
};
```

3.2.4 基于 FLEX 的后端设计

最初的设想是使用 FLEX 的 C++接口来实现 FLEX 后端，但是其无法直接在运行时加载正则表达式定义的文法。经过资料查找发现，参考 FLEX 设计的现代化正则表达式处理库 RE/FLEX 具备此功能，因此决定使用 RE/FLEX 来实现 FLEX 后端。其实先与 DFA 后端几乎一致，只是将 DFA 判断程序替换为 RE/FLEX 的正则表达式判断程序。

3.2.5 后端工厂类设计

该部分主要解决的问题是：

- 根据后端类型创建不同的词法分析器实例
- 根据后端类型加载不同格式的配置文件

3.2.5.1 DFA 后端数据模型设计

根据 DFA 后端的设计，配置文件中需要包含以下信息：

- 词法类别列表，其中包括每个类别的名称和优先级
- DFA 列表，其中包括每个 DFA 的状态集、字符集、初始状态、接受状态集和状态转换表
- 词法类别和 DFA 的映射关系

同时注意到词法类别列表对 FLEX 后端也是有用的，因此可以将其分离在一个单独的配置文件中。使用 YAML 格式进行配置，方便后续扩展，一个典型的配置示例如下：

```yaml
token_types:
  - name: "IDENTIFIER"
    priority: 1
    has_content: true
  - name: "NUMBER"
    priority: 2
    has_content: true
  - name: "OPERATOR"
    priority: 3
    has_content: false # 假设操作符没有内容
```

对于 DFA 定义，在实验一中已经设计过了一个 DFA 数据模型，但是其字符集表示占用了大量空间。考虑到经典 DFA 的字符集由单个字符组成，因此可以将其简化为一个字符串表示。一个典型的 DFA 配置文件如下：

```yaml
dfas:
  - name: "IDENTIFIER"
    character_set: "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_"
    states_set: ["q0", "q1"]
    initial_state: "q0"
    accepting_states: ["q1"]
    transitions:
      - from: "q0"
        to: "q1"
        character: "abcde"
      - from: "q1"
        to: "q1"
        character: "Z"
```

3.3 实现细节

3.3.1 前缀匹配程序

从实验设计的自然语言描述中可以抽象出三个单一职责的部分：

1. 单轮匹配程序：对一个字符串按最长匹配原则进行单轮匹配
2. 词法类别匹配程序：检查某个字符串是否符合某个词法类别
3. 字符串总匹配程序：不断匹配一个字符串，每次去掉单论匹配的字串，直到待匹配的字符串为空

其核心代码实现如下：

单轮匹配程序：（以 DFA 后端为例）

```
DFABasedLexerHelper::SingleIterationResult DFABasedLexer::SingleIterationM
atchLongestPrefix(const std::string &input) const
{
    spdlog::debug("Performing single iteration match for input: {}", input
);
    DFABasedLexerHelper::IntermediateState longest_prefix_state;
    bool has_matched = false;
    // iterate through all substrings of the input
    for (size_t length = 1; length <= input.length(); ++length)
    {
        // get the current substring
        std::string current_substring = input.substr(0, length);
        // iterate through all token types
        for (const auto &token_type : token_types)
        {
            // check if the current substring matches the token type
            spdlog::debug("Checking substring: {}, against token type: {}"
, current_substring, token_type.name);
            if (MatchTokenType(current_substring, token_type))
            {
                bool update_longest_prefix = false;
                // if the longest prefix state is empty, update it
                if (!has_matched)
                {
                    update_longest_prefix = true;
                    has_matched = true;
                }
                // elif the current substring is longer than the longest p
refix state, update it
                else if (length > longest_prefix_state.prefix_length)
                {
                    update_longest_prefix = true;
                }
                // if the current substring is equal to the longest prefix
 state, perform priority check
                else if (length == longest_prefix_state.prefix_length)
                {
                    if (token_type.priority_level < longest_prefix_state.t
oken_priority_level)
                    {
                        update_longest_prefix = true;
                    }
                    else if (token_type.priority_level == longest_prefix_s
tate.token_priority_level)
                    {
                        // undefined, raise error
                        throw std::runtime_error("Ambiguous token match: "
 + longest_prefix_state.token_type_name + " and " + token_type.name + " wi
th same priority level: " + std::to_string(token_type.priority_level));
                    }
                }
                // update the longest prefix state
                if (update_longest_prefix)
                {
                    spdlog::debug("Updating longest prefix state: type = {
}, value = {}, length = {}, priority = {}", token_type.name, current_subst
ring, length, token_type.priority_level);
                    longest_prefix_state.token_type_name = token_type.name
;
                    if (token_type.has_content)
                    {
                        longest_prefix_state.token_value = current_substri
ng;
                    }
                    else
                    {
                        spdlog::debug("Token type {} has no content, setti
ng value to '-'", token_type.name);
                        longest_prefix_state.token_value = "-
"; // placeholder for non-content token types
                    }
                    longest_prefix_state.prefix_length = length;
                    longest_prefix_state.token_priority_level = token_type
.priority_level;
                }
            }
        }
    }
    // construct the result
    DFABasedLexerHelper::SingleIterationResult result;
    result.has_matched = has_matched;
    result.longest_prefix_state = longest_prefix_state;
    return result;
};
```

单字符串匹配总程序：（以 DFA 后端为例）

```
std::vector<Token> DFABasedLexer::ParseStringToTokens(const std::string &i
nput) const
{
    std::vector<Token> tokens;
    std::string unconsumed_input = input;
    // Iterate until the input is fully consumed
    while (!unconsumed_input.empty())
    {
        // try to match the longest prefix with current prefix length
        DFABasedLexerHelper::SingleIterationResult result = SingleIteratio
nMatchLongestPrefix(unconsumed_input);
        // check if a match was found
        if (!result.has_matched)
        {
            throw std::runtime_error("No matching token type found for inp
ut: " + unconsumed_input);
        }
        else
        {
            // if a match was found, create a token and add it to the list
            Token token;
            token.type = result.longest_prefix_state.token_type_name;
            token.value = result.longest_prefix_state.token_value;
            tokens.push_back(token);
            spdlog::debug("Token found: type = {}, value = {}", token.type
, token.value);
            // update the unconsumed input
            unconsumed_input = unconsumed_input.substr(result.longest_pref
ix_state.prefix_length);
        }
    }
    return tokens;
}
```

词法类别匹配程序-DFA 后端：

```
bool DFABasedLexer::MatchTokenType(const std::string &input, const TokenTy
pe &token_type) const
{
    // TODO: log handling
    std::string simulation_log;
    auto it = dfa_simulators.find(token_type.name);
    if (it != dfa_simulators.end())
    {
        return it->second->SimulateString(input, simulation_log);
    }
    return false;
}
```

词法类别匹配程序-FLEX 后端：

```
// check if a give string match a specific token type, this is the only di
fference from dfa_based_lexer
bool FlexBasedLexer::MatchTokenType(const std::string &input, const TokenT
ype &token_type)
{
    auto it = regex_patterns.find(token_type.name);
    if (it != regex_patterns.end())
    {
        // build the matcher
        reflex::Matcher matcher(*it->second, input);
        // check if the input matches the regex pattern
        if (matcher.matches())
        {
            spdlog::debug("Matched token type: {} with input: {}", token_t
ype.name, input);
            return true;
        }
        else
        {
            spdlog::debug("No match for token type: {} with input: {}", to
ken_type.name, input);
            return false;
        }
    }
    spdlog::error("Token type {} not found in regex pattern matchers", tok
en_type.name);
    return false;
}
```

3.3.2 后端构造程序

构造后端的主要流程是：

1. 加载配置文件
2. 检查配置文件
3. 根据配置文件生成后端

这里给出核心函数：

