## 实验一 词法分析

**一、实验目的：**

通过设计编制调试一个具体的词法分析程序，加深对词法分析原理的理解。并掌握在对程序设计语言源程序进行扫描过程中将其分解为各类单词的词法分析方法。

编制一个读单词过程，从输入的源程序中，识别出各个具有独立意义的单词，即基本保留字、标识符、常数、运算符、分隔符五大类。并依次输出各个单词的内部编码及单词符号自身值。（遇到错误时可显示“Error”，然后跳过错误部分继续显示）

**二、实验预习提示**

* 1. 词法分析器的功能和输出格式

词法分析器的功能是输入源程序，输出单词符号。词法分析器的单词符号常常表示成以下的二元式(单词种别码，单词符号的属性值)。本实验中，采用的是一类符号一种别码的方式。

* 1. 单词的BNF表示

<标识符>-> <字母><字母数字串>

<字母数字串>-><字母><字母数字串>|<数字><字母数字串>|

<下划线><字母数字串>|ε

<无符号整数>-> <数字><数字串>

<数字串>-> <数字><数字串> |ε

<加法运算符>-> +

<减法运算符>-> -

<大于关系运算符>-> >

<大于等于关系运算符>-> >=

3、“超前搜索”方法

词法分析时，常常会用到超前搜索方法。如当前待分析字符串为“a>+”,当前字符为’>’，此时，分析器倒底是将其分析为大于关系运算符还是大于等于关系运算符呢？显然，只有知道下一个字符是什么才能下结论。于是分析器读入下一个字符’+’，这时可知应将’>’解释为大于运算符。但此时，超前读了一个字符’+’，所以要回退一个字符，词法分析器才能正常运行。在分析标识符，无符号整数等时也有类似情况。

4、模块结构

缓冲区

扫描一个字符

主函数main()

N

输入文件名，判断能否打开文件

Y

缓冲区中是否还有字符

Y

结束

取单词

扫描一个字符

调用

返回

输出

N

**三、实验过程和指导：**

（一）准备：

1.阅读课本有关章节，明确语言的语法，写出基本保留字、标识符、常数、运算符、分隔符和程序例。

2.初步编制好程序。

3.准备好多组测试数据。

（二）上课上机：

将源代码拷贝到机上调试，发现错误，再修改完善。

第二次上机调试通过。

（三）程序要求：

程序输入/输出示例：

如源程序为C语言。输入如下一段：

（2，”main”）

（5，”（“）

（5，”）“）

（5，”{“）

（1，”int”）

（2，”a”）

（5，”,”）

（2，”b”）

（5，”;”）

（2，”a”）

（4，”=”）

（3，”10”）

（5，”;”）

（2，”b”）

（4，”=”）

（2，”a”）

（4，”+”）

（3，”20”）

（5，”;”）

（5，”}“）

main()

{

int a,b;

a = 10;

b = a + 20;

}

要求输出如右图。

要求：

识别保留字：if、int、for、while、do、return、break、continue；

单词种别码为1。

其他的都识别为标识符；单词种别码为2。

常数为无符号整形数；单词种别码为3。

运算符包括：+、-、\*、/、=、>、<、>=、<=、!= ；

单词种别码为4。

分隔符包括：,、;、{、}、(、)； 单词种别码为5。

以上为参考，具体可自行增删。

（四）程序思路（仅供参考）：

这里以开始定义的C语言子集的源程序作为词法分析程序的输入数据。在词法分析中，自文件头开始扫描源程序字符，一旦发现符合“单词”定义的源程序字符串时，将它翻译成固定长度的单词内部表示，并查填适当的信息表。经过词法分析后，源程序字符串（源程序的外部表示）被翻译成具有等长信息的单词串（源程序的内部表示），并产生两个表格：常数表和标识符表，它们分别包含了源程序中的所有常数和所有标识符。

0.定义部分：定义常量、变量、数据结构。

1.初始化：从文件将源程序全部输入到字符缓冲区中。

2.取单词前：去掉多余空白。

3.取单词后：去掉多余空白（可选，看着办）。

4.取单词：利用实验一的成果读出单词的每一个字符，组成单词，分析类型。（关键是如何判断取单词结束？取到的单词是什么类型的单词？）

5.显示结果。

（五）练习该实验的目的和思路：

程序开始变得复杂起来，可能是大家目前编过的程序中最复杂的，但相对于以后的程序来说还是简单的。因此要认真把握这个过渡期的练习。

本实验和以后的实验相关。通过练习，掌握对字符进行灵活处理的方法。

（六）为了能设计好程序，注意以下事情：

1.模块设计：将程序分成合理的多个模块（函数），每个模块做具体的同一事情。

2.写出（画出）设计方案：模块关系简图、流程图、全局变量、函数接口等。

3.编程时注意编程风格：空行的使用、注释的使用、缩进的使用等。

**四、上交：**

1.程序源代码（上交软盘）；

2.已经测试通过的测试数据3组；

3.实验报告：

实验名称

实验目的和要求

（一）实验内容

（1）功能描述：该程序具有什么功能？

（2）程序结构描述：函数调用格式、参数含义、返回值描述、函数功能；函数之间的调用关系图。

（3）程序总体执行流程图

（二）实验过程记录：出错次数、出错严重程度、解决办法摘要。

（三）实验总结：你在编程过程中花时多少？多少时间在纸上设计？多少时间上机输入和调试？多少时间在思考问题？遇到了哪些难题？你是怎么克服的？你对你的程序的评价？你的收获有哪些？

