编译原理实验一报告

学号	1190202401
姓名	陈豪

一、实验环境

Ubuntu20.04TLS

gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0

flex 2.6.4

bison (GNU Bison) 3.5.1

二、程序实现的功能

本程序实现了对C--语言的词法和语法分析，同时选做了1.1，1.2和1.3

必做内容

为了能够在最终打印出语法树信息，因此在实验中的node.h文件如下定义了结构体node。

```
typedef struct node{
    uint32_t lineno;//行号
    nodeType type;//类型
    char * name;//名字
    char * value;//值
    struct node* child;//儿子节点
    struct node* brother;//兄弟节点
}Node;
typedef Node* pNode;
```

并定义相关处理函数头如下，同时它们的具体实现在node.c中已经编写了详细的注释了。

```
pNode newTokenNode(uint32_t lineno,nodeType type,
    const char *name,const char *value);
pNode newSyntaxNode(uint32_t lineno,nodeType type,
    const char *name,int argc,...);
void printSyntaxTree(pNode currentNode,int height);
void freeNode(pNode currentNode);
```

由于这些函数将会在词法分析和语法分析中反复被调用，因此使用C语言的inline方式以空间换时间来加快程序执行。

同时因为语法产生式的变量不是固定的，所以使用c语言变长参数的方式，定义创建语法节点的函数newSyntaxNode。

在lexical.l中新建词法节点的例子如下：

```
{INT} { yylval.node = newTokenNode(yylineno,INT_TYPE, "INT", yytext); return
INT;}
```

在syntax.y中新建语法树节点的例子，其中@$.first_line获取整个产生式的第一行位置：

```
Program:            ExtDefList                                  { $$ =
newSyntaxNode(@$.first_line, NON_TERMINAL, "Program", 1, $1); root = $$; }
    ;
```

因为所有语法节点使用统一类型，因此需要在syntax.y中如下声明

```
%union{
    pNode node;
}
```

选做内容c

选做1.1

如下编写了正则表达式

```
INT (0[0-7]*)|([1-9]+[0-9]*)|(0[xX][0-9a-fA-F]+)
```

可以看出这里将不同进制的数都识别到了一起，因此在建立节点的时候将会把值都统一转换成10进制数进行存取。转换函数头如下，其定义在util.h中并在util.c中具体实现

```
int convertHexToDec(const char *hexStr);
int convertOctToDec(const char *octStr);
```

同时，对于错误的八进制值和十六进制值将被下面的正则表达式识别。

```
0[0-9]+ {...}
0[xX][0-9a-zA-Z]+ {...}
```

选做1.2

按照附录A如下编写了浮点数的正则表达式

```
FLOAT {digit}+"."{digit}+|{digit}*"."{digit}+[eE][+-]?{digit}+|{digit}+"."
{digit}*[eE][+-]?{digit}+
```

同时针对错误的浮点数在lexer.l中编写如下

```
{digit}+{ID}  { ... /*非浮点数但是以数字开头且其中有e的*/}
"."{digit}+ { ... /*有基数且基数在.后，但没有指数和指数符号*/}
{digit}+"." { ... /*有基数且基数在.前，但没有指数和指数符号*/}
{digit}*"."{digit}+[eE] { ... /*有基数且基数在.后，指数符号，但没有指数*/}
{digit}+"."{digit}*[eE] { ... /*有基数且基数在.前，指数符号，但没有指数*/}
{digit}+[eE][+-]?{digit}* { ... /*有基数，指数符号，指数但基数中没有.*/}
"."[eE][+-]?{digit}+ { ... /*无基数，有指数符号，指数*/}
```

对于选做1.1和选做1.2的错误类型，我选择在词法分析时获取并且值都设置为0。同时也可以通过建立一个新的错误标记token来表示这些错误的词法单元。

选做1.3

针对注释编写了下面的正则表达式,对应的动作是匹配到了就不执行语句。

```
linecomment "//".*
multilinecomment "/*"[/]*([^*/][/]*|[*][^*/]*)*"*/"
```

三、程序编译方式

本次程序使用了makefile进行编译。编译命令如下

```
~/compilerLab/Lab1/code$ make
```

编译后，测试样例文的命令件如下

```
~/compilerLab/Lab1/code$ make test
```

四、参考资料

- [1] 编译原理实践与指导教程
- [2] flex and bison中文版
- [3] github