```
std::unique_ptr<LexerInterface> YAMLLexerFactory::CreateLexer(const std::s
tring& lexer_type, const std::string& general_config, const std::string& s
pecific_config) {
    // Load general configurations
    unchecked_token_types = LoadGeneralConfigs(general_config);
    // check if the general configurations are valid
    try {
        CheckTokenTypes(unchecked_token_types);
    }
    catch (const std::exception& e) {
        throw std::runtime_error("Token types check failed: " + std::strin
g(e.what()));
    }
    if (lexer_type == "DFA"){
        // Load DFA configurations
        unchecked_dfa_mapping = LoadDFAConfigs(specific_config);
        // check if the DFA configurations are valid
        try{
        CheckDFAConfigurations(unchecked_dfa_mapping);
        }
        catch (const std::exception& e) {
            throw std::runtime_error("DFA configurations check failed: " +
 std::string(e.what()));
        }
        try {
        CheckTokenDFARelationships(unchecked_token_types, unchecked_dfa_ma
pping);
        }
        catch (const std::exception& e) {
            throw std::runtime_error("Token and DFA relationship check fai
led: " + std::string(e.what()));
        }
        // prepare construction info
        std::vector<TokenType> token_types;
        std::vector<DFA> dfa_configurations;
        // fill the token types and DFA configurations with unchecked_toke
n_types and unchecked_dfas_mapping, which are already checked
        for (const auto& token_type : unchecked_token_types) {
            token_types.push_back(token_type);
            // find the corresponding DFA configuration
            auto it = unchecked_dfa_mapping.find(token_type.name);
            if (it != unchecked_dfa_mapping.end()) {
                dfa_configurations.push_back(*(it->second));
            }
            else {
                throw std::runtime_error("DFA configuration not found for
token type: " + token_type.name);
            }
        }
        DFALexerSetup construction_info{token_types, dfa_configurations};
        // create a new DFA-based lexer
        return std::make_unique<DFABasedLexer>(construction_info);
    }
    else if (lexer_type == "FLEX") {
        // Load regex configurations
        unchecked_regex_mapping = LoadRegexConfigs(specific_config);
        // check if the regex configurations are valid
        try {
            CheckRegexTokenTypeRelationships(unchecked_token_types, unchec
ked_regex_mapping);
        }
        catch (const std::exception& e) {
            throw std::runtime_error("Token and regex relationship check f
ailed: " + std::string(e.what()));
        }
        // prepare construction info
        std::vector<TokenType> token_types;
        std::vector<std::string> regex_patterns;
        // fill the token types and regex patterns with unchecked_token_ty
pes and unchecked_regex_mapping, which are already checked
        for (const auto& token_type : unchecked_token_types) {
            token_types.push_back(token_type);
            // find the corresponding regex pattern
            auto it = unchecked_regex_mapping.find(token_type.name);
            if (it != unchecked_regex_mapping.end()) {
                regex_patterns.push_back(it->second);
            }
            else {
                throw std::runtime_error("Regex pattern not found for toke
n type: " + token_type.name);
            }
        }
        FlexLexerSetup construction_info{token_types, regex_patterns};
        // create a new flex-based lexer
        return std::make_unique<FlexBasedLexer>(construction_info);
    }
    else {
        throw std::invalid_argument("Unsupported lexer type");
    }
};
```

4 实验结果

4.1 数据准备

将 PPT 中的语法按照预定格式写成 YAML 表示，得益于 YAML 的词典特征，DFA 配置、词法配置和 FLEX 正则表达式配置可以写在同一个文件中。文件具体内容请见附录部分或源码，仅给出 FLEX 正则表达式部分。（由于标准正则表达式 PPT 中已经给出，DFA 也唯一确定了）

由于 YAML 语法中要求对反斜杠\进行转义，而 FLEX 表达式本身又要求对反斜杠进行转义，因此当 FLEX 正则表达式中出现一个*需要匹配的反斜杠*时，在 YAML 文件中需要用 *4 个连续的反斜杠进行表示。*

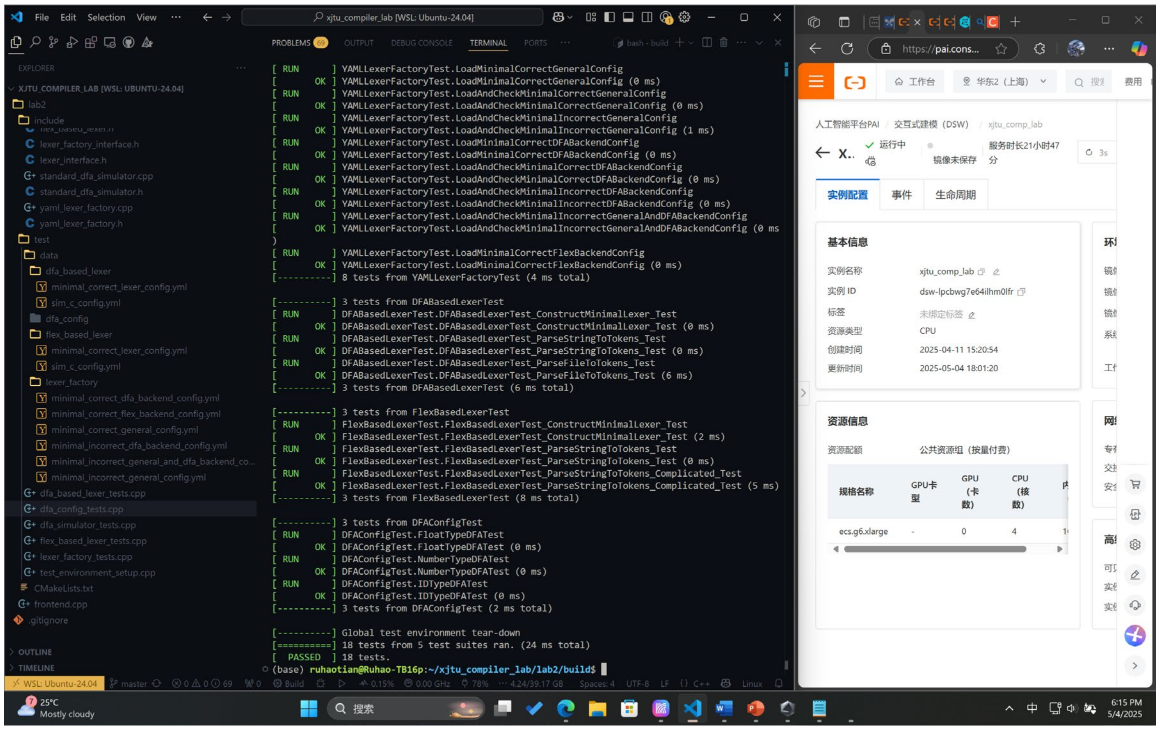
```
regexps:
- regexp: "^[a-z][a-z||0-9]*$" # flex regex syntax
  token_type: "ID"
- regexp: "^[+-]?[0-9]+$"
  token_type: "NUM"
- regexp: "^([+-]?[0-9]*\\.[0-9]+)|([+-]?[0-9]+\\.[0-9]*)$"
  token_type: "FLO"
- regexp: "^\\+$"
  token_type: "ADD"
- regexp: "^\\*$"
  token_type: "MUL"
- regexp: "^[(<)||[(<=)||(==)]]$"
  token_type: "ROP"
- regexp: "^\\\\=$"
  token_type: "ASG"
- regexp: "^\\\\\\($"
  token_type: "LPA"
- regexp: "^\\\\\\)$"
  token_type: "RPA"
- regexp: "^\\\\\\[$"
  token_type: "LBK"
- regexp: "^\\\\\\]$"
  token_type: "RBK"
- regexp: "^\\\\\\{$"
  token_type: "LBR"
- regexp: "^\\\\\\}$"
  token_type: "RBR"
- regexp: "^\\\\,$"
```

