【词法分析和语法分析】编译原理实验一（hit）2022-lab1

原创

芝麻。 已于 2022-04-02 13:09:37 修改 2782 收藏 20

分类专栏： 编译原理实验 文章标签： ubuntu linux 经验分享

于 2022-03-15 19:53:23 首次发布

本文链接：https://blog.csdn.net/kar1yn/article/details/123499513

编译原理实验 专栏收录该内容

1 篇文章 2 订阅

订阅专栏

词法分析与语法分析

环境配置

实验指导书要求虚拟机版本为Ubuntu12.04，但是现在Ubuntu现在已经是20.04版本了，所以我尝试了安装12.04的Ubuntu，但是已经无法通过apt-get来进行安装flex了，所以最终还是选择可20.04版本。

之前的20.04版本已经换过清华源，但是flex是无法通过清华源下载的，由于之前忘记保存原来的源了，所以被逼无奈只能重新安装虚拟机。

flex

实验指导书要求的flex版本是2.5.35，但是尝试安装之后发现无法检索到该版本，所以自动选择最新版本。
Ctrl+Alt+T打开终端
然后进行命令运行：

```
sudo apt-get install flex
```

会提示一些包的安装失败
根据Terminal的提示，使用命令行：

```
sudo apt-get install flex --fix-missing
```

接下来一路yes就行了
安装完之后，可以试着输入

```
flex -V
```

就可以查看flex的版本号，顺便检验一下是否安装完成
我这里的版本号是2.6.4

```
karlyn@ubuntu:~$ flex -V
flex 2.6.4
```

bison

没有什么不同，基本与flex流程一样，Terminal输入：

```
sudo apt-get install bison
```

然后输入

```
bison -V
```

查看版本，顺便检查是否安装完成
我这里的版本是3.5.1

```
karlyn@ubuntu:~$ bison -V
bison (GNU Bison) 3.5.1
Written by Robert Corbett and Richard Stallman.

Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is N
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOS
```

GCC

```
sudo apt-get install build-essential
```

接下来照例是查看版本

```
gcc --version
```

我的版本是9.4.0

```
karlyn@ubuntu:~$ gcc --version
gcc (Ubuntu 9.4.0-1ubuntu1~20.04) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE
.
```

一些废话

对于09年翻译的不知道什么时候出版的实验书，我只能很抱歉的说做到哪里停止我也不太确定，因为无法确信最新的版本是否能兼容之前的版本，同时我也对hit现在才开始使用这本实验书感到疑惑。当然实验书说了只要不是很离谱的调用，GCC之类的不同版本兼容应该不会产生问题。
对这门课感兴趣的同学可以去b站看看陈鄞老师的课，老师讲的很细致。

实验内容

> 编写一个程序
> 对使用C–语言书写的源代码进行词法和语法分析(C–语言的文法参见《编译原理实践与指导教程》附录A)，并打印分析结果。
> 实验要求使用词法分析工具GNU Flex和语法分析工具GNU Bison，并使用C语言来完成。
> 程序要能够查出C–源代码中可能包含的下述几类错误:
> 1)词法错误(错误类型A):
> 出现C–词法中未定义的字符以及任何不符合C–词法单元定义的字符;
> 2)语法错误(错误类型B)。

词法分析

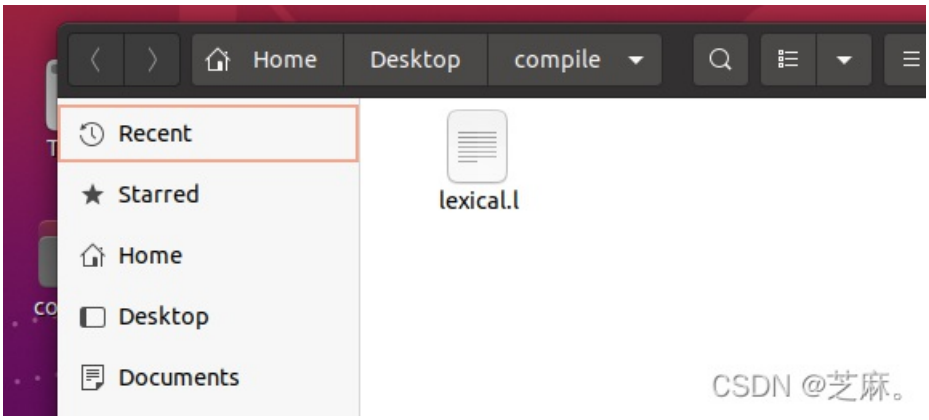
实验指导书中那一长串的内容我就不一一复述粘贴了，下面直接进入到flex代码的编写阶段：
按照实验指导书的说法
我们需要新建一个flex文件。
首先为了防止文件混乱的显示，我们先在桌面上创建一个文件夹吧。

```
ls
cd Desktop
mkdir compile
cd compile
touch lexical.l
```