**五、参考源程序：**

实验一（词法分析）源代码如下：

/\*cifa fenxi chengxu\*/

#include <stdio.h>

#include <ctype.h>

#include <alloc.h>

#include <stdlib.h>

#include <string.h>

#define NULL 0

FILE \*fp;

char cbuffer;

char \*key[8]={"if","else","for","while","do","return","break","continue"};

char \*border[6]={",",";","{","}","(",")"};

char \*arithmetic[4]={"+","-","\*","/"};

char \*relation[6]={"<","<=","=",">",">=","<>"};

char \*consts[20];

char \*label[20];

int constnum=0,labelnum=0;

int search(char searchchar[],int wordtype)

{ int i=0;

switch (wordtype) {

case 1:for (i=0;i<=7;i++)

{ if (strcmp(key[i],searchchar)==0)

return(i+1);

}

case 2:{for (i=0;i<=5;i++)

{ if (strcmp(border[i],searchchar)==0)

return(i+1);

} return(0);

}

case 3:{for (i=0;i<=3;i++)

{ if (strcmp(arithmetic[i],searchchar)==0)

{ return(i+1);

}

}

return(0);

}

case 4:{for (i=0;i<=5;i++)

{ if (strcmp(relation[i],searchchar)==0)

{ return(i+1);

}

}

return(0);

}

case 5:{for (i=0;i<=constnum;i++)

{ if (strcmp(consts[i],searchchar)==0)

{return(i+1);

}

}

consts[i-1]=(char \*)malloc(sizeof(searchchar));

strcpy(consts[i-1],searchchar);

constnum++;

return(i);

}

case 6:{for (i=0;i<=labelnum;i++)

{ if (strcmp(label[i],searchchar)==0)

{ return(i+1);

}

}

label[i-1]=(char \*)malloc(sizeof(searchchar));

strcpy(label[i-1],searchchar);

labelnum++;

return(i);

}

}

}

char alphaprocess(char buffer)

{ int atype;

int i=-1;

char alphatp[20];

while ((isalpha(buffer))||(isdigit(buffer)))

{ alphatp[++i]=buffer;

buffer=fgetc(fp);

}

alphatp[i+1]='\0';

if (atype=search(alphatp,1))

printf("%s (1,%d)\n",alphatp,atype-1);

else { atype=search(alphatp,6);

printf("%s (6,%d)\n",alphatp,atype-1);

}

return(buffer);

}

char digitprocess(char buffer)

{ int i=-1;

char digittp[20];

int dtype;

while ((isdigit(buffer)))

{ digittp[++i]=buffer;

buffer=fgetc(fp);

}

digittp[i+1]='\0';

dtype=search(digittp,5);

printf("%s (5,%d)\n",digittp,dtype-1);

return(buffer);

}

char otherprocess(char buffer)

{ int i=-1;

char othertp[20];

int otype,otypetp;

othertp[0]=buffer;

othertp[1]='\0';

if (otype=search(othertp,3))

{ printf("%s (3,%d)\n",othertp,otype-1);

buffer=fgetc(fp);

goto out;

}

if (otype=search(othertp,4))

{ buffer=fgetc(fp);

othertp[1]=buffer;

othertp[2]='\0';

if (otypetp=search(othertp,4))

{ printf("%s (4,%d)\n",othertp,otypetp-1);

goto out;

}

else

othertp[1]='\0';

printf("%s (4,%d)\n",othertp,otype-1);

goto out;

}

if (buffer==':')

{ buffer=fgetc(fp);

if (buffer=='=')

printf(":= (2,2)\n");

buffer=fgetc(fp);

goto out;

}

else

{ if (otype=search(othertp,2))

{ printf("%s (2,%d)\n",othertp,otype-1);

buffer=fgetc(fp);

goto out;

}

}

if ((buffer!='\n')&&(buffer!=' '))

printf("%c error,not a word\n",buffer);

buffer=fgetc(fp);

out: return(buffer);

}

void main()

{

int i;

for (i=0;i<=20;i++)

{

label[i]=NULL;

consts[i]=NULL;

};

if ((fp=fopen("example.c","r"))==NULL)

printf("error");

else

{

cbuffer = fgetc(fp);

while (cbuffer!=EOF)

{

if (isalpha(cbuffer))

cbuffer=alphaprocess(cbuffer);

else if (isdigit(cbuffer))

cbuffer=digitprocess(cbuffer);

else cbuffer=otherprocess(cbuffer);

}

printf("over\n");

getchar();

}

}