```
  token_type: "CMA"
- regexp: "^\\\\;$"
  token_type: "SCO"
- regexp: "^int$"
  token_type: "INT"
- regexp: "^float$"
  token_type: "FLOAT"
- regexp: "^if$"
  token_type: "IF"
- regexp: "^else$"
  token_type: "ELSE"
- regexp: "^while$"
  token_type: "WHILE"
- regexp: "^return$"
  token_type: "RETURN"
- regexp: "^input$"
  token_type: "INPUT"
- regexp: "^print$"
  token_type: "PRINT"
- regexp: "^void$"
  token_type: "VOID"
```

为了解决词法优先级冲突，还需要给每个此法类别一个独有的优先级。主要设计四个大类：关键字（while 等）为最高优先级大类，特殊符号（括号等）为第二优先级大类，运算和赋值符号次之，最后是 ID、NUM、FLO 等。每个大类中理论上不会冲突，但是为了清晰期间还是强制规定每个类别的优先级各不相同，优先级详情请见附录 YAML 文件中的 token_types 部分，数字越小优先级越高。

4.2 单元测试与功能验证

实验采用 GoogleTest 测试框架来分别验证后端的准确性，配合 SPDLOG 库观察程序行为。共设计了 18 个单元测试，下面仅给出直接验证 DFA/FLEX 后端词法分析功能的单元测试。



```
[ RUN      ] YAMLLexerFactoryTest.LoadMinimalCorrectGeneralConfig
[       OK ] YAMLLexerFactoryTest.LoadMinimalCorrectGeneralConfig (0 ms)
[ RUN      ] YAMLLexerFactoryTest.LoadAndCheckMinimalCorrectGeneralConfig
[       OK ] YAMLLexerFactoryTest.LoadAndCheckMinimalCorrectGeneralConfig (0 ms)
[ RUN      ] YAMLLexerFactoryTest.LoadAndCheckMinimalIncorrectGeneralConfig
[       OK ] YAMLLexerFactoryTest.LoadAndCheckMinimalIncorrectGeneralConfig (1 ms)
[ RUN      ] YAMLLexerFactoryTest.LoadMinimalCorrectDFABackendConfig
[       OK ] YAMLLexerFactoryTest.LoadMinimalCorrectDFABackendConfig (0 ms)
[ RUN      ] YAMLLexerFactoryTest.LoadAndCheckMinimalCorrectDFABackendConfig
[       OK ] YAMLLexerFactoryTest.LoadAndCheckMinimalCorrectDFABackendConfig (0 ms)
[ RUN      ] YAMLLexerFactoryTest.LoadAndCheckMinimalIncorrectDFABackendConfig
[       OK ] YAMLLexerFactoryTest.LoadAndCheckMinimalIncorrectDFABackendConfig (0 ms)
[ RUN      ] YAMLLexerFactoryTest.LoadAndCheckMinimalIncorrectGeneralAndDFABackendConfig
[       OK ] YAMLLexerFactoryTest.LoadAndCheckMinimalIncorrectGeneralAndDFABackendConfig (0 ms
)
[ RUN      ] YAMLLexerFactoryTest.LoadMinimalCorrectFlexBackendConfig
[       OK ] YAMLLexerFactoryTest.LoadMinimalCorrectFlexBackendConfig (0 ms)
[----------] 8 tests from YAMLLexerFactoryTest (4 ms total)

[----------] 3 tests from DFABasedLexerTest
[ RUN      ] DFABasedLexerTest.DFABasedLexerTest_ConstructMinimalLexer_Test
[       OK ] DFABasedLexerTest.DFABasedLexerTest_ConstructMinimalLexer_Test (0 ms)
[ RUN      ] DFABasedLexerTest.DFABasedLexerTest_ParseStringToTokens_Test
[       OK ] DFABasedLexerTest.DFABasedLexerTest_ParseStringToTokens_Test (0 ms)
[ RUN      ] DFABasedLexerTest.DFABasedLexerTest_ParseFileToTokens_Test
[       OK ] DFABasedLexerTest.DFABasedLexerTest_ParseFileToTokens_Test (6 ms)
[----------] 3 tests from DFABasedLexerTest (6 ms total)

[----------] 3 tests from FlexBasedLexerTest
[ RUN      ] FlexBasedLexerTest.FlexBasedLexerTest_ConstructMinimalLexer_Test
[       OK ] FlexBasedLexerTest.FlexBasedLexerTest_ConstructMinimalLexer_Test (2 ms)
[ RUN      ] FlexBasedLexerTest.FlexBasedLexerTest_ParseStringToTokens_Test
[       OK ] FlexBasedLexerTest.FlexBasedLexerTest_ParseStringToTokens_Test (0 ms)
[ RUN      ] FlexBasedLexerTest.FlexBasedLexerTest_ParseStringToTokens_Complicated_Test
[       OK ] FlexBasedLexerTest.FlexBasedLexerTest_ParseStringToTokens_Complicated_Test (5 ms)
[----------] 3 tests from FlexBasedLexerTest (8 ms total)

[----------] 3 tests from DFAConfigTest
[ RUN      ] DFAConfigTest.FloatTypeDFATest
[       OK ] DFAConfigTest.FloatTypeDFATest (0 ms)
[ RUN      ] DFAConfigTest.NumberTypeDFATest
[       OK ] DFAConfigTest.NumberTypeDFATest (0 ms)
[ RUN      ] DFAConfigTest.IDTypeDFATest
[       OK ] DFAConfigTest.IDTypeDFATest (0 ms)
[----------] 3 tests from DFAConfigTest (2 ms total)