流程如下

```
karlyn@ubuntu:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
karlyn@ubuntu:~$ cd Desktop
karlyn@ubuntu:~/Desktop$ mkdir compile
karlyn@ubuntu:~/Desktop$ ls
compile
karlyn@ubuntu:~/Desktop$ cd compile
karlyn@ubuntu:~/Desktop/compile$ touch lexical.l
karlyn@ubuntu:~/Desktop/compile$ ls
lexical.l
```

然后在桌面就能看到文件夹了



然后看你自己习惯决定是vim打开还是双击打开。
接下来就是词法分析的核心部分了。
首先需要了解Flex源代码的文件，主要包含三个部分

```
{definitions}
%%
{rules}
%%
{user subroutines}
```

我们先从定义部分开始：

定义部分

如果学过正则定义的同学在这一块看起来会比较轻松，就是将一些正则表达式取一些别名，方便后面使用的便利性。
那么这里就要简单学一下正则表达式的书写了。
不多赘述，可以看看实验指导书，我把一些重要的罗列在下面：

> 1.“.”匹配除了换行符号”\n"之外的任意字符
> 2.[]匹配一个字符类，类似于一个集合，只要集合中任意元素被匹配了，那么就算整个集合都匹配了，如果要取补的话，只需要在[]里面最前面加上"^";
> 3.行首行尾的匹配："^“用于[]之外会匹配一行的开头，”$“会匹配一行的结尾，“<<“EOF”>>”（里面没有引号）会匹配文件的结尾
> 4.{}表示出现的次数，比如A{1,3}表示A或AA或AAA
> 5.”*“表示闭包操作，”+“表示正闭包操作
> 6.”?“匹配零个或一个表达式，举个例子：-?1匹配的是1或者-1
> 7.”|“作为选择操作
> 8.”“转义符，具体用法和C相似
> 9.”"“匹配被引起来的字符，比如”…“匹配的就是…而不是三个任意字符。
> 10.选择匹配符：”/"，比如0/1匹配的是01中的0，但是不匹配0/2中的0

下面我们进行一些简单的正则定义的书写

```
/* regular definitions 正则定义*/
int [1-9]+[0-9]{0,31}|0
int8 0[0-9]{3}
int16 0x[0-9A-Fa-f]{0,4}
float 0|([1-9][0-9]*)\.[0-9]*
id [_a-zA-Z][_0-9a-zA-Z]*
relop >|<|>=|<=|==|!=
%%
```

这部分写完之后就要进入规则部分了。

规则部分

rule的具体书写模式如下：

> pattern {action}

那不就很简单了，输入一个正则表达式，对该表达式进行行为描述，具体需要进行什么行为描述我们尚不能清楚，但是可以基本列出所有表达式，然后printf和return；
我这里简单看过后续，所以知道应当如何去写，但是在这里我们按下不表，先简单写点别的，比如按照以下格式去写：

```
";" {printf("SEMI");}
//具体代码这里不给出了嗬，主要是重复的机械工作。
```

那就开始吧，照抄活动！
抄完之后，就简单进行一下flex：

```
flex lexical.l
```

然后就生成一个文件了。

自定义部分

这是第三个部分，在这一部分我没有加入任何内容，pass即可。

联合调试

然后新建一个main.c文件，把指导书上的内容抄进去。

```
#include <stdio.h>
extern FILE* yyin;

int main(int argc,char** argv){
    if(argc>1){
      if(!(yyin=fopen(argv[1],"r"))){
      perror(argv[1]);
      return 1;
      }
    }
      while(yylex()!=0);
      return 0;
}
```

