**编译原理实验报告**

日期：2021年11月12日

**一、实验题目**

对下述文法和单词表定义的语言设计编制一个语法分析器。

1. 单词符号及种别表

|  |  |  |
| --- | --- | --- |
| 单词符号 | 种别编码 | 单词值 |
| main | 1 |  |
| int | 2 |  |
| float | 3 |  |
| double | 4 |  |
| char | 5 |  |
| if | 6 |  |
| else | 7 |  |
| do | 8 |  |
| while | 9 |  |
| l(l|d)\* | 10 | 内部字符串 |
| ( +|-|ε ) dd\*(.dd\* | ε)( e ( +|-|ε ) dd\*|ε) | 20 | 二进制数值表示 |
| = | 21 |  |
| + | 22 |  |
| - | 23 |  |
| \* | 24 |  |
| / | 25 |  |
| ( | 26 |  |
| ) | 27 |  |
| { | 28 |  |
| } | 29 |  |
| , | 30 |  |
| ; | 31 |  |
| > | 32 |  |
| >= | 33 |  |
| < | 34 |  |
| <= | 35 |  |
| == | 36 |  |
| != | 37 |  |

1. 语法结构定义

<表达式> ::= <项>{ +<项>|-<项>}

<项> ::= <因子>{\*<因子>|/<因子>}

<因子> ::=ID|num|(<表达式>)

num::=( +|-|ε ) 数字数字\*(.数字数字\* | ε)( e ( +|-|ε ) 数字数字\*|ε)

ID::=字母(字母|数字)\*

字母::=a|b|c…|z|A|B|C…|Z

数字::=0|1|2…|9

**二、实验目的**

1、通过该实验,熟练应用编译原理的基本理论和方法

2、学会用C/C++高级程序设计语言设计一个语法分析器的技术

3、加深对编译原理的分析理论的理解，培养动手实践能力

**三、实验要求**

词法分析程序需具备语法分析的功能：

处理用户提交的符合上述文法的源代码序列，进行语法分析，并给出语法是否正确的结论。

例如：控制台输入d+-11.7e-17，经过语法分析后输出如下序列：

success!

例如：控制台输入(123.456+-456.789e-120)\*m2+(a++456)\*c123，经过语法分析后输出如下序列：

success!

例如：控制台输入(123.456+-456.789e-120)\*m2+(a++456)\*-c123，经过语法分析后输出如下序列：

error!

**四、实验步骤**

（包括基本设计思路、流程框图、算法设计、函数相关说明、输入与输出以及程序运行结果）

基本设计思路：采用递归下降子程序的方法来进行语法分析，为每一个非终结符创建一个子过程(函数)，每个子过程处理一个非终结符对应的所有产生，调用开始符号对应的子程序开始语法分析。

具体来说这里一共有5个子过程，分别处理表达式、项、因子、num和id，在表达式中先调用一个项的处理子过程，然后向后看一个字符，如果是+或-，那么再调用一次项的处理子过程；项的处理过程与表达式相似，无非换成了因子和\*/；在因子处理子过程中，通过向后看一个字符，如果是字母则调用id子过程，否则如果是数字则调用num子过程，如果是(则调用表达式子过程，这里注意处理完表达式后，要匹配一个)，所有的匹配过程有match函数完成，匹配成功后会预读下一个字符；num和id的处理类似，只需按照产生式的形式，进行不断地匹配和判断即可。