[----------] Global test environment tear-down
[==========] 18 tests from 5 test suites ran. (24 ms total)
[  PASSED  ] 18 tests.
(base) ruhaotian@Ruhao-TB16p:~/xjtu_compiler_lab/lab2/build$
```

人工智能平台PAI / 交互式建模（DSW） / xjtu_comp_lab

运行中　镜像未保存　服务时长21小时47分

基本信息

实例名称	xjtu_comp_lab
实例 ID	dsw-lpcbwg7e64ilhm0lfr
标签	未绑定标签
资源类型	CPU
创建时间	2025-04-11 15:20:54
更新时间	2025-05-04 18:01:20

资源信息

资源配额　公共资源组（按量付费）

规格名称	GPU卡型	GPU（卡数）	CPU（核数）
ecs.g6.xlarge	-	0	4

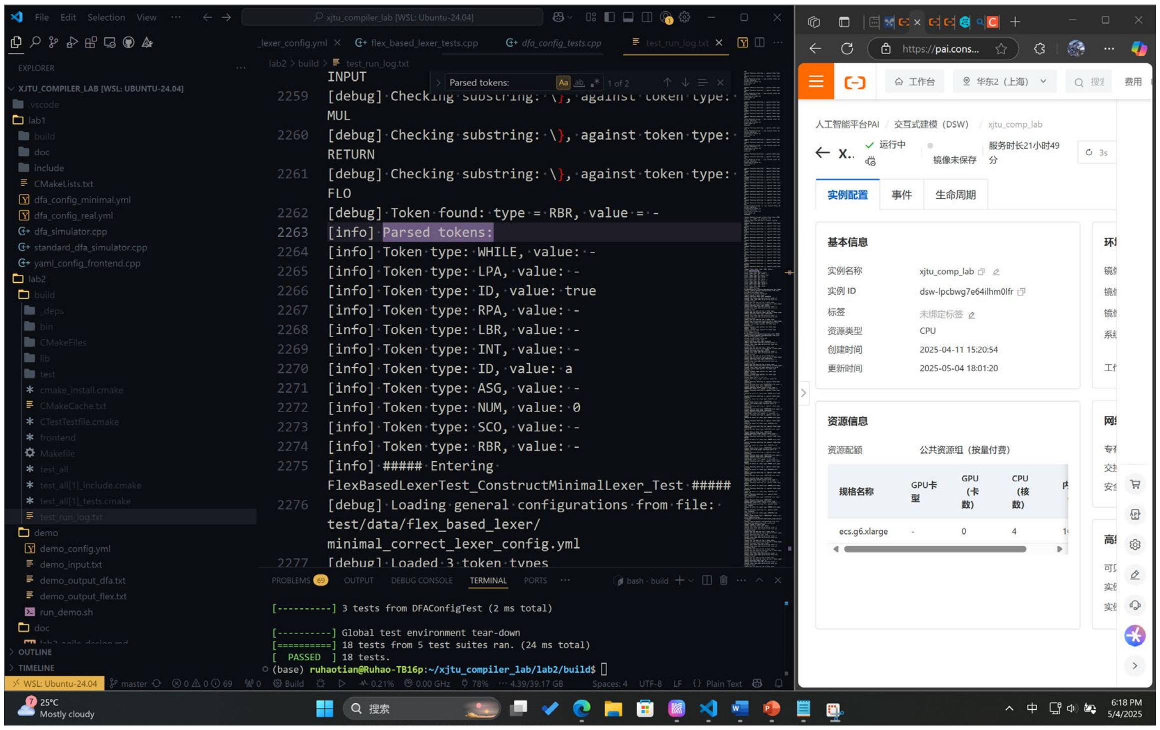
4.2.1　DFA 后端功能验证

用例设计如下：

1. 模拟输入上述配置数据
2. 将 PPT 中的测试语句输入
3. 检查是否报错、匹配到的词法单元是否为 11 个、并使用 spdlog 输出匹配的词法单元。

```
TEST(DFABasedLexerTest, DFABasedLexerTest_ParseFileToTokens_Test){
    spdlog::info("###### Entering DFABasedLexerTest_ParseFileToTokens_Test
######");
    std::string config_file = "test/data/dfa_based_lexer/sim_c_config.yml"
;
    // check if the file exist
    std::ifstream file(config_file);
    ASSERT_TRUE(file.good()) << "File " << config_file << " does not exist
.";

    YAMLLexerFactory lexer_factory;
    auto lexer = lexer_factory.CreateLexer("DFA", config_file, config_file
);

    std::string input = "while \\(true\\) \\{int a\\=0\\;\\}";
    // std::string input = "while";
    LexerResult result = lexer->Parse(input);

    ASSERT_TRUE(result.success);
    ASSERT_EQ(result.tokens.size(), 11);

    spdlog::info("Parsed tokens:");
    for (const auto& token : result.tokens) {
        spdlog::info("Token type: {}, value: {}", token.type, token.value)
;
    }
}
```

上文中已显示单元测试通过，为了确保输出正确，查看单元测试的日志。



```
      INPUT
2259  [debug] Checking substring: \}, against token type:
      MUL
2260  [debug] Checking substring: \}, against token type:
      RETURN
2261  [debug] Checking substring: \}, against token type:
      FLO
2262  [debug] Token found: type = RBR, value = -
2263  [info] Parsed tokens:
2264  [info] Token type: WHILE, value: -
2265  [info] Token type: LPA, value: -
2266  [info] Token type: ID, value: true
2267  [info] Token type: RPA, value: -
2268  [info] Token type: LBR, value: -
2269  [info] Token type: INT, value: -
2270  [info] Token type: ID, value: a
2271  [info] Token type: ASG, value: -
2272  [info] Token type: NUM, value: 0
2273  [info] Token type: SCO, value: -
2274  [info] Token type: RBR, value: -
2275  [info] ##### Entering
      FlexBasedLexerTest_ConstructMinimalLexer_Test #####
2276  [debug] Loading general configurations from file:
      test/data/flex_based_lexer/
      minimal_correct_lexer_config.yml
2277  [debug] Loaded 3 token types
```

```
[----------] 3 tests from DFAConfigTest (2 ms total)

[----------] Global test environment tear-down
[==========] 18 tests from 5 test suites ran. (24 ms total)
[  PASSED  ] 18 tests.
(base) ruhaotian@Ruhao-TB16p:~/xjtu_compiler_lab/lab2/build$
```

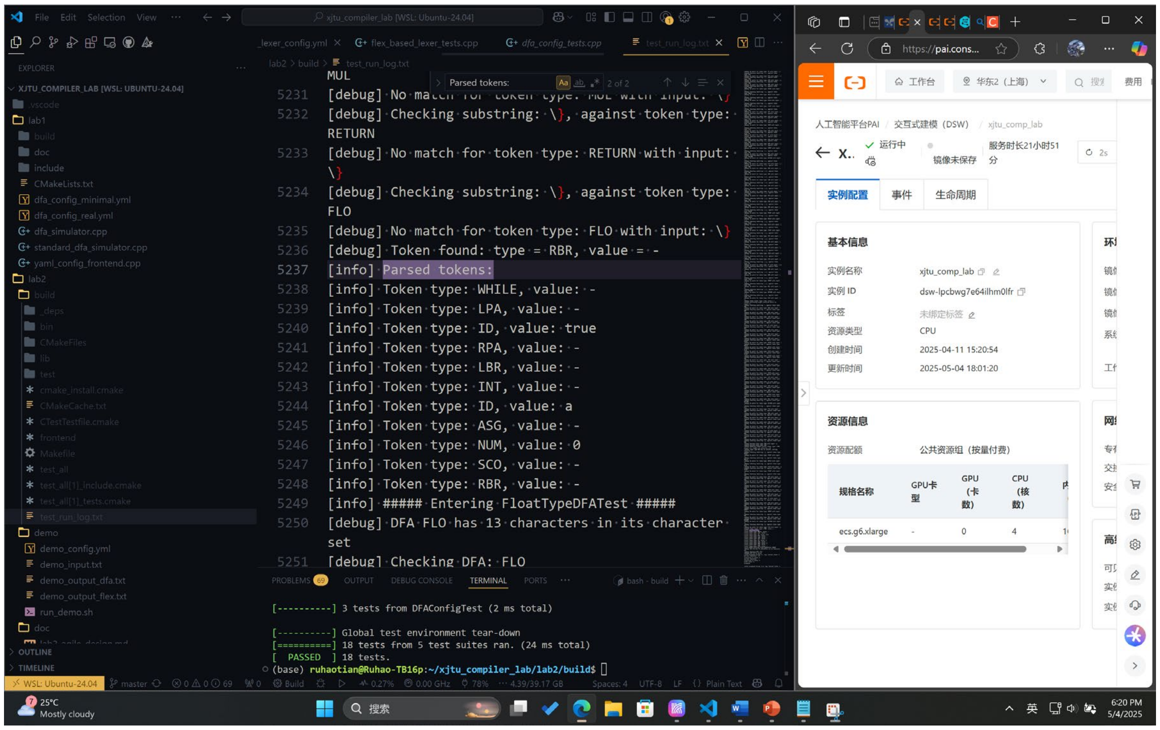
在 2263 行可见 Token 都可以正确输出（特别注意 PPT 中的文法没有定义 True，所以这里 True 识别为了 ID）

4.2.2 FLEX 后端功能验证

用例的设计和 DFA 后端完全一致，只是对象换成了 FLEX 后端，详见下述程序：