就是这段。
然后就可以进行接下来的操作了。

```
gcc main.c lex.yy.c -lfl -o scanner
./scanner test1.cmm
```

就会输出这样的结果了

```
karlyn@ubuntu:~/Desktop/compile$ ./scanner test1.cmm
TYPE
ID
LP
RP
LC
TYPE
ID
ASSIGNOP
INT
SEMI
TYPE
ID
ASSIGNOP
Error type A at Line 4: Mysterious characters "~".
ID
SEMI
RC
```


这样就完成了任务的第一步，也就是词法实别！
接下来就是语法识别了，有点难唔。

语法分析

到了这一段就需要开始不断调整前面的代码，来跟着实验指导书一步一步走。
首先是编写一个syntax.y的源程序，这里我就开始摸不着头脑了
因为它涉及到一个公共函数yylet()。这部分先不管吧，我们直接开始根据下面的要求编写源程序。
源程序依然是被分为三个部分
首先我们开始的是第一个部分：

定义部分

首先在我们需要明确一点，就是语法树最终是用树来存储以及输出的，也就是说我们为了实验的目的print需要将整个语法树存放到一个多叉树里面，所以无论如何实现这样一个syntax.y程序，我们总是要实现一个多叉树的。所以这时候我们先定义一个多叉树吧。
这个多叉树写在哪里呢？我不清楚，所以随便试一试，那就挑一个C语言文档来写吧。
所以就简单写了点GMT，下面是头文件。

```c
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX 10
extern int yylineno;
extern char *yytext;
extern int yylex(void);


struct GMT{
    int line;
    char *name;
    int n; //num of children
    union{
        char type[MAX];
        int i;
        int f;
    };
    struct GMT* child[MAX]; //children
};
struct GMT* newLeaf(char* s,int yyline);
struct GMT *newNode(char *s,int yyline,int num,struct GMT* arr[]);
struct GMT *nullNode(char *s,int num);
void printTree(struct GMT* r,int layer);
```

然后就进入到syntax.y的书写
回归到定义部分，是有一些定义的
比如token的定义、type的定义
以及优先级的定义
这里都是直接照着实验指导书敲就行了

```
%union{
struct GMT* tree;
}

/*Basic variable*//*基本变量*/
%token <tree> INT FLOAT ID
/*Sentence ending symbol*//*句尾符号*/
%token <tree> SEMI COMMA ASSIGNOP RELOP DOT NOT
/*Operation symbol*//*运算符号*/
%token <tree> PLUS MINUS STAR DIV
/*brackets*//*括号*/
%token <tree> LP RP LB RB LC RC
/*Reserved word*//*保留字*/
%token <tree> TYPE STRUCT RETURN IF ELSE WHILE AND OR
/*non-terminal*//*非终结节点*/
%type <tree> Program ExtDefList ExtDef ExtDecList Specifier StructSpecifier OptTag Tag VarDec FunDec VarList Par
amDec CompSt StmtList Stmt DefList Def DecList Dec Exp Args

/*First or Last*/
%right ASSIGNOP
%left OR
%left AND
%left RELOP
%left PLUS MINUS
%left STAR DIV
%right NOT
%left LP COMMA RP LB RB DOT
%nonassoc ELSE
```

规则部分

之后就要进入到规则模块了，这里比较难的是放error，简单来说就是先通过实验指导书后给的语法规则来构建树，打印就是基于这时候所构建的树。error存在的作用就是为了错误恢复，继续执行语法分析。
这里我的error实在放不明白，就不丢上去献丑了。
或许我之后能放明白在这里补上。
同时我们需要修改原来的lexical.l文件，引用一下syntax.tab.h
才可以实现两者yylex()共用，实现值的传递。
还是举一点简单的例子，

```
";" {yylval.tree=newLeaf("SEMI",yylineno); return SEMI;}
"," {yylval.tree=newLeaf("COMMA",yylineno); return COMMA;}
"=" {yylval.tree=newLeaf("ASSIGNOP",yylineno); return ASSIGNOP;}
```

