****

编译原理实验报告

|  |  |
| --- | --- |
| 学生姓名 | 代玮宸 |
| 学 号 | 8208221431 |
| 专业班级 | 信安2202 |
| 指导教师 | 吴炜 |
| 学 院 | 计算机学院 |
| 完成时间 | 2024.5 |

目录

一、实验概述 ..3

（一）实验目的 3

（二）实验内容及要求 3

二、总体设计

（一）词法分析器总体设计……………………………………………………………………………4

（二）语法分析器总体设计……………………………………………………………………………6

三、详细设计与实现……………………………………………………..………………………………………..8

（二）词法分析器的设计与实现……………………………………………………..…………….8

（三）语法分析器的设计与实现…………………………………..………………………………16

四、实验运行结果……………………………………………………………………………………………….. 28

结束语 29

参考文献 30

一 实验概述

（一）实验目的

编译原理是计算机专业的一门重要专业课，旨在介绍编译程序构造的一般原理和基本方法。内容包括语言和文法、词法分析、语法分析、语法制导翻译、中间代码生成、存储管理、代码优化和目标代码生成。 编译原理是计算机专业设置的一门重要的专业课程。通过实践设计简单的词法分析器和语法分析器，以深入理解编译原理关于词法分析和语法分析的相关内容。

（二）实验内容及要求

1．词法分析器与语法分析器

词法分析（lexical analysis）是计算机科学中将字符序列转换为单词（Token）序列的过程。进行词法分析的程序或者函数叫作词法分析器（Lexical analyzer，简称Lexer），也叫扫描器（Scanner）。词法分析器一般以函数的形式存在，供语法分析器调用。

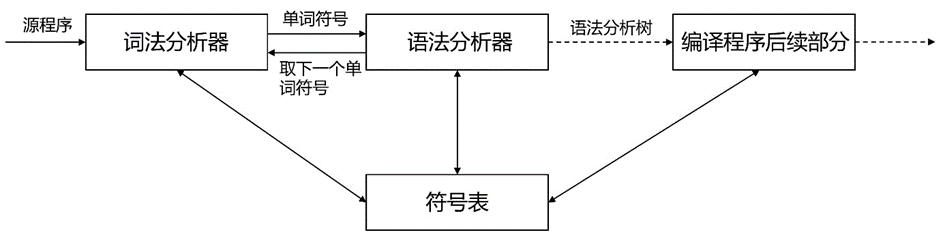
语法分析器（Parser）通常是作为编译器或解释器的组件出现的，它的作用是进行语法检查、并构建由输入的单词组成的数据结构（一般是语法分析树、抽象语法树等层次化的数据结构）。语法分析器通常使用一个独立的词法分析器从输入字符流中分离出一个个的“单词”，并将单词流作为其输入。实际开发中，语法分析器可以手工编写，也可以使用工具（半）自动生成。

图1-1 编译工作流程图

语法分析的任务使在词法分析识别出单词后，分析并判定程序的语法结构是否符合语法规则，根据语言的语法规则把单词符号分解成个语法单位（短语、字句、句子、程序段、程序），通过语法分析确定整个输入串是否构成语法上正确的“程序”，是编译过程的核心部分。词法分析器和语法分析器在编译器中的关系见上图。

2. 简单词法分析器与简单语法分析器的设计实验要求

⑴单独完成本次实验的分析器设计与实现。

⑵实验项目的基本要求如下：①理解词法分析器和语法分析器；②掌握分析器的工作原理；③编写简单的分析器，并附上程序readme.md文件。

⑶所有实验项目所需仪器设备如下：①计算机（联网状态）

⑷各个实验项目的具体要求如下：

实验项目1：编写词法分析器

实验教学的内容或要求：①理解词法分析器工作原理；②编写词法分析器；

实验项目2： 编写语法分析器

实验教学的内容或要求：①理解语法分析器工作原理；②编写语法分析器。

二 总体设计

（一）词法分析器总体设计

1．词法分析器设计要求

在计算机翻译中，词法分析的关键之一在于如何识别单词的种类，特别是面对定义形式一样、结构完全相同的标识符和保留字，如何正确地解决对它们的识别处理，是词法分析中的一项重要任务。由于我们涉及程序语句中涉及单词较少，故在词法分析阶段忽略了单词输入错误的检查，而将编译程序的重点放在中间代码生成阶段。词法分析器的功能是输入源程序，输出单词符号。