```
TEST(FlexBasedLexerTest, FlexBasedLexerTest_ParseStringToTokens_Complicate
d_Test) {
    spdlog::info("##### Entering FlexBasedLexerTest_ParseStringToTokens_Co
mplicated_Test #####");

    std::string config_file = "test/data/flex_based_lexer/sim_c_config.yml
";
    // check if the file exist
    std::ifstream file(config_file);
    ASSERT_TRUE(file.good()) << "File " << config_file << " does not exist
.";
```

```
    YAMLLexerFactory lexer_factory;
    auto lexer = lexer_factory.CreateLexer("FLEX", config_file, config_fil
e);

    std::string input = "while \\(true\\) \\{int a\\=0\\;\\}";
    LexerResult result = lexer->Parse(input);

    ASSERT_TRUE(result.success);
    ASSERT_EQ(result.tokens.size(), 11);
    spdlog::info("Parsed tokens:");
    for (const auto& token : result.tokens) {
        spdlog::info("Token type: {}, value: {}", token.type, token.value)
;
    }
}
```
结果见 5237 行。


```
     MUL
5231 [debug] No match for token type: MUL with input: \}
5232 [debug] Checking substring: \}, against token type:
     RETURN
5233 [debug] No match for token type: RETURN with input:
     \}
5234 [debug] Checking substring: \}, against token type:
     FLO
5235 [debug] No match for token type: FLO with input: \}
5236 [debug] Token found: type = RBR, value = -
5237 [info] Parsed tokens:
5238 [info] Token type: WHILE, value: -
5239 [info] Token type: LPA, value: -
5240 [info] Token type: ID, value: true
5241 [info] Token type: RPA, value: -
5242 [info] Token type: LBR, value: -
5243 [info] Token type: INT, value: -
5244 [info] Token type: ID, value: a
5245 [info] Token type: ASG, value: -
5246 [info] Token type: NUM, value: 0
5247 [info] Token type: SCO, value: -
5248 [info] Token type: RBR, value: -
5249 [info] ##### Entering FloatTypeDFATest #####
5250 [debug] DFA FLO has 13 characters in its character
     set
5251 [debug] Checking DFA: FLO
```

```
[----------] 3 tests from DFAConfigTest (2 ms total)

[----------] Global test environment tear-down
[==========] 18 tests from 5 test suites ran. (24 ms total)
[  PASSED  ] 18 tests.
(base) ruhaotian@Ruhao-TB16p:~/xjtu_compiler_lab/lab2/build$
```

人工智能平台PAI / 交互式建模（DSW） / xjtu_comp_lab

运行中　镜像未保存　服务时长21小时51分

基本信息

实例名称	xjtu_comp_lab
实例 ID	dsw-lpcbwg7e64ilhm0lfr
标签	未绑定标签
资源类型	CPU
创建时间	2025-04-11 15:20:54
更新时间	2025-05-04 18:01:20

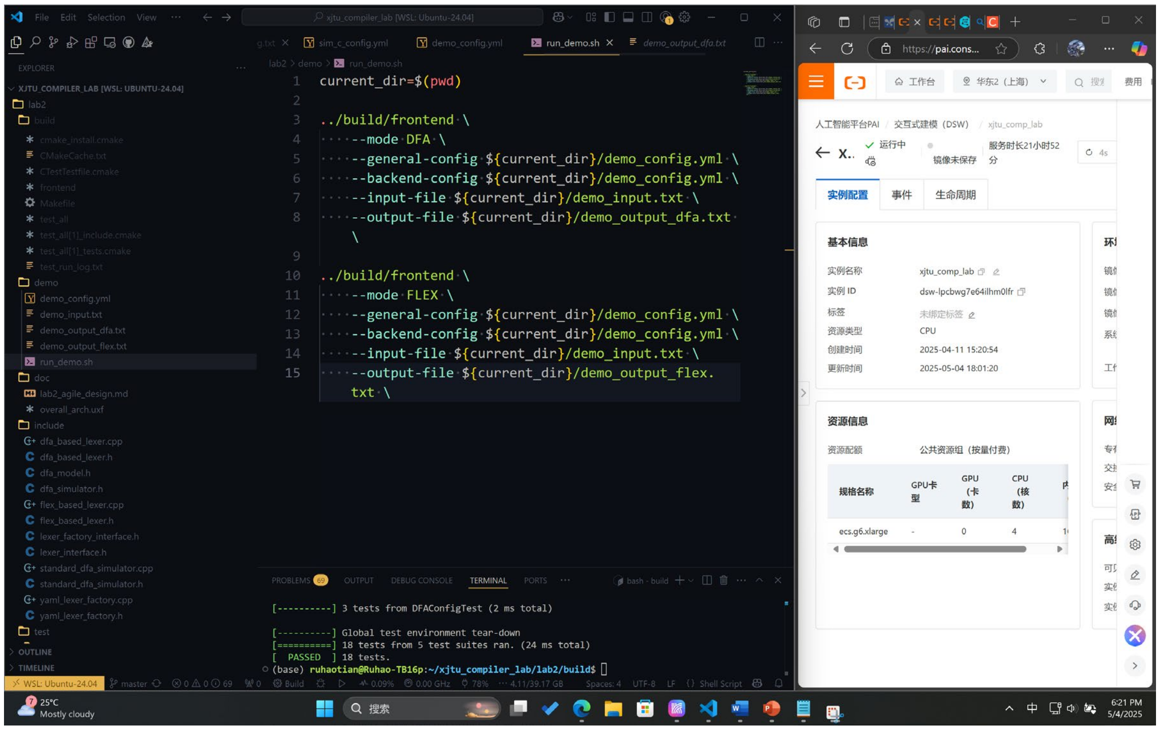
资源信息

资源配额　公共资源组（按量付费）

规格名称	GPU卡型	GPU（卡数）	CPU（核数）
ecs.g6.xlarge	-	0	4

4.3 运行 CLI

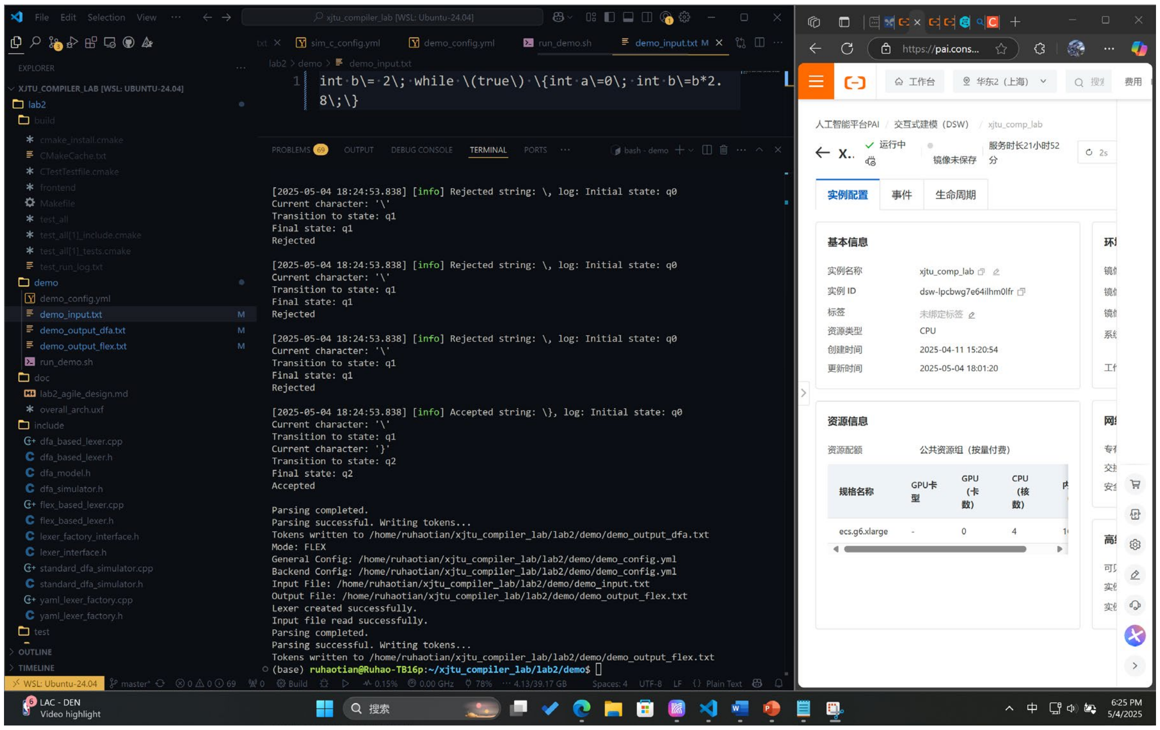
下面展示命令行运行过程，在 Shell 脚本中调用前端 CLI，指定好后端、配置、输入和输出文件。