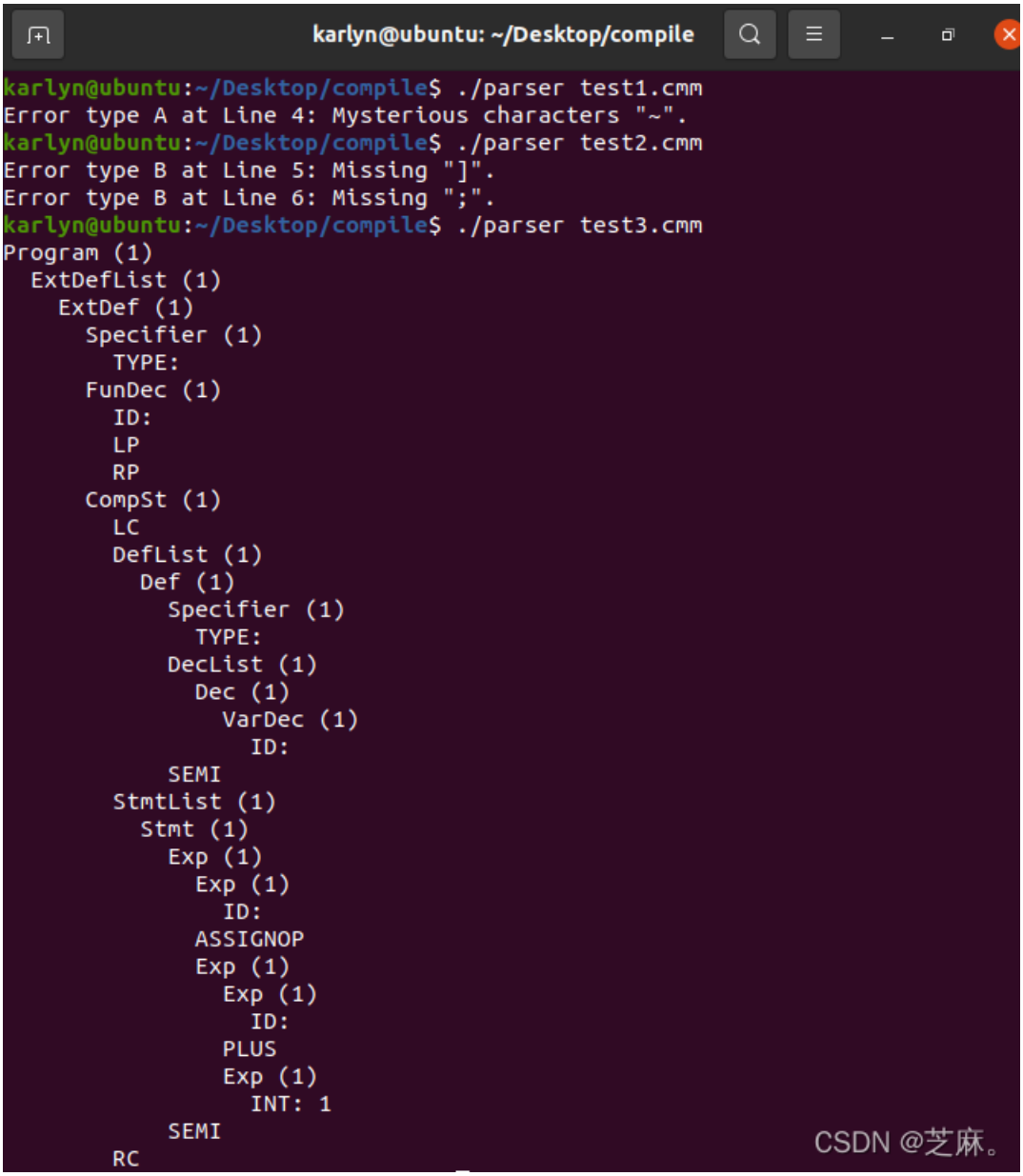
就如同这样修改一下规则。然后按照实验指导书的要求：

```
gcc main.c syntax.tab.c -lfl -o parser
```

就可以实现语法分析器parser了。

结果

简单展示一下运行结果吧：



唔写完了，开始摸鱼。