函数相关说明：

expr(): 分析表达式 term():分析项 factor():分析因子 id():分析标识符

num():分析整数和浮点数 match(int):匹配指定的单个字符并预读下一个字符

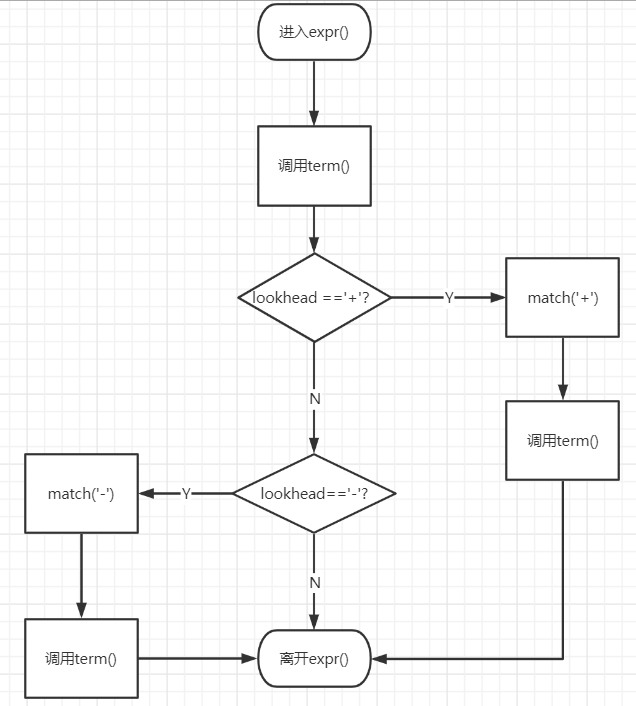
matchAlpha():匹配大小写字母并预读下一个字符

matchDigit():匹配0-9并预读下一个字符

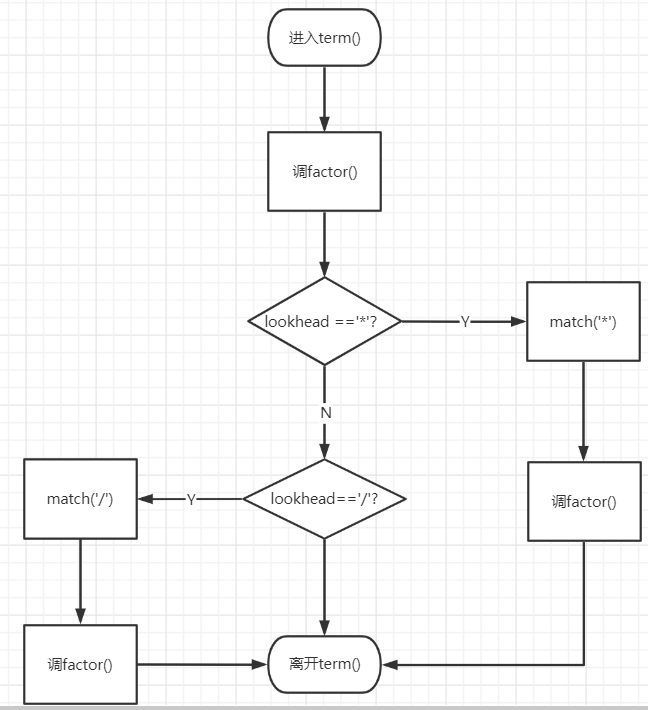
success():该输入被接受后的处理 error():出错处理 getNext():读取下一个字符

主要流程图：

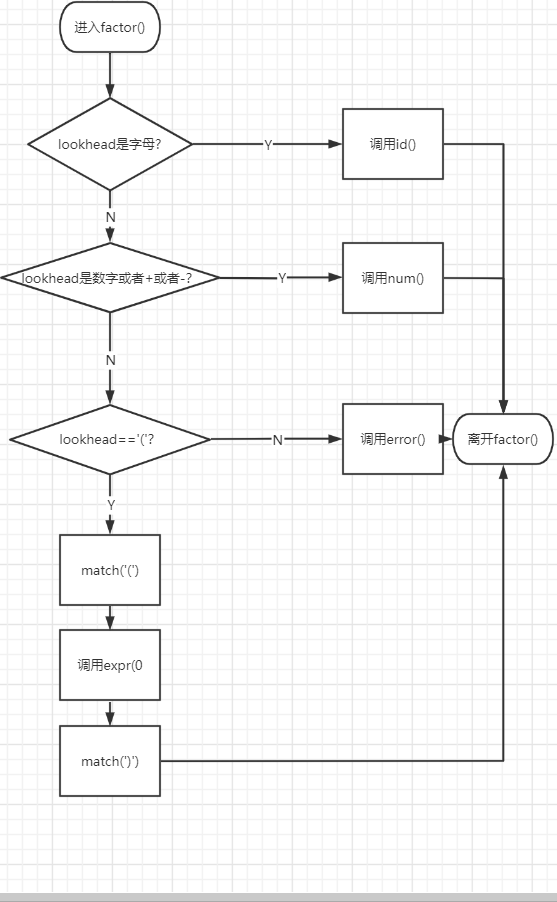
expr流程图：



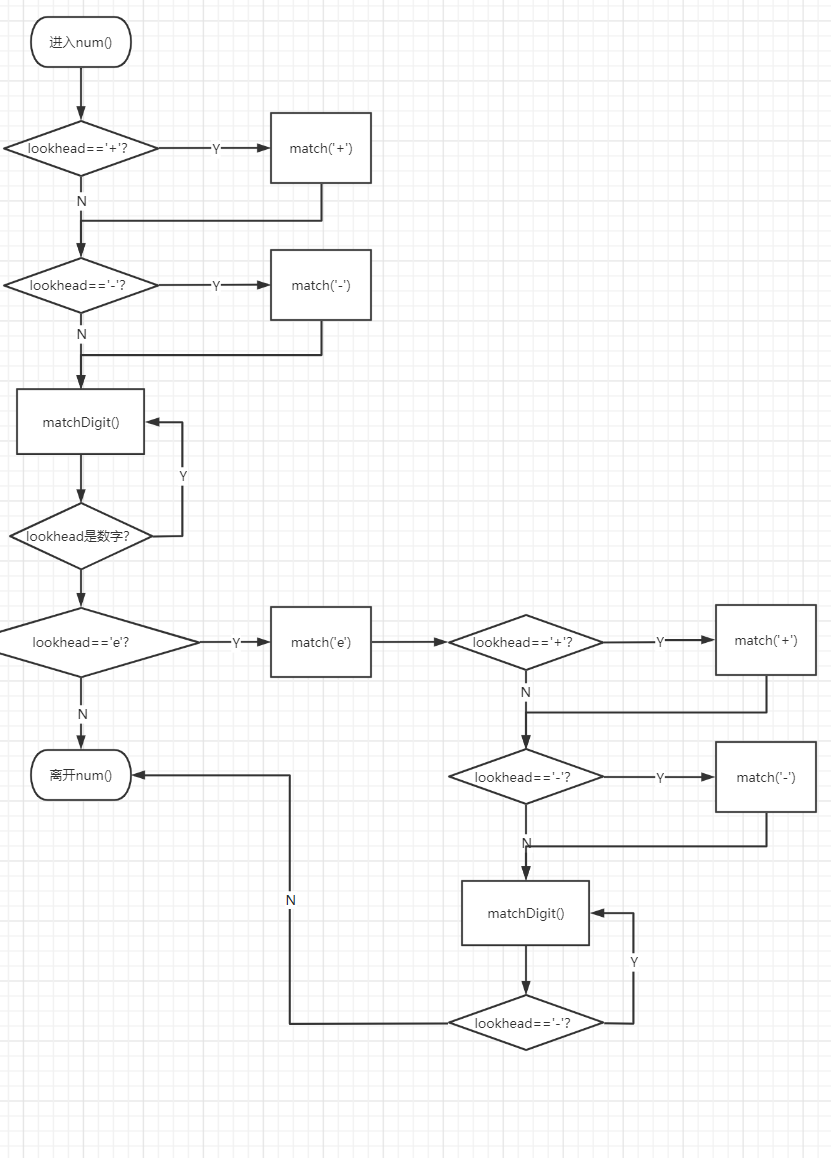
term流程图:



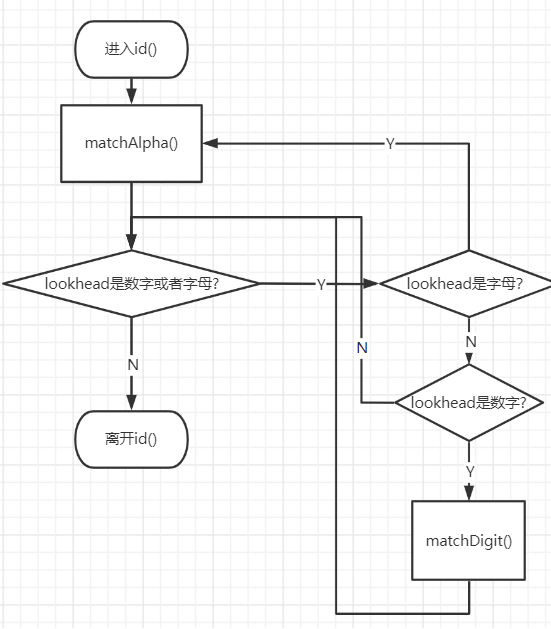
factor流程图：



num流程图

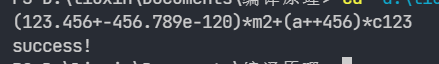


id流程图：



程序运行结果：







**五、实验心得体会**

通过本次实验编写一个基于递归下降的语法分析器，对与语法分析过程有了进一步的认识，递归下降的方法是一种自顶向下的分析方法，从开始符号对应的子过程开始，按照产生式依次调用其他非终结符符号的子程序进行处理，如果任何一步发现错误则转到出错处理，直到读取到最后一个字符则表示语法分析成功，该输入被该文发接受，这种递归下降的分析方法需要每次向后看一个字符来决定要使用那个产生式。如果文法含有左公共因子，需要先提取左公共因子，然后用对新的文法编写递归子程序，编写递归子程序思路比较简单，为每一个非终结符创建一个处理子程序，通过产生式来编写每个子程序之间的调用逻辑，并不麻烦。

**六、源程序清单（代码）**

#include <iostream>

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <string>

using namespace std;

class Parser

{

public:

int lookhead; // 下一个要处理的字符

Parser()

{

getNext();

}

// 处理表达式

void expr()

{

term();

if (lookhead == '+')

{

match('+');

term();

}

else if (lookhead == '-')

{

match('-');

term();

}

}

// 处理项

void term()

{

factor();

if (lookhead == '\*')

{

match('\*');

factor();

}

else if (lookhead == '/')

{

match('/');

factor();

}

}

// 处理因子

void factor()

{

if (isalpha(lookhead))

id();

else if (isdigit(lookhead) ||

lookhead == '+' ||

lookhead == '-')

num();

else if (lookhead == '(')

{

match('(');

expr();

match(')');

}

else

error();

}

// 处理标识符

void id()

{

matchAlpha();

while (isalpha(lookhead) || isdigit(lookhead))

{

if (isalpha(lookhead))

matchAlpha();

else

matchDigit();

}

}

// 处理数字

void num()

{

if (lookhead == '+')

match('+');

else if (lookhead == '-')

match('-');

matchDigit();

while (isdigit(lookhead))

matchDigit();

if (lookhead == '.')

{

match('.');

matchDigit();

while (isdigit(lookhead))

matchDigit();

}

if (lookhead == 'e')

{

match('e');

if (lookhead == '+')

match('+');

else if (lookhead == '-')

match('-');

matchDigit();

while (isdigit(lookhead))

matchDigit();

}

}

// 匹配单个字符，成功则读取下一个字符

void match(int t)

{

if (lookhead == t)

getNext();

else

error();

}

// 匹配字母

void matchAlpha()

{

if (isalpha(lookhead))

getNext();

else

error();

}

// 匹配数字

void matchDigit()

{

if (isdigit(lookhead))

getNext();

else

error();

}

// 读取下一个字符

void getNext()

{

char tmp;

if (!(cin >> tmp))

success();

lookhead = tmp;

cout << tmp;

}

// 出错处理

void error()

{

puts("\nerror!");

exit(1);

}

// 成功后的响应

void success()

{

printf("\nsuccess!");

exit(0);

}

};

int main()

{

freopen("input.txt", "r", stdin);

Parser a = Parser();

a.expr();

system("pause");

return 0;

}