```
current_dir=$(pwd)

../build/frontend \
    --mode DFA \
    --general-config ${current_dir}/demo_config.yml \
    --backend-config ${current_dir}/demo_config.yml \
    --input-file ${current_dir}/demo_input.txt \
    --output-file ${current_dir}/demo_output_dfa.txt \

../build/frontend \
    --mode FLEX \
    --general-config ${current_dir}/demo_config.yml \
    --backend-config ${current_dir}/demo_config.yml \
    --input-file ${current_dir}/demo_input.txt \
    --output-file ${current_dir}/demo_output_flex.txt \
```

```
[----------] 3 tests from DFAConfigTest (2 ms total)

[----------] Global test environment tear-down
[==========] 18 tests from 5 test suites ran. (24 ms total)
[  PASSED  ] 18 tests.
(base) ruhaotian@Ruhao-TB16p:~/xjtu_compiler_lab/lab2/build$
```

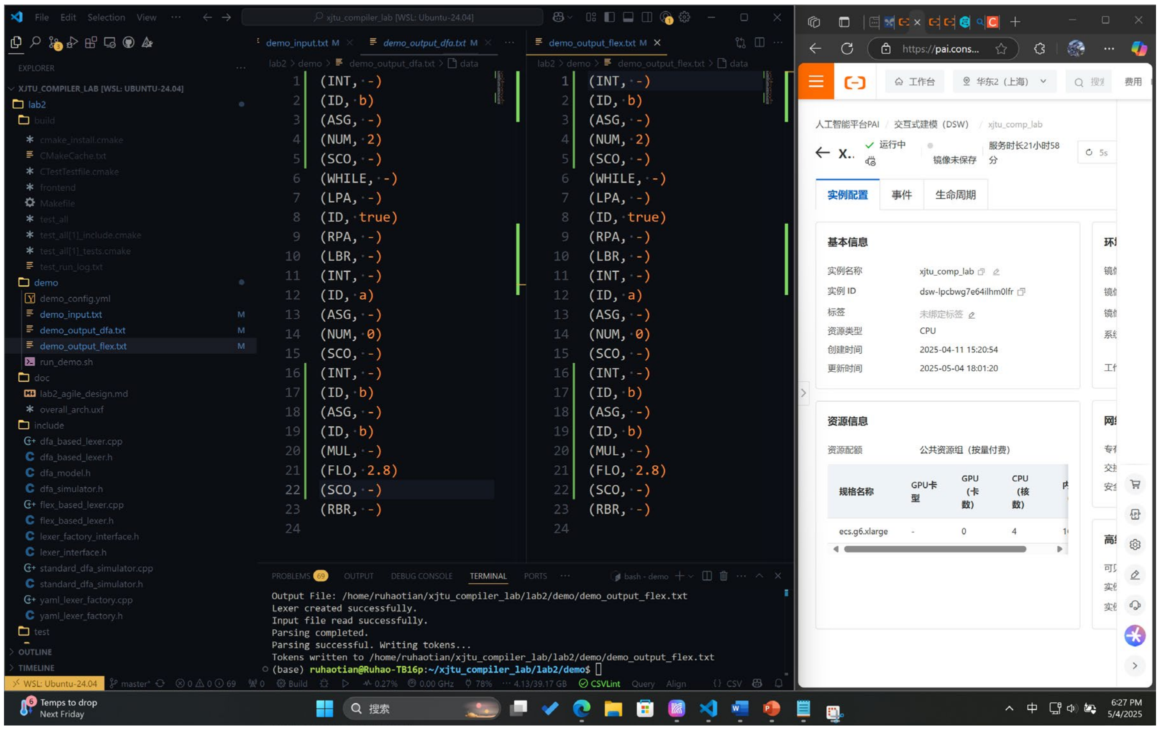
其中 demo_input 的内容如下：



```
int b\= 2\; while \(true\) \{int a\=0\; int b\=b*2.8\;\}
```

```
[2025-05-04 18:24:53.838] [info] Rejected string: \, log: Initial state: q0
Current character: '\'
Transition to state: q1
Final state: q1
Rejected

[2025-05-04 18:24:53.838] [info] Rejected string: \, log: Initial state: q0
Current character: '\'
Transition to state: q1
Final state: q1
Rejected

[2025-05-04 18:24:53.838] [info] Rejected string: \, log: Initial state: q0
Current character: '\'
Transition to state: q1
Final state: q1
Rejected

[2025-05-04 18:24:53.838] [info] Accepted string: \}, log: Initial state: q0
Current character: '\'
Transition to state: q1
Current character: '}'
Transition to state: q2
Final state: q2
Accepted

