**实验一：词法分析程序设计**

1. **实验目的**

1.掌握程序语言的词法结构。

2.掌握状态转换图的使用。

2.掌握设计词法分析程序的一般方法。

4.学会熟练调式程序。

5.用状态转换图描述程序词法结构的一般方法，并在此基础上完成简版C语言子集词法分析程序的设计和调试运行。

1. **实验内容**

（1）简单词法分析的状态转换图



（2）状态转换图的实现

状态转换图非常容易用程序实现,基本思想是扫描到单词符号第一个字符的种类，拼写出相应单词符号。

设置3个变量

char token [ ]：存放构成单词符号的字符串。

int Sum：整型常数。

int syn：单词种别码。

整体框架为一个if if else else，分别对应标识符，常数，符号三种情况。

先将关键字表示为一个全局字符串数组。如果扫描标识符，则采用strcmp函数让该字符串与定义的全局关键字数组进行对比，如果能查到匹配单词，则该单词为关键字，否则为一般标识符。如果扫描到常数，直接让syn等于单词符号对应的种别码表中常数对应的种别码。如果扫描到其他字符，通过switch case结构，让syn等于每一个符号所对应的种别码值。

1. **实验代码**

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

char prog[80];

char token[80];//构成单词符号的字符串

char ch;

int syn = 0;//单词种别码

int sum = 0;//整型常量

int n, m, p = 0;

char rwtab[6][10] = { "begin","if","then","while","do","end" };

void scaner()

{

/\* 共分为三大块，分别是标示符、数字、符号，对应下面的 if else if 和 else \*/

sum = 0;

for (m = 0; m < 8; m++)

{

token[m++] = NULL;

}

ch = prog[p++];

m = 0;

while ((ch== ' ') || (ch == '\n'))

{

ch = prog[p++];

}

//标识符或者变量名

if (((ch <= 'z') && (ch >= 'a')) || ((ch <= 'Z') && (ch >= 'A')))

{

while (((ch <= 'z') && (ch >= 'a')) || ((ch <= 'Z') && (ch >= 'A')) || ((ch >= '0') && (ch <= '9')))

{

token[m++] = ch;

ch = prog[p++];

}

p--;

syn = 10;

for (n = 0; n < 6; n++)//将识别出来的字符和已定义的关键字作比较

{

if (strcmp(token, rwtab[n]) == 0)

{

syn = n + 1;

break;

}

}

}

//常数

else if((ch>='0')&&(ch<='9'))

{

while ((ch >= '0') && (ch <= '9'))

{

sum = sum \* 10 + ch - '0';

ch = prog[p++];

}

p--;

syn = 11;

}

//符号

else switch (ch)

{

case'<':

token[m++] = ch;

ch = prog[p++];

if (ch == '=')

{

syn = 22;

token[m++] = ch;

}

else

{

syn = 20;

p--;

}

break;

case'>':

token[m++] = ch;

ch = prog[p++];

if (ch == '=')

{

syn = 24;

token[m++] = ch;

}

else

{

syn = 23;

p--;

}

break;

case'+':

token[m++] = ch;

ch = prog[p++];

if (ch == '+')

{

syn = 17;

token[m++] = ch;

}

else

{

syn = 13;

p--;

}

break;

case'-':

token[m++] = ch;

ch = prog[p++];

if (ch == '-')

{

syn = 29;

token[m++] = ch;

}

else

{

syn = 14;

p--;

}

break;

case'!':

token[m++] = ch;

ch = prog[p++];

if (ch == '=')

{

syn = 21;

token[m++] = ch;

}

else

{

syn = 31;

p--;

}

break;

case'=':

token[m++] = ch;

ch = prog[p++];

if (ch == '=')

{

syn = 25;

token[m++] = ch;

}

else

{

syn = 18;

p--;

}

break;

case'\*':

syn = 15;

token[m++] = ch;

break;

case'/':

syn = 16;

token[m++] = ch;

break;

case'(':

syn = 27;

token[m++] = ch;

break;

case')':

syn = 28;

token[m++] = ch;

break;

case'{':

syn = 5;

token[m++] = ch;

break;

case'}':

syn = 6;

token[m++] = ch;

break;

case';':

syn = 26;

token[m++] = ch;

break;

case'\"':

syn = 30;

token[m++] = ch;

break;

case'#':

syn = 0;

token[m++] = ch;

break;

case':':

syn = 17;

token[m++] = ch;

break;

default:

syn = -1;

break;

}

token[m++] = '\0';

}

int main()

{

p = 0;

printf("please put a string end with#:\n");

do

{

scanf("%c", &ch);

prog[p++] = ch;

