编译原理实验一　词法分析语语法分析

111220174 张赫
mattzhang9@gmail.com

词法分析

1. 声明部分：

   此部分定义了一些用于词法分析的正则表达式。包括识别 '\r' 和需要做的八进制和十六进制字符。对于八进制和十六进制的错误并没有在词法处进行处理。

2. 转换规则部分：

```
{id}        {initTree("ID",yytext);return ID;}
```

   大部分的规则都符合如下这个格式：
   可以看到，左边即相应的正则表达式；右边有两个语句，第一个是按照匹配到的文本建立语法树的节点，第二个是返回词法单元。在这个部分要注意把return, else, while, if等规则放在 id 前面，否则会先匹配到 id 而不会匹配到保留字。

3. 相关函数：

   首先介绍一下语法树节点的结构：每个节点包含terminal, type, text, lineno 以及children和sibling。terminal记录是否是终结符，type记录语法单元，text记录词素，lineno记录行号，children和sibling用于语法树构建。
   需要注意的是，text虽然被定义为char[]型，但不光可以记录char[]型，还可以更改指针类型记录int或float。
   然后是 initTree 函数。该函数该函数使用

```
yylval.iNode = (struct ASTNode *)malloc(sizeof(struct ASTNode));
```

   直接申请语法树节点并将指针通过yylval 返回给语法分析文件。同时为了简化代码，将相同的赋值部分放入函数initCommonTree，具体的赋值函数只是调用此函数后对text部分进行修改。对于八进制和十六进制，直接手写了函数将其转化为十进制存储，如果不符合规范则会被拆成两部分或多部分匹配，然后在语法分析处报错。然后对语法树节点进行赋值。对于INT型，使用

```
int temp = atoi(text);
memcpy(yylval.iNode->text, &temp, sizeof(int));
```

   来将int存储到text字段，从而统一化存储方式，float同理。

语法分析

1. 声明部分：

   将token按优先级声明，并声明全局变量语法树根 root和错误统计error。将yylval的类型设为单一的语法树节点类型，这样就可以直接用在词法分析部分生成的所有语法树节点。

2. 转换规则部分：

本部分处理所有词素，对于正确可推导的语法用connTree构建语法树；对于错误的推导记录错误并尝试进行错误恢复。正常的部分和语法要求完全相同，错误恢复部分花了很长时间，最后终于至少能将样例中的所有错误恢复。比如对于a[5,3]，如果使用Exp : error RB 恢复是不对的，这样一来会在改行报两个错误。后来发现应该用EXP : EXP LB error RB才能将前面的Exp和LB吃掉。对于会单独成行的地方几乎都加了错误恢复，但对于一些奇怪的错误或者少终结符的错误仍然无法恢复。

3. 相关函数：

最重要的函数是connTree函数。该函数采用可变参数数量的方式简化了组建语法树的过程，可以用一条语句就组建语法树而不用在每条规则都因为树节点的不同数量而写多条语句。具体方式采用va_list进行参数列表构造，使用

```
va_list ap;
va_start(ap, num);
var = va_arg(ap, struct ASTNode*);
...
va_end(ap);
```

来访问所有参数，从而使用sibling和children来构建语法树。

用display()函数深度优先遍历先序打印节点。当碰到int型的时候，使用

```
*(int*)(t->text)
```

将原来用 text存储的数值打印出来，其余按照要求打印。

主程序

main.c与实验指导攻略要求的类似，用error是否等于0来判断是否有错，如果没有错则输出语法树。

使用方法

在source文件夹中有README.md来说明使用方法。使用 make parser来编译出parser，make clean清理。对于实验要求中给出的六个例子，已经放在test文件夹中，只要在source文件中执行 bash test.sh即可。