Parsing completed.
Parsing successful. Writing tokens...
Tokens written to /home/ruhaotian/xjtu_compiler_lab/lab2/demo/demo_output_dfa.txt
Mode: FLEX
General Config: /home/ruhaotian/xjtu_compiler_lab/lab2/demo/demo_config.yml
Backend Config: /home/ruhaotian/xjtu_compiler_lab/lab2/demo/demo_config.yml
Input File: /home/ruhaotian/xjtu_compiler_lab/lab2/demo/demo_input.txt
Output File: /home/ruhaotian/xjtu_compiler_lab/lab2/demo/demo_output_flex.txt
Lexer created successfully.
Input file read successfully.
Parsing completed.
Parsing successful. Writing tokens...
Tokens written to /home/ruhaotian/xjtu_compiler_lab/lab2/demo/demo_output_flex.txt
(base) ruhaotian@Ruhao-TB16p:~/xjtu_compiler_lab/lab2/demo$
```

可见运行成功，接下来查看 demo_output_dfa.txt 和 demo_output_flex.txt，分别是两个后端的输出结果。


demo_output_dfa.txt

```
(INT, -)
(ID, b)
(ASG, -)
(NUM, 2)
(SCO, -)
(WHILE, -)
(LPA, -)
(ID, true)
(RPA, -)
(LBR, -)
(INT, -)
(ID, a)
(ASG, -)
(NUM, 0)
(SCO, -)
(INT, -)
(ID, b)
(ASG, -)
(ID, b)
(MUL, -)
(FLO, 2.8)
(SCO, -)
(RBR, -)
```

demo_output_flex.txt

```
(INT, -)
(ID, b)
(ASG, -)
(NUM, 2)
(SCO, -)
(WHILE, -)
(LPA, -)
(ID, true)
(RPA, -)
(LBR, -)
(INT, -)
(ID, a)
(ASG, -)
(NUM, 0)
(SCO, -)
(INT, -)
(ID, b)
(ASG, -)
(ID, b)
(MUL, -)
(FLO, 2.8)
(SCO, -)
(RBR, -)
```

```
Output File: /home/ruhaotian/xjtu_compiler_lab/lab2/demo/demo_output_flex.txt
Lexer created successfully.
Input file read successfully.
Parsing completed.
Parsing successful. Writing tokens...
Tokens written to /home/ruhaotian/xjtu_compiler_lab/lab2/demo/demo_output_flex.txt
(base) ruhaotian@Ruhao-TB16p:~/xjtu_compiler_lab/lab2/demo$ 
```

可见其均可以正确识别。

5 附录

5.1 关于复现

源码都在压缩包中的 lab2 文件夹下。按照 cmakelist 中的 find_package 安装 yaml-cpp、reflex 等第三方库，然后用 cmake 编译即可。编译后的两个可执行文件分别是 frontend（cli）和 test_all（单元测试），在 demo 文件夹下展示了上述 cli 命令示例。单元测试的 log 会输出到 build 文件夹中的 test_run_log.txt。

5.2 PPT 中文法对应的 YAML 配置文件

以下是转换为程序可接受格式的原始配置文件：（即 demo 文件夹中的 yml 配置文件）

```yaml
vars: &vars
  - var_a_to_z: &var_a_to_z "abcdefghijklmnopqrstuvwxyz"
  - var_A_to_Z: &var_A_to_Z "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
  - var_0_to_9: &var_0_to_9 "0123456789"
  - var_id_char: &var_id_char "abcdefghijklmnopqrstuvwxyz0123456789"
  - var_num_char: &var_num_char "0123456789+-"
  - var_float_char: &var_float_char "0123456789+-."
token_types:
- name: "ID"
  priority: 41
  has_content: true
- name: "NUM"
```

```yaml
  priority: 31
  has_content: true
- name: "FLO"
  priority: 32
  has_content: true
- name: "ADD"
  priority: 21
  has_content: false
- name: "MUL"
  priority: 22
  has_content: false
- name: "ROP"
  priority: 23
  has_content: true
- name: "ASG"
  priority: 24
  has_content: false
- name: "LPA"
  priority: 11
  has_content: false
- name: "RPA"
  priority: 12
  has_content: false
- name: "LBK"
  priority: 13
  has_content: false
- name: "RBK"
  priority: 14
  has_content: false
- name: "LBR"
  priority: 15
  has_content: false
- name: "RBR"
  priority: 16
  has_content: false
- name: "CMA"
  priority: 17
  has_content: false
- name: "SCO"
  priority: 18
  has_content: false
- name: "INT"
  priority: 1
  has_content: false
- name: "FLOAT"
  priority: 2
  has_content: false
- name: "IF"
  priority: 3
  has_content: false
- name: "ELSE"
  priority: 4
  has_content: false
- name: "WHILE"
  priority: 5
  has_content: false
- name: "RETURN"
  priority: 6
  has_content: false
- name: "INPUT"
  priority: 7
  has_content: false
- name: "PRINT"
  priority: 8
  has_content: false
- name: "VOID"
  priority: 9
  has_content: false
dfas:
- name: "FLO"
  character_set: *var_float_char
  states_set: [ "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K" ]
  initial_state: "A"
  accepting_states: [ "G", "H", "I", "J", "K" ]
  transitions:
  - from: "A"
    to: "B"
    character: "+"
  - from: "A"
    to: "C"
    character: "-"
  - from: "A"
    to: "D"
    character: *var_0_to_9
  - from: "A"
    to: "E"
    character: "."
  - from: "B"
    to: "D"
    character: *var_0_to_9
  - from: "B"
    to: "E"
    character: "."
  - from: "C"
    to: "D"
    character: *var_0_to_9
  - from: "C"
    to: "E"
    character: "."
  - from: "D"
    to: "F"
    character: *var_0_to_9
  - from: "D"
    to: "G"
    character: "."
  - from: "E"
    to: "H"
    character: *var_0_to_9
  - from: "F"
    to: "F"
    character: *var_0_to_9
  - from: "F"
    to: "G"
    character: "."
  - from: "G"
    to: "I"
    character: *var_0_to_9
  - from: "H"
    to: "J"
    character: *var_0_to_9
  - from: "I"
    to: "K"
    character: *var_0_to_9
  - from: "J"
    to: "J"
    character: *var_0_to_9
  - from: "K"
    to: "K"
    character: *var_0_to_9
- name: "ID"
  character_set: *var_id_char
  states_set: [ "q0", "q1" ]
  initial_state: "q0"
  accepting_states: [ "q1" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: *var_a_to_z
  - from: "q1"
    to: "q1"
    character: *var_0_to_9
  - from: "q1"
    to: "q1"
    character: *var_a_to_z
- name: "NUM"
  character_set: *var_num_char
  states_set: [ "A", "B", "C", "D" ]
  initial_state: "A"
  accepting_states: [ "D" ]
  transitions:
  - from: "A"
    to: "B"
    character: "+"
  - from: "A"
    to: "C"
    character: "-"
  - from: "A"
    to: "D"
    character: *var_0_to_9
  - from: "B"
    to: "D"
    character: *var_0_to_9
  - from: "C"
    to: "D"
    character: *var_0_to_9
  - from: "D"
    to: "D"
    character: *var_0_to_9
- name: "ADD"
  character_set: "+"
  states_set: [ "q0", "q1" ]
  initial_state: "q0"
  accepting_states: [ "q1" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "+"
- name: "MUL"
  character_set: "*"
  states_set: [ "q0", "q1" ]
  initial_state: "q0"
  accepting_states: [ "q1" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "*"
- name: "ROP"
  character_set: "<="
  states_set: [ "q0", "q1", "q2", "q3" ]
  initial_state: "q0"
  accepting_states: [ "q1", "q3" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "<"
  - from: "q0"
    to: "q2"
    character: "="
  - from: "q1"
    to: "q3"
    character: "="
  - from: "q2"
    to: "q3"
    character: "="
- name: "ASG"
  character_set: "\\="
  states_set: [ "q0", "q1", "q2" ]
  initial_state: "q0"
  accepting_states: [ "q2" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "\\"
  - from: "q1"
    to: "q2"
    character: "="
- name: "LPA"
  character_set: "\\("
  states_set: [ "q0", "q1", "q2" ]
  initial_state: "q0"
  accepting_states: [ "q2" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "\\"
  - from: "q1"
    to: "q2"
    character: "("
- name: "RPA"
  character_set: "\\)"
  states_set: [ "q0", "q1", "q2" ]
  initial_state: "q0"
  accepting_states: [ "q2" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "\\"
  - from: "q1"
    to: "q2"
    character: ")"
- name: "LBK"
  character_set: "["
  states_set: [ "q0", "q1" ]
  initial_state: "q0"
  accepting_states: [ "q1" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "["
- name: "RBK"
  character_set: "]"
  states_set: [ "q0", "q1" ]
  initial_state: "q0"
  accepting_states: [ "q1" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "]"
- name: "LBR"
  character_set: "\\{"
  states_set: [ "q0", "q1", "q2" ]
  initial_state: "q0"
  accepting_states: [ "q2" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "\\"
  - from: "q1"
    to: "q2"
    character: "{"
- name: "RBR"
  character_set: "\\}"
  states_set: [ "q0", "q1", "q2" ]
  initial_state: "q0"
  accepting_states: [ "q2" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "\\"
  - from: "q1"
    to: "q2"
    character: "}"
- name: "CMA"
  character_set: "\\,"
  states_set: [ "q0", "q1", "q2" ]
  initial_state: "q0"
  accepting_states: [ "q2" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "\\"
  - from: "q1"
    to: "q2"
    character: ","
- name: "SCO"
  character_set: "\\;"
  states_set: [ "q0", "q1", "q2" ]
  initial_state: "q0"
  accepting_states: [ "q2" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "\\"
  - from: "q1"
    to: "q2"
    character: ";"
- name: "INT"
  character_set: "int"
  states_set: [ "q0", "q1", "q2", "q3" ]
  initial_state: "q0"
  accepting_states: [ "q3" ]
  transitions:
```

