1. **实验目的**

设计、编制并调试一个词法分析程序，加深对词法分析原理的理解。

1. **实验要求**

**2.1 待分析的单词符号**

1. 关键字：begin if else while do end
2. 运算符和界符 : = + - \* / < <= <> > >= = ; ( ) #
3. 其他单词是标识符（ID）和整型常数（NUM），通过以下正规式定义：

ID=letter（letter| digit）\*

NUM=digit digit \*

1. 空格由空白、制表符和换行符组成。空格一般用来分隔ID、NUM,运算符、界符和关键字，词法分析阶段通常被忽略。

**2.2 单词符号对应的种别码表**

|  |  |  |  |
| --- | --- | --- | --- |
| 单词符号 | 种别码 | 单词符号 | 种别码 |
| begin | 1 | : | 17 |
| if | 2 | := | 18 |
| then | 3 | > | 20 |
| while | 4 | <> | 21 |
| do | 5 | <= | 22 |
| end | 6 | < | 23 |
| 标识符 | 10 | >= | 24 |
| 常数 | 11 | = | 25 |
| \* | 13 | ; | 26 |
| / | 14 | ( | 27 |
| + | 15 | ) | 28 |
| - | 16 | # | 0 |

**2.3 词法分析程序的功能**

输入：C语言源程序。

输出：二元式（单词种别，单词自身）

1. **实验算法思想**

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。

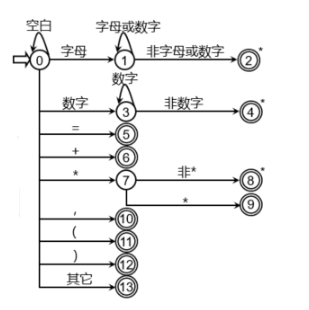
int Sum：整型常数。

int syn：单词种别码。

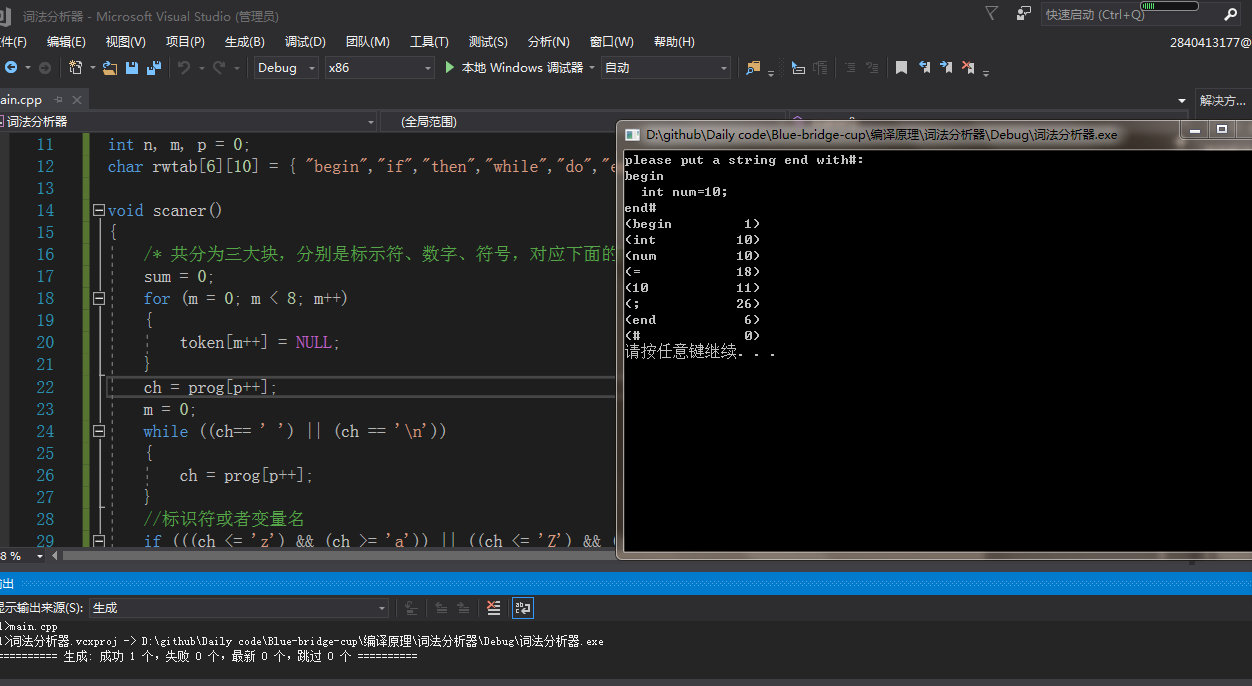
整体框架为一个if if else else，分别对应标识符，常数，符号三种情况。

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. **状态转换图**

****

1. **运行结果截图**

****

1. **实验源代码**

#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. **实验总结**

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。