基于Flex和Bison的CMinus编译器

计算机科学与技术 2022302111469 王垚

1. 实验环境

本实验在Windows环境下通过WSL运行的Ubuntu22.04操作系统进行开发，以下是详细的实验配置：  
（1）操作系统：Linux (Ubuntu22.04)

1. 工具链版本: Bison: 3.8.2 ; Flex: 2.6.4; GDB: 12.1; G++: 11.4.0
2. 项目结构：

compile\_exm/

├ main.cpp // 主程序入口，负责调用词法分析器和语法分析器

├ ASTCommon.hpp // 公共头文件，定义语法树节点结构和辅助函数

├ cminus.l // Flex 词法分析器定义文件，定义词法规则

├ cminus.y // Bison 语法分析器定义文件，定义语法规则

├ Makefile // 自动化构建脚本，用于编译项目

├ lex.yy.c // Flex 生成的词法分析器代码

├ cminus.tab.c // Bison 生成的语法分析器代码

├ cminus.tab.h // Bison 生成的语法分析器头文件  
├ testx.in //测试文件(x表示1.2.3 可以手动添加其他测试文件)

├ .vscode/

├── tasks.json

├── launch.json

├── c\_cpp\_properties.json

├── settings.json

1. 功能介绍
2. 能够识别出CMINUS词法中未定义的字符以及不符合词法单元定义的字符，并能够错误类型A的提示信息。

例如程序能够识别出以下的函数声明中的未定义字符，并给出相应的输出

1 int j=~i;

2 int k=@;

Error type A at Line 1: Mysterious character "~"

Error type A at Line 2: Mysterious character "@"

Error type B at Line 2: Missing ";"

第二行的这个错误其实只是因为@是一个非法字符，但是输出缺失分号的语法错误，是因为@不能识别成一个有效的TOKEN,Bison无法规约表达式，只能通过error SEMI恢复，所以会报错，但是这已经是Bison 的标准错误恢复机制，故没有进一步修改。

1. 能够识别不符合词法定义的十进制数（12abc）、八进制数（09）和十六进制数（0x1G）,并输出错误类型A的提示信息。

int a = 12abc; // 错误，十进制数后不能有字母

int b = 09; // 错误，八进制数不能有8或9

int c = 0x1G; // 错误，十六进制数不能有G

int d = 0x; // 错误，十六进制数缺少数字

Error type A at Line 1: "12abc" is not a valid decimal int integer

Error type A at Line 2: "09" is not a valid octal int integer

Error type A at Line 3: "0x1G" is not a valid hexadecimal int integer

Error type A at Line 4: "0x" is not a valid hexadecimal int integer

不过我的匹配规则是如果开头是0并且后面没有跟x,那么就直接匹配成八进制数；如果是以0x开头，那么就匹配成十六进制数，最后根据stofChange函数的结果进行是否合法的判断，并且不会多报出语法错误。虽然这样能够很好地满足需求，但是一定程度上影响了正则表达式的可读性。这样的好处在于能够把输入的这组字符全部读取完，并且防止额外词义的产生。

1. 能够识别指数形式的浮点数，对于不符合词法定义的指数类型的浮点数（1.05e）能够输出错误类型A的提示信息。特别说明，在c++中，形如’123.’,’.123’,’1.23’这样的浮点数都是合法的，所以这里直接假设成都合法。

1 float d = 1.05e; // 错误，缺少指数部分

2 float e = 2.3E+; // 错误，缺少指数数字

3 float f = 4.5e-; // 错误，缺少指数数字

Error type A at Line 1: "1.05e" is not a valid float

Error type A at Line 2: "2.3E+" is not a valid float

Error type A at Line 3: "4.5e-" is not a valid float

1. 能够识别“//”和“/\*\*/”形式的注释：当输出文件包含符合定义的注释形式时，程序能够滤除这些注释；当输出文件中包含不符合定义的注释，主要指的是嵌套多行注释，能够输出错误类型A的提示信息，并给出当前的嵌套深度。

1 /\* 这是一个嵌套的多行注释 /\* 嵌套开始 \*/

2 int d = 40; // 嵌套结束 \*/

Error type A at Line 1: Nested comments are not allowed. Current depth: 2

1 /\* 这是一个嵌套的多行注释 /\* 嵌套开始 \*/

2 int d = 40;

Error type A at Line 1: Nested comments are not allowed. Current depth: 2

Error type A at Line 2: Unterminated comment. Current depth: 1

说明：这个未闭合的’/\*’的行号是以程序的最后一行（包含有换行符的行）对应输出，’/\*’嵌套的错误是都会输出，但是未闭合的只会输出最后一个。

1. 构建语法树并输出：对于没有检测出语法错误和词法错误的程序，能够用先序遍历的方式打印出语法树每个结点的信息。

1 int a=0;

Program (1)

ExtDefList (1)

ExtDef (1)

Specifier (1)

TYPE: int

ExtDecList (1)

Dec (1)

VarDec (1)

ID: a

ASSIGNOP

Exp (1)

INT: 0

SEMI

说明：为了使编译测试用例时，能够输出跟《课程设计说明》中一模一样的信息，我在bison中没有通过yyerror函数直接进行错误输出（比较笼统，不能准确输出错误的信息），而是选择直接在错误对应的地方，比如缺少’)’’]’等识别的地方进行输出。

1. **运行方法**

**3.1编译方法**

**3.1.1 方法1: 使用Makefile文件进行编译（推荐）**

·使用make命令编译项目；

make

·使用make clean清除项目中生成的中间文件和可执行文件

make clean

**3.1.2 方法2：手动生成词法和语法分析器**

在终端中依次输入这些命令

flex cminus.l //生成词法分析器

bison -d cminus.y //生成语法分析器

g++ lex.yy.c cminus.tab.c main.cpp -o cminus --std=c++11 //生成可执行文件cminus

**3.2运行用例**

使用以下命令运行编译生成的可执行文件cminus，并指定测试文件test.in作为输入:

./cminus test.in

其中，cminus是项目生成的可执行文件，test.in是测试文件的名称（可根据需要修改），运行完后在终端中输出结果。