词法分析中的一个重要环节为识别单词符号类型，为了便于语法分析，通常将单词符号分为五类。

标识符

用来命名程序中出现的变量、数组、函数、过程、标号等，通常是一个字母开头的字母数字串，如 length，nextch 等。

基本字

也可以成为关键字或保留字。如 if，while，for，do，goto 等。他们具有标识符的形式，但他们不是由用户而是由语言定义的，其意义是约定的。多数语言中规定，他们不能作为标识符或者标识符的前缀，即用户不能使用它们来定义用户使用的名字，故我们称它为保留字，这些语言如 Pascal 和 C 等。但也有的语言允许将基本字作为标识符或者标识符的前缀，这类语言如 Fortran 等。

常数

包括各种类型的常数，如整型、实型、字符型、布尔型等。如：5、3.1415926、a、TRUE等都是常数。

运算符

算术运算符+、-、×、÷；关系运算符<,<=,>,>=,==,!=以及逻辑运算符&&，()，||或者!等。

界符

如，、；等单字界符和/,/,//等双字界符，空白符等。

在进行词法分析后，识别出来的单词应该采用某种中间表示形式，以便为编译后续阶段方便地引用。通常一个单词用一个二元式来表示：

（单词类别，单词的属性）

其中，第一元用于区分单词所属的类别，以整数编码表示。第二元用于区分该类别中的哪一个单词符号，即单词符号的值。

2．设计步骤

设计scaner（）函数。

我们需要对五种单词符号进行识别分析，这里将单词符号分为三大块进行识别。首先判断字符是否为关键字或者标识符，并与已定义好的关键字进行比较，从而判断为关键字或者标识符；然后是数字的识别；最后是其他字符的判断，它们被一一定义好的判断进行识别，这样所有的字符便被识别出来了。

利用选择结构（if，else if，else）对所需识别的字符进行分类，在结构体里分别对识别到的字符串做处理。scaner（）函数要做到的还有单词类别和单词的属性作返回值，这样就是函数与程序通信，达成生成二元组的目的。

设计main（）函数

依次读入源程序符号，对源程序进行单词切分和识别，直到源程序结束。对正确的单词，按照它的种别以<种别码，值>的形式保存在符号表中。对不正确的单词，做出错误处理。

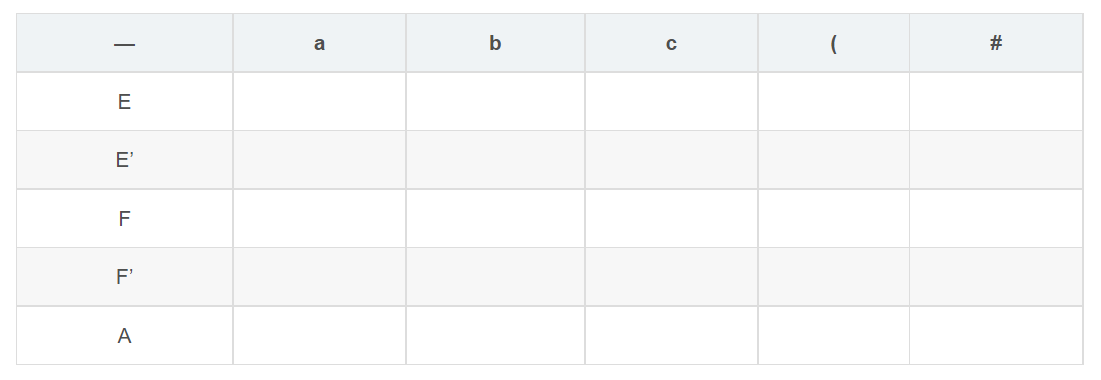
读入输入的待分析程序作字符数组内容，对读入的字符数组调用scaner（）函数并且根据二元组不同选择不同的方式进行二元组的打印，直到scaner（）函数扫描到“#”程序结束符，终止程序。main()函数需要包含错误处理，具体表现为 “Error in row + digit”。

（二）语法分析器总体设计

1．语法分析器设计要求

上而下分析法和自下而上分析法。其中自上而下的分析法包括递归下降法和LL(1)预测分析法。L L(1)中的第一个L表示从左到右扫描输入串，第二个L表示最左推导，1表示分析时每一步只需向前查看一个符号。LL(1)预测分析法以其执行效率高，便于维护的优点而被广泛采用。而本文我就以LL(1)预测分析法为主进行详细的介绍。自上而下分析的主旨是：对于任何输入串，试图用一切可能的方法，从文法开始符号（根结点）出发，自上而下的为输入串建立一课语法树析器设计要求。