```yaml
  - from: "q0"
    to: "q1"
    character: "i"
  - from: "q1"
    to: "q2"
    character: "n"
  - from: "q2"
    to: "q3"
    character: "t"
- name: "FLOAT"
  character_set: "float"
  states_set: [ "q0", "q1", "q2", "q3", "q4", "q5" ]
  initial_state: "q0"
  accepting_states: [ "q5" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "f"
  - from: "q1"
    to: "q2"
    character: "l"
  - from: "q2"
    to: "q3"
    character: "o"
  - from: "q3"
    to: "q4"
    character: "a"
  - from: "q4"
    to: "q5"
    character: "t"
- name: "IF"
  character_set: "if"
  states_set: [ "q0", "q1", "q2" ]
  initial_state: "q0"
  accepting_states: [ "q2" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "i"
  - from: "q1"
    to: "q2"
    character: "f"
- name: "ELSE"
  character_set: "else"
  states_set: [ "q0", "q1", "q2", "q3", "q4" ]
  initial_state: "q0"
  accepting_states: [ "q4" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "e"
  - from: "q1"
```

```yaml
    to: "q2"
    character: "l"
  - from: "q2"
    to: "q3"
    character: "s"
  - from: "q3"
    to: "q4"
    character: "e"
- name: "WHILE"
  character_set: "while"
  states_set: [ "q0", "q1", "q2", "q3", "q4", "q5" ]
  initial_state: "q0"
  accepting_states: [ "q5" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "w"
  - from: "q1"
    to: "q2"
    character: "h"
  - from: "q2"
    to: "q3"
    character: "i"
  - from: "q3"
    to: "q4"
    character: "l"
  - from: "q4"
    to: "q5"
    character: "e"
- name: "RETURN"
  character_set: "return"
  states_set: [ "q0", "q1", "q2", "q3", "q4", "q5", "q6" ]
  initial_state: "q0"
  accepting_states: [ "q6" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "r"
  - from: "q1"
    to: "q2"
    character: "e"
  - from: "q2"
    to: "q3"
    character: "t"
  - from: "q3"
    to: "q4"
    character: "u"
  - from: "q4"
    to: "q5"
    character: "r"
  - from: "q5"
    to: "q6"
    character: "n"
- name: "INPUT"
  character_set: "input"
  states_set: [ "q0", "q1", "q2", "q3", "q4", "q5" ]
  initial_state: "q0"
  accepting_states: [ "q5" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "i"
  - from: "q1"
    to: "q2"
    character: "n"
  - from: "q2"
    to: "q3"
    character: "p"
  - from: "q3"
    to: "q4"
    character: "u"
  - from: "q4"
    to: "q5"
    character: "t"
- name: "PRINT"
  character_set: "print"
  states_set: [ "q0", "q1", "q2", "q3", "q4", "q5" ]
  initial_state: "q0"
  accepting_states: [ "q4" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "p"
  - from: "q1"
    to: "q2"
    character: "r"
  - from: "q2"
    to: "q3"
    character: "i"
  - from: "q3"
    to: "q4"
    character: "n"
  - from: "q4"
    to: "q5"
    character: "t"
- name: "VOID"
  character_set: "void"
  states_set: [ "q0", "q1", "q2", "q3", "q4" ]
  initial_state: "q0"
  accepting_states: [ "q4" ]
  transitions:
  - from: "q0"
    to: "q1"
    character: "v"
```

```yaml
  - from: "q1"
    to: "q2"
    character: "o"
  - from: "q2"
    to: "q3"
    character: "i"
  - from: "q3"
    to: "q4"
    character: "d"
regexps:
- regexp: "^[a-z][a-z||0-9]*$" # flex regex syntax
  token_type: "ID"
- regexp: "^[+-]?[0-9]+$"
  token_type: "NUM"
- regexp: "^([+-]?[0-9]*\\.[0-9]+)|([+-]?[0-9]+\\.[0-9]*)$"
  token_type: "FLO"
- regexp: "^\\+$"
  token_type: "ADD"
- regexp: "^\\*$"
  token_type: "MUL"
- regexp: "^[(<)||[(<=)||(==)]]$"
  token_type: "ROP"
- regexp: "^\\\\=$"
  token_type: "ASG"
- regexp: "^\\\\\\($"
  token_type: "LPA"
- regexp: "^\\\\\\)$"
  token_type: "RPA"
- regexp: "^\\\\\\[$"
  token_type: "LBK"
- regexp: "^\\\\\\]$"
  token_type: "RBK"
- regexp: "^\\\\\\{$"
  token_type: "LBR"
- regexp: "^\\\\\\}$"
  token_type: "RBR"
- regexp: "^\\\\,$"
  token_type: "CMA"
- regexp: "^\\\\;$"
  token_type: "SCO"
- regexp: "^int$"
  token_type: "INT"
- regexp: "^float$"
  token_type: "FLOAT"
- regexp: "^if$"
  token_type: "IF"
- regexp: "^else$"
  token_type: "ELSE"
- regexp: "^while$"
  token_type: "WHILE"
- regexp: "^return$"
  token_type: "RETURN"
```

```yaml
- regexp: "^input$"
  token_type: "INPUT"
- regexp: "^print$"
  token_type: "PRINT"
- regexp: "^void$"
  token_type: "VOID"
```