} while (ch != '#');

p = 0;

do

{

scaner();

switch (syn)

{

case 11:

printf("(%-10d%5d)\n", sum, syn);

break;

case -1:

printf("worng");

getchar();

default:

printf("(%-10s%5d)\n", token, syn);

break;

}

} while (syn != 0);

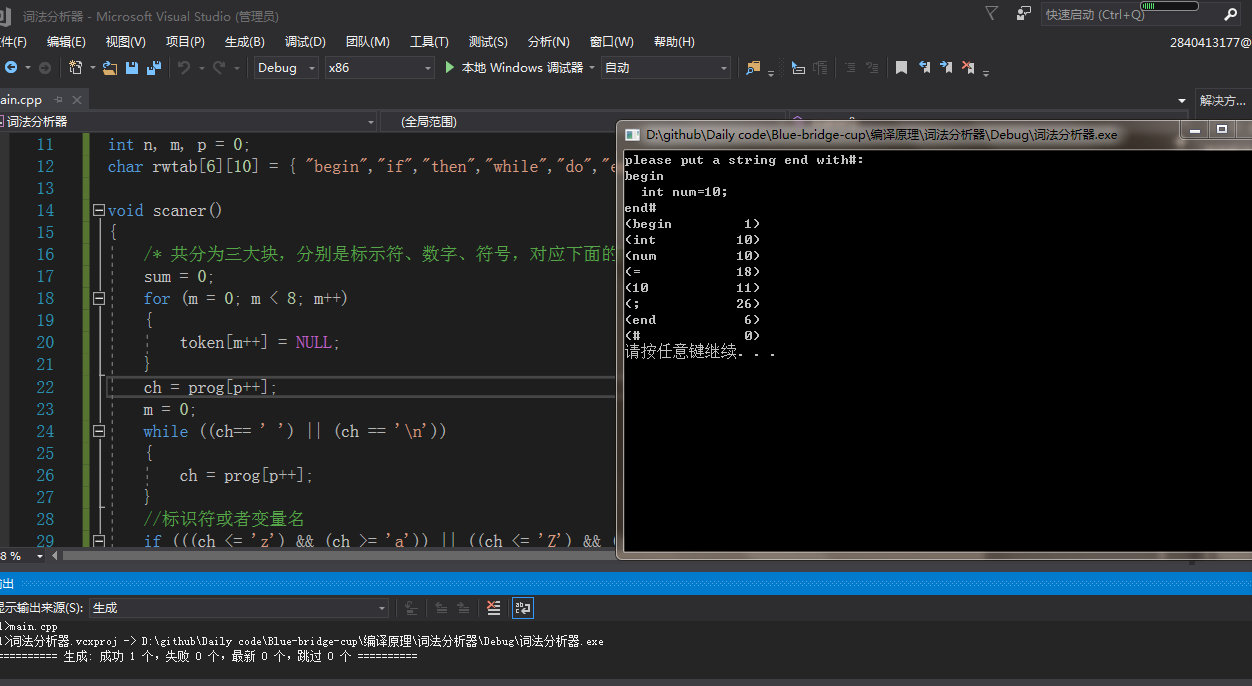
getchar();

system("pause");

return 0;

}

1. **实验结果**

****

1. **实验结果分析**

完成了词法分析器的功能：输入字符流，输出单词的内部表示。

通过本次实验，我加深了对词法分析原理的理解，掌握了词法分析器的构造和方法，学会了如何自己动手设计，编写并调制一个词法分析器，同时还更加巩固了对c语言的应用，更加培养了学习的兴趣。

**实验二：语法分析程序设计**

1. 实验目的

1.掌握语法分析方法。

2.掌握使用算符优先分析法。

3.完成语法分析程序的设计和实现。

4.程序能完成对指定语言的语法分析。

5.重点及难点：熟悉算符优先分析等语法分析方法，掌握语法分析的一般过程。在此基础上完成基于算数表达式的语法分析程序的设计和调试运行。

1. 实验内容

**一、 递归下降分析器**

在不含左递归和每个非终结符的所有候选终结首符集都两两不相交的条件下，我们就可能构造一个不带回溯的自上而下的分析程序，这个分析程序是由一组递归过程(或函数)组成的，每个过程(或函数)对应文法的一个非终结符。这样的一个分析程序称为递归下降分析器。

本次实验完成如下文法的递归下降分析器的设计。

文法G[E]为：

G[E]: E→E+T∣T

T→T\*F∣F

F→(E)∣i

经过消去直接左递归后得到文法G'[E]为：

G'[E]: E→TE'