预测分析法是LL(1)分析法的一种实现方法，它通过一张表来关联非终结符和终结符，这张表就是预测分析表，预测分析表可以说是预测分析法的核心部分。我们在进行该方法时，首先要构建预测分析表。

首先介绍一下预测分析表的结构，简单来说他就是一张表，表的两个属性分别是非终结符和终结符（包括‘#’），形如：  
其中对应的内容是产生式的形式，若是没有产生式则可以写入标记来表示其匹配不到具体的产生式，进入报错处理程序。

在根据文法构造预测分析表时需要扫描全部产生式：

1. 计算产生式的右部的FIRST集合，如果求出来的FIRST集合中包含终结符，那么就把这条产生式放入对应非终结符和终结符的格子中
2. 如果ε在FIRST集合中，则计算该非终结符的FOLLOW集合，如果求出来的FOLLOW集合中包含终结符，那么就把这条产生式放入对应的非终结符和终结符的格子中

当预测分析表构造完成后，我们便可以运用预测分析法进行语法分析了。我们使用用一个栈来存放过程数据，主要步骤如下：

1. 获取栈顶的元素A，获取输入串目前指针指向的元素a
2. 若A = ‘#’ ,a = ‘#’ 则匹配成功
3. 若A = a 但是A和a不为’#’，则pop栈顶元素，输入串指针+1
4. 若A为非终结符，这查询预测分析表，把由A和a确定的产生式的右部从右往左依次压入到栈中，若右部是ε，那就不做操作
5. 查找预测分析表得到预设的出错字符则调用error()

2．设计步骤

本次实验要设计实现的是预测分析程序，实验平台只要求设计根据产生式求解非终结符的FIRST（）和FOLLOW（）的find\_first函数和find\_follow函数和预测分析表的生成函数。

构造FIRST集: 扫描所有的产生式，对于每一个X，连续使用以下规则，直至每个集合的FIRST集合不再增大为止，每一遍扫描所有产生式如果有FIRST集合变化则重新扫描：

如果X是终结符，则FIRST(X) = {X}。

如果X是非终结符 ，且有产生式 X → a… 则把a加入到FIRST集合中。 若 X → ε也是其中一条产生式，则把ε也加入到FIRST集合中。

如果X是非终结符，且 X → Y… 是一条产生式，其中Y是非终结符，那么则把FIRST(Y) \ {ε}（即FIRST集合去除ε）加入到FIRST(X)中。

构造FOLLOW集: 扫描所有产生式.对于文法G的每一个非终结符A构造FOLLOW(A)可以连续使用以下规则，直至每一个FOLLOW集合不在增大为止：

对于文法的开始符号S，置 # 于FOLLOW(S)中。

若有产生式 A → αBβ，其中α和β是非终结符和终结符构成的串，B为非终结符，则把FIRST(β) \ {ε}加入到FOLLOW(B)中。

若有产生式 A → αB 或 A → αBβ ε∈FIRST(β) ，则把FOLLOW(A)加入到FOLLOW(B)中。

预测分析表构造：

根据产生式的右部的FIRST集合，如果FIRST集合中包含终结符，那么就把这条产生式放入对应非终结符和终结符的格子中

如果ε在FIRST集合中，则计算该非终结符的FOLLOW集合，如果求出来的FOLLOW集合中包含终结符，那么就把这条产生式放入对应的非终结符和终结符的格子中。

三 详细设计与实现

（一）词法分析器的设计与实现

1．详细设计

使用C++语言实现词法分析器：

#include<stdio.h>

#include<string.h>

#include<iostream>

using namespace std;

char prog[200], token[20];

char ch;

int syn, p, m = 0, n, row, sum = 0;

const char\* rwtab[10] = { "if","int","for","while","do","return","break","continue","using","namespace" };

const char\* rwtab1[8] = { "main","a","b","c","d","e","f","g" };

void scaner()

{

for (n = 0; n < 8; n++) token[n] = NULL;

ch = prog[p++];

while (ch == ' ')

{

ch = prog[p];

p++;

}

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

{

m=0;

while(1){

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

token[m++]=ch;

ch=prog[p++];

}else{

p--;

break;

}

}

token[m]='\0';

syn=0;

for(n=0;n<8;n++){

if (strcmp(token, rwtab1[n]) == 0)

{

syn = 2;

break;

}

}

for(n=0;n<10;n++){

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

syn = 1;

break;

}

}

if(!syn)syn=2;

}

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

{

sum=0;

syn = 3;

while(1){

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

sum\*=10;

sum+=ch -'0';

ch=prog[p++];

}else{

p--;

break;

}

}

}

else switch (ch) //其他字符

{

case'<':m = 0; token[m++] = ch;

ch = prog[p++];

if (ch == '>')

{

syn = 4;

token[m++] = ch;

}

else if (ch == '=')

{

syn = 4;

token[m++] = ch;

}

else if (ch == '<')

{

syn = 4;

token[m++] = ch;

}

else

{

syn = 4;

p--;

}

break;

case'>':m = 0; token[m++] = ch;

ch = prog[p++];

if (ch == '=')

{

syn = 4;

token[m++] = ch;

}

else if (ch == '>')

{

syn = 4;

token[m++] = ch;

}

else

{

syn = 4;

p--;

}

break;

case':':m = 0; token[m++] = ch;

ch = prog[p++];

if (ch == '=')

{

syn = 4;

token[m++] = ch;

}

else

{

syn = 4;

p--;

}

break;

case'/':m = 0; token[m++] = ch;

ch = prog[p++];

if (ch == '/')

{

syn = 5;

token[m++] = ch;

}

else

{

syn = 4;

p--;

}

break;

case'\*':syn = 4; token[0] = ch; break;

case'+':syn = 4; token[0] = ch; break;

case'-':syn = 4; token[0] = ch; break;

case'=':syn = 4; token[0] = ch; break;

case';':syn = 5; token[0] = ch; break;

case',':syn = 5; token[0] = ch; break;

case'(':syn = 5; token[0] = ch; break;

case')':syn = 5; token[0] = ch; break;

case'{':syn = 5; token[0] = ch; break;

case'}':syn = 5; token[0] = ch; break;

case'"':syn = 5; token[0] = ch; break;

case'#':syn = 0; token[0] = ch; break;

case'\n':syn = -2; break;

default: syn = -1; break;

}

}

int main()

{

p = 0;

row = 1;

cout << "Please input string:" << endl;

do

{

cin.get(ch);

prog[p++] = ch;

} while (ch != '#');

p=0;

do{

scaner();

switch (syn)

{

case 0: break;

case 3: cout << "(" << syn << "," << sum << ")" << endl; break;

case -1: cout << "Error in row " << row << "!" << endl; break;

case -2: row = row++;break;

default: cout << "(" << syn << "," << token << ")" << endl; break;

}

if(syn==-2)row++;

} while (syn != 0);

}

2．关键步骤解析

在scaner（）函数中，我们可以找到（if-else if-else）这样的选择结构，在if部分做读入字符串以及对字符串的种属进行判断及判断是保留字或者是标识符。具体内容如下：

while(1){

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

token[m++]=ch;

ch=prog[p++];

}else{

p--;

break;

}

}

token[m]='\0';

syn=0;

for(n=0;n<8;n++){

if (strcmp(token, rwtab1[n]) == 0)

{

syn = 2;

break;

}

}

for(n=0;n<10;n++){

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

syn = 1;

break;

}

}

if(!syn)syn=2;

以上内容中，while结构体读字符串，for循环结构体判断是否是保留字或标识符。

在else if中我们希望读入的是常数，但其实我们在prog数组中储存的是数字的ASC码，故需要进行转换同时还要注意需要读取N多位数字。具体实现内容如下：

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

{

sum=0;

syn = 3;

while(1){

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

sum\*=10;

sum+=ch -'0';

ch=prog[p++];

}else{

p--;

break;

}

}

}

在else中我们使用switch选择结构进行对ch读入字符的识别，需要注意的是有两个及两个以上的字符组成的一个符号，如：“==”，“<<”等。具体实现内容如下：

else switch (ch) //其他字符

{

case'<':m = 0; token[m++] = ch;

ch = prog[p++];

if (ch == '>')

{

syn = 4;

token[m++] = ch;

}

else if (ch == '=')

{

syn = 4;

token[m++] = ch;

}

else if (ch == '<')

{

syn = 4;

token[m++] = ch;

}

else

{

syn = 4;

p--;

}

break;

case'>':m = 0; token[m++] = ch;

ch = prog[p++];

if (ch == '=')

{

syn = 4;

token[m++] = ch;

}

else if (ch == '>')

{

syn = 4;

token[m++] = ch;

}

else

{

syn = 4;

p--;

}

break;

case':':m = 0; token[m++] = ch;

ch = prog[p++];

if (ch == '=')

{

syn = 4;

token[m++] = ch;

}

else

{

syn = 4;

p--;

}

break;

case'/':m = 0; token[m++] = ch;

ch = prog[p++];

if (ch == '/')

{

syn = 5;

token[m++] = ch;

}

else

{

syn = 4;

p--;

}

break;

case'\*':syn = 4; token[0] = ch; break;

case'+':syn = 4; token[0] = ch; break;

case'-':syn = 4; token[0] = ch; break;

case'=':syn = 4; token[0] = ch; break;

case';':syn = 5; token[0] = ch; break;

case',':syn = 5; token[0] = ch; break;

case'(':syn = 5; token[0] = ch; break;

case')':syn = 5; token[0] = ch; break;

case'{':syn = 5; token[0] = ch; break;

case'}':syn = 5; token[0] = ch; break;

case'"':syn = 5; token[0] = ch; break;

case'#':syn = 0; token[0] = ch; break;

case'\n':syn = -2; break;

default: syn = -1; break;

}

对于main函数中的内容，我们只需实现把字符串读入到prog数组，在syn！=0时及scaner函数没有读到“#”时调用scaner函数以读出单词符号二元组，并根据不同的二元组进行相对应的输出。

（二）语法分析器的设计与实现

1．详细设计

使用C++语言实现语法分析器，由于程序内容过多，现遴选重要部分做详细说明：

1.对非终结符，终结符，产生式定义

//定义非终结符

typedef struct NOterminal

{

char character;

int first\_number; //FIRST集的判定，初始为0

int follow\_number; //FOLLOW集的判定，初始为0

char\* FIRST; //FIRST 集合

char\* FOLLOW; //FOLLOW集合

struct NOterminal\* next;

} noterminal;

//定义终结符

typedef struct Terminal

{

char character; //当前的字符

struct Terminal\* next;

} terminal;

//定义产生式

typedef struct PRODUCTION

{

char source; //产生的开始

char\* result; //产生的结果

struct PRODUCTION\* next; //指向下一条

} production;

2.读文件操作及读取输入内容

size\_t read(void)

{

int line = 0, model = 0, old\_line = 0;

int number = 0;

char current\_get = 0, read\_temp[10] = { 0 };

FILE\* read\_stream = fopen("/data/workspace/myshixun/test.txt", "r");

if (!read\_stream)

{

printf("error in open file ,can`t open file\n");

exit(EXIT\_FAILURE);

}

insert\_to\_terminal('#'); //插入栈底元素，以#表示栈底

insert\_to\_terminal('^'); //插入空串元素，以^表示栈底

while (!feof(read\_stream))

{

current\_get = fgetc(read\_stream);

while (current\_get == ' ')

current\_get = fgetc(read\_stream); //忽略空格

while (current\_get == '\n')

{

current\_get = fgetc(read\_stream);

line++; //跳过换行

}

switch (line)

{

case 0:

insert\_to\_noterminal(current\_get);

break;

case 1:

insert\_to\_terminal(current\_get);

break;

case 3:

ungetc(current\_get, read\_stream);

old\_line = 1;

break;

default:

perror("error in format of program \n");

emergency(0);

break;

}

if (old\_line)

break;

}

while (!feof(read\_stream))

{

memset(read\_temp, 0, sizeof(read\_temp));

old\_line = line;

current\_get = fgetc(read\_stream);

while (current\_get == ' ')

current\_get = fgetc(read\_stream); //忽略空格

while (current\_get == '\n')

{

current\_get = fgetc(read\_stream);

line++; //跳过换行

}

model = ((line - old\_line) > model) ? (line - old\_line) : model;

switch (model)

{

case 0:

case 1:

fscanf(read\_stream, "%s", read\_temp);

insert\_to\_production(current\_get, read\_temp);

break;

case 2:

ungetc(current\_get, read\_stream);

TEST\_LIST[number] = (char\*)malloc(20);

memset(TEST\_LIST[number], 0, 20);

fscanf(read\_stream, "%s", TEST\_LIST[number++]);

break;

default:

perror("error in format of program \n");

emergency(1);

break;

}

}

fclose(read\_stream);

return success;

}

3.得出产生式的FIRST集合和FOLLOW集合

/\*\*\*\*\*\*Begin\*\*\*\*\*\*/

//求FIRST集合

size\_t find\_first(noterminal\* this\_noterminal, production\* this\_production)

{

noