第二部分

毋哲

语法分析程序

任务：

语法分析程序的主要任务是进行语法分析，对语法错误进行检查，以便于进行语义分析的时候使用的是符合文法语法的程序。

如果不是 LL(1)文法怎么办？

我们都知道，LL(1)文法就是只需要读一个单词，便可根据文法知道这个单词属于哪一个语法成分。如果我们的文法是 LL(1)文法，对于我们语法分析程序的编写来说会变得非常简单。但是，由于我们在文法修改的时候并没有将文法改成 LL(1)文法（详情请见文法修改那一节），因此，我们就不能够通过只读一个单词，来判断这个单词属于哪一个语法成分。在我们进行文法修改的时候，我们希望通过向前读 3 个单词，来判断语法成分。这种想法在文法修改时，感觉是非常正确的一种处理方法（相比于把文法改成 LL(1)而言）。但是，当我们进行语法分析的时候，我们会发现，向前读三个单词来分析会使语法分析程序变得复杂。具体的编程思想其实并不难，难的是对细节的考虑以及细心，这里我给出一个大的框架供大家参考。

```
读入一个单词
if (单词是'const')
{
    //常量定义
    for (i=0;i<3;i++)
    {
        读三个词，存入数组 WordList
    }
    if (WordList[0]是'int','char','float')
    {
        if (WordList[1]是标识符)
        {
            if (WordList 是'=')
            {
                进入常量定义子函数
            }
            else
            {
                出错
            }
        }
        else
        {
            出错
        }
```

```
}
else
{
    出错
}
do{
    for (i=0;i<3;i++)
    {
        读三个词，存入数组 WordList
    }
    if (step==0//现在还处于分析常量的阶段)
    {
        if (WordList[0]是'int','char','float')
        {
            if (WordList[1]是标识符)
            {
                if (WordList 是'=')
                {
                    进入常量定义子函数
                }
                else
                {
                    if (WordList[2]是';'或',')
                    {
                        step=1;//进入变量分析阶段
                        isThereWord=1;//现在已经有三个词读出
                    }
                    else if (WordList[2]是'(')
                    {
                        isThereWord=1;//现在已经有三个单词读出
                        step=2;//进入函数分析阶段
                    }
                    else
                        出错
                }
            }
            else
            {
                出错
            }
        }
        else
        {
            if (WordList[0]是'void')
```

```
                    {
                        isThereWord=1;//现在已经有三个单词读出
                        step=2;//进入函数分析阶段
                    }
                    else
                        出错
                }
            }
        }while (现在 step==0 且还有单词未读出);
    }
    do{
        //常量定义->step+1->变量定义->step+1->函数定义->step+1->主函数定义
        if (step==0//如果 step==0，说明前面没有 const 声明常量，需要读三个单词)
        {
            for (i=1;i<3;i++)
            {
                读三个单词存入 WordList;
            }
            step=1;//现在是变量分析阶段
        }
        if (step==0//现在还处于分析常量的阶段)
        {
            if (WordList[0]是'int','char','float')
            {
                if (WordList[1]是标识符)
                {
                    if (WordList 是',')
                    {
                        进入变量定义子函数
                    }
                    else if (WordList[2]是';')
                    {
                        符号表操作
                    }
                    else if (WordList[2]是‘(’)
                    {
                        step=2;//进入函数分析阶段
                    }
                    else
                    {
                        出错
                    }
                }
                else
```

```
            {
                出错
            }
        }
        else
        {
            if (WordList[0]是'void')
                step=2;//进入函数分析阶段
            else
                出错
        }
    }
    if (step==2)//函数定义
    {
        if (WordList[0]是'void','int','char','float')
        {
            if (WordList[1]是标识符)
            {
                if (WordList[2]是'(')
                {
                    参数表声明函数
                    if (现在的单词不是')')
                        出错
                    else
                    {
                        读一个单词;
                        if (这个单词不是'{')
                            出错
                        else
                        {
                            读一个单词;
                            复合语句分析函数
                            if (现在的单词不是'}')
                                出错
                        }
                    }
                }
                else
                {
                    出错
                }
            }
            else
            {
```

```
                    if (WordList[1]是 main)
                        step=3;//进入主函数分析阶段
                    else
                        出错
                }
            }
            else
            {
                出错
            }
        }
        if (step==3)//主函数定义
        {
            if (WordList[0]是 void)
            {
                if (WordList[1]是 main)
                {
                    if (WordList[2]是'(')
                    {
                        读一个单词;
                        if (现在的单词是')')
                        {
                            读一个单词
                            if (现在的单词是'{')
                            {
                                复合语句子程序
                                if (现在的单词不是'}')
                                    SendError(line,16,FROMDOG);
                            }
                            else
                            {
                                出错
                            }
                        }
                        else
                        {
                            出错
                        }
                    }
                    else
                    {
                        出错
                    }
                }
```

```
                else
                {
                    出错
                }
            }
            else
            {
                出错
            }
        }
    }while ((WordList[1].type 不是 main) &&  还有单词可以读出);
```

语法错误分类的粒度大小

在我们最开始往语法分析程序中加入错误处理的代码的时候，都倾向于将同一种现象的错误按一种错误来处理。比如说 if 语句的条件判断少了')'（如 if (a<b a=b;），和 for 语句的条件判断少了')'（如 for (i=1;i<10;i=i+1 a=b;）。当我们遇到这种情况时，很自然的想法是将这两种缺少了')'归为同一种错误。但是这种想法是否是合理的呢？我们不妨看下面的程序。

```
if (a<b
{
    a=a+1;
    b=b-1;
}
FindType(;
```

在上面的程序中，出现了两个少了')'的错误，一个是 if 的条件判断，另一个是在调用 FindType 这个函数时。通常我们希望错误处理程序可以将出错的语句跳过继续编译后面的部分。如果遇到上述的问题，我们会发现，同一种错误，处理的方法却是不同的。在 if 语句中，我们发现没有')'，则希望从遇到的'{'开始重新编译。而 FindType 语句中，我们希望在遇到';'之后重新编译。于是，同一种错误，两种截然不同的处理方式，看到这里，我们便可以发现将同一种错误合并所带来的不便利性。所以为了大家在写错误处理程序更方便，更容易，我建议应该将所有不同类型的错误都分开，即使这些错误的现象都是一样的。这样在错误处理程序中，我们才不会因为得判断这是哪一种缺少')'，而加大程序编写的难度。

错误处理程序

任务：

错误处理程序的主要任务是处理我们在语法分析时遇到的各种错误。主要的错误方法有：跳过出错代码使得编译器能继续编译以及改正错误。

错误处理程序的分类

编译中，错误主要是出现在编译器的前端，错误的类型有词法错误，语法错误和语义错误。

词法错误就是在词法分析的时候出现了不符合文法的字符，即不符合任何一种单词的组合规则。

语法错误就是在语法分析的时候出现的不符合文法的错误，如语句末尾缺少分号等。

语义错误就是在进行语义分析时，出现的不符合程序语义的错误。如定义变量重名，使

用的变量没有声明等等。

每当出现一个错误的时候，我们都要输出相应的错误信息以提示程序员对程序进行修改，除此之外，错误处理程序还需要对错误进行一定的处理，使得程序可以继续编译下去。如果遇见一个错误便停止编译，对于那些一个程序中会出好多错误的程序员来说，编译成功一个程序会非常麻烦而且费时。

不同类型的错误的处理

说到不同类型的错误处理，在输出错误信息之外，错误处理程序需要做的只是对语法错误进行处理。这是为什么呢？出现语义错误的语句是完全符合文法的，所以我们并不需要对程序本身处理什么，而词法错误的处理是包含在词法分析程序中的（想想状态机），所以我们错误处理程序所要做的只是处理语法错误。

处理语法错误的方法每个人都可能有一套，主要区分度就是出错时跳过的单词粒度的大小。比如说定义了一个函数

```
fx(int a)
int b;
b=a*a;
return b;
}
```

这个 fx 函数在函数体部分缺少了’{’。对于粒度大的错误处理方法，我们可以直接跳过这个函数不编译（直接找’}’），对于粒度小的错误处理方法，我们可以在发现少了’{’之后继续编译。粒度大的比较好写，而且情况比较少，但是能查出的错误有限。粒度小的相对比较复杂，情况也比较多（这就需要注意我们之前提过的，在语法分析程序中，对同一种现象的不同类型错误的分类），但是一次编译可以查出更多的语法错误。

遇到错误不处理行不行？

遇到错误不处理，继续编译，这样不是会做到粒度最小的错误处理吗？

这样当然也可以，但是同一个错误可能导致非常多的其他错误的出现，反而会增加程序员修改错误的难度。

比如说下面这个程序片段：

```
int a~,b,c;
```

这里出现了一个非法字符’~’，这时程序继续编译，会把’,’当做一种新的语法成分的开始，但是没有哪种语法成分是以’,’开始的，所以又会报错，接下来是’b’，是一个标识符，再接下来又是’,’，又会出现语法错误，以此类推，直到编译完’;’之后，编译器才能重回正轨。在这里我们发现，如果不处理，额外的附加错误可能会很多，最坏的情况是每次编译报告的错误信息，只有第一条是正确的，其他都是错误的。所以我们可以看到，错误的处理还是很重要的。