   E'→+TE'∣ε

   T→FT'

   T'→\*FT∣ε

    F→(E) ∣i

该文法对应的递归下降分析器如下：

void match (token t)

{

if (lookahead == t)

lookahead = nexttoken;

else error ( );

}

void E( )

{

T( );

E'( );

}

void E'( )

{

if (lookahead == '+')

{

match ('+');

T( );

E'( );

}

}

void T( )

{

F( );

T'( );

}

void T'( )

{

if (lookahead =='\*')

{

match ('\*');

F( );

T'( );

}

}

void F( )

{

if (lookahead == 'i')

match ('i');

else if (lookahead =='(')

{

match ('(');

E ( );

if (lookahead ==')')

match (')');

else error ( );

}

else error ( );

}

**2. LL(1)分析法**

本次语法分析实验也可以选作LL（1）分析程序。

LL(1)分析法的基本思想是根据输入串的当前输入符号来惟一确定选用某条规则(产生式)来进行推导；当这个输入符号与推导的第一个符号相同时，再取输入串的下一个符号，继续确定下一个推导应选的规则；如此下去，直到推导出被分析的输入串为止。

一个LL(1)分析器由一张LL(1)分析表(也称预测分析表)、一个先进后出分析栈和一个控制程序(表驱动程序)组成，如下图所示。



图 LL(1)分析器

把第一个输入符号读入a；

do{

把栈顶符号弹出并放入x中；

if(x∈VT)

{

if(x＝＝a) 将下一输入符号读入a;

else error( );

}

else

if(M[x,a]＝“x→y1y2…yk”)

{

按逆序依次把yk、yk−1、…、y1压入栈中;

输出“x→y1y2…yk”;

}

else error( );

1. 实验代码
2. 递归下降分析法

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include<stdio.h>

#include<stdlib.h>

void E();

void T();

void E1();

void T1();

void F();

void i();

char s[100];

int j, k;

int main() {

printf("请输入一个语句，以#号结束语句\n");

while (1) {

k = 0;

j = 0;

scanf("%s", &s);

if (s[0] == '#')

return 0;

E();

if (k == 0) {

if (s[j] == '#')

printf("合法！\n");

}

printf("请输入一个语句，以#号结束语句\n");

}

system("pause");

return 1;

}

void match(char t)

{

if (s[j] == t)

{

j = j + 1;

}

else

{

printf("不合法");

}

}

void E()

{

if (k == 0)

{

T();

E1();

}

}

void E1()

{

if (k == 0)

{

if (s[j] == '+')

{

match('+');

T();

E1();

}

else if (s[j] != '#'&&s[j] != ')')

{

printf("不合法！\n");

k = 1;

}

}

}

void T()

{

if (k == 0)

{

F();

T1();

}

}

void T1()

{

if (k == 0)

{

if (s[j] == '\*')

{

match('\*');

F();

T1();

}

else if (s[j] != '#'&&s[j] != ')'&&s[j] != '+')

{

printf("不合法！\n");

k = 1;

}

}

}

void F()

{

if (k == 0)

{

if (s[j] == '(')

{

match('(');

E();

if (s[j] == ')')

match(')');

else if (s[j] == '#')

{

printf("不合法！\n括号不匹配\n");

++j;

k = 1;

}

}

else i();

}

}

void i() {

if (s[j] >= '0'&&s[j] <= '9')

++j;

else

{

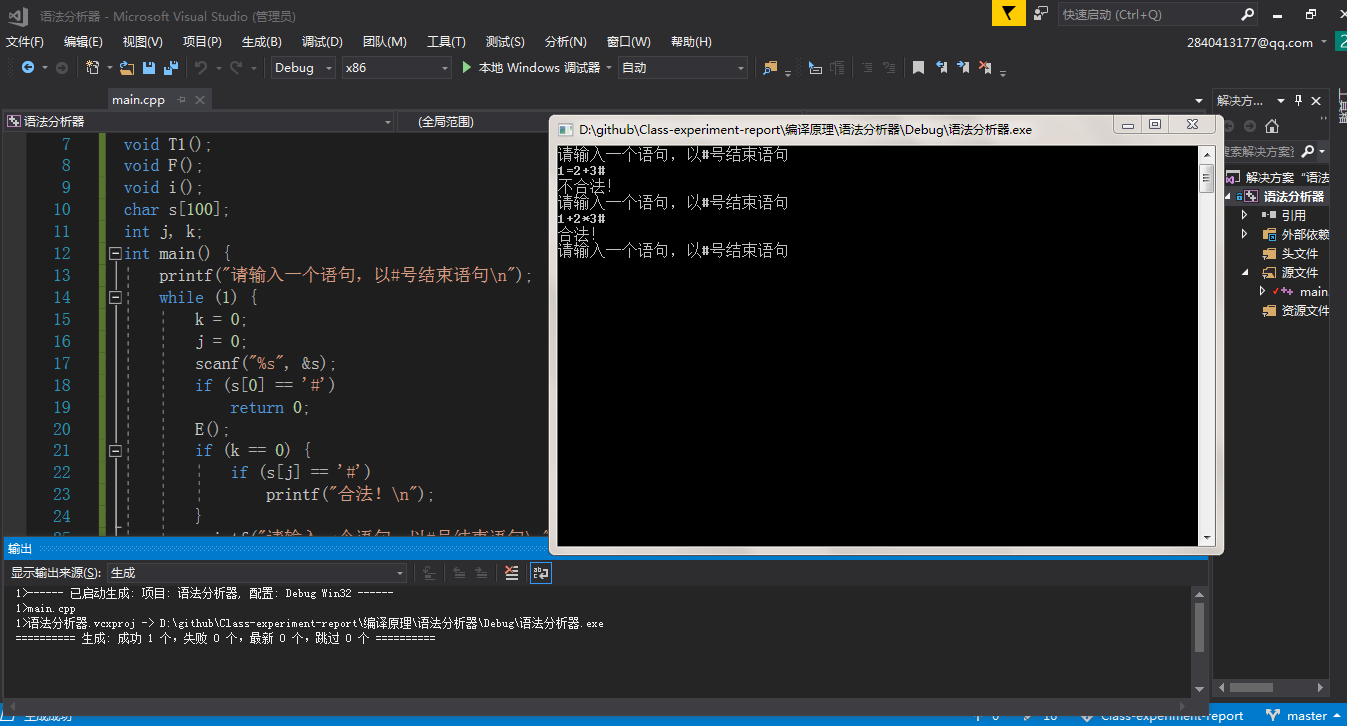
printf("不合法！\n");

k = 1;

}

}

1. 实验结果



1. 实验结果分析

完成语法分析的功能：输入算数表达式，输出表达式的语法分析结果

**实验三：中间代码生成程序设计**

1. 实验目的

1.掌握中间代码生成的基本方法。

2.掌握语法制导翻译模式。

3.完成算术表达式的中间代码生成程序。

4.掌握语法制导翻译模式的核心思想和工作原理，在此基础上完成基于算数表达式的中间代码生成程序的设计和调试运行。

1. 实验内容

算符优先分析法是一种简单且直观的自下而上分析方法，它特别适合于分析程序语言中的各类表达式，并且宜于手工实现。所谓算符优先分析，就是依照算术表达式的四则运算过程来进行语法分析，即这种分析方法要预先规定运算符(确切地说是终结符)之间的优先关系和结合性质，然后借助于这种关系来比较相邻运算符的优先级，以确定句型的“可归约串”来进行归约。因此，算符优先分析法不是一种规范归约，在整个归约过程中起决定性作用的是相继两个终结符的优先关系。

附加语义的方法是采用语法制导翻译的方法，语法制导翻译的方法就是为每个产生式配上一个翻译子程序(称语义动作或语义子程序)，并在语法分析的同时执行这些子程序。语义动作是为产生式赋予具体意义的手段，它一方面指出了一个产生式所产生的符号串的意义，另一方面又按照这种意义规定了生成某种中间代码应做哪些基本动作。在语法分析过程中，当一个产生式获得匹配(对于自上而下分析)或用于归约(对于自下而上分析)时，此产生式相应的语义子程序就进入工作，完成既定的翻译任务。

本次实验完成采用语法制导翻译的方法利用算符优先分析法完成算数表达式的翻译。

1. 实验代码

#define \_CRT\_SECURE\_NO\_WARNINGS 1

#include <stdio.h>

#include<stdlib.h>

#define MAX 100

char z[MAX], a[MAX], q, op;

int i, j, k, x1, x2, x3, s[MAX];

int f(char);

void get(void);

int isnum(char);

int main()

{

printf(" 请输入表达式\n");

i = 0;

do {

i++;

scanf("%c", &a[i]);

} while (a[i] != '#');

i = j = k = 1;

z[j] = '#';

get();

while (!(z[j] == '#'&&q == '#'))

{

if (isnum(q))

{

s[k++] = q - 48;

get();

}

else if (f(z[j]) > f(q))

{

op = z[j--];

x1 = s[k - 1];

x2 = s[k - 2];

k = k - 2;

switch (op)

{

case '+':x3 = x1 + x2; break;

case'-':x3 = x1 - x2; break;

case'\*':x3 = x1 \* x2; break;

case'/':x3 = x1 / x2; break;

}

s[k++] = x3;

printf("(%c,%d,%d,%d)\n", op, x1, x2, x3);

}

else if (f(z[j]) < f(q))

{

j++;

z[j] = q;

get();

}

else if (f(z[j]) == f(q))

{

if (z[j] == '('&&q == ')')

{

j = j - 1;

get();

}

else printf("´íÎó\n");

}

else printf("´íÎó\n");

}

system("pause");

return 0;

}

int f(char op)

{

switch (op)

{

case '(':return 1;

case '+':return 2;

case'-':return 2;

case'\*':return 3;

case'/':return 3;

case')':return 4;

case '#':return 0;

default:printf("´íÎó\n");

return -1;

}

}

void get(void)

{

q = a[i];

i++;;

}

int isnum(char num)

{

if (num >= 48 && num <= 57)

{

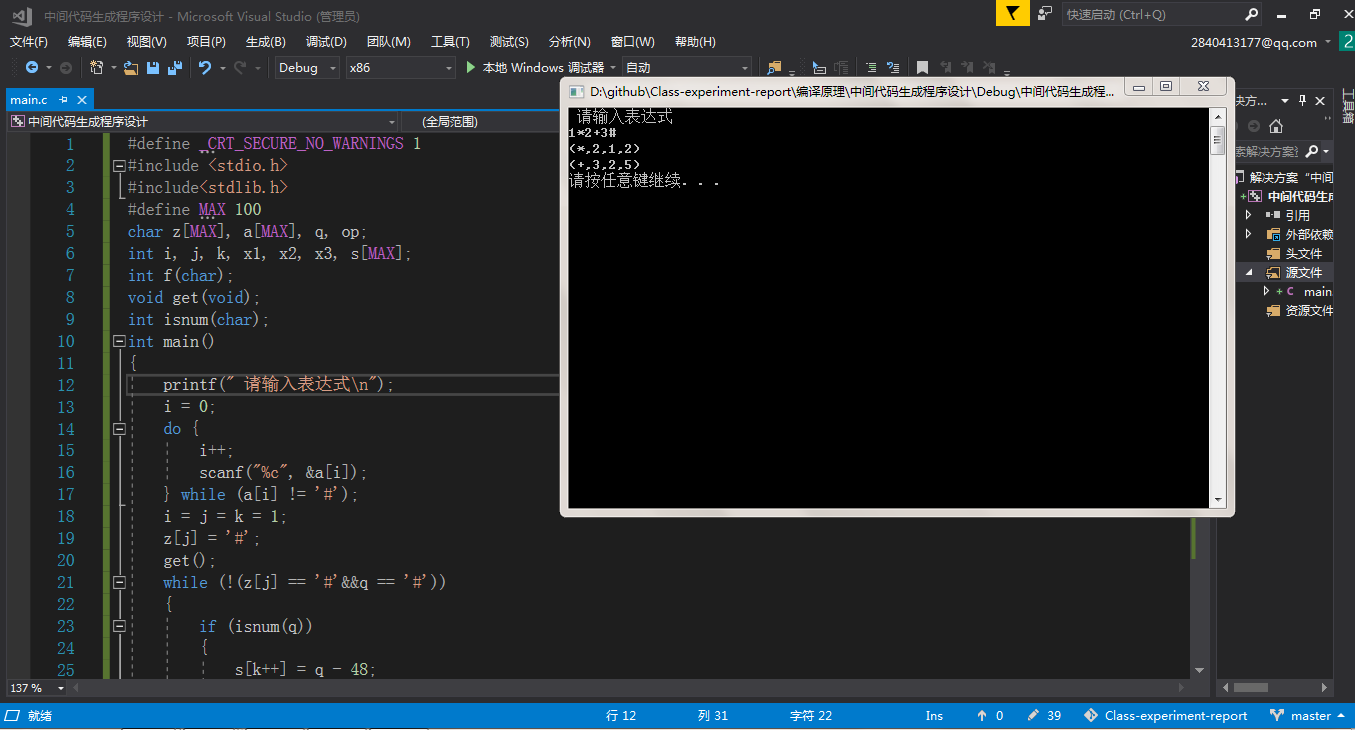
return 1;

}

return 0;

}

1. 实验结果



1. 实验结果分析

实验结果显示完成了算数表达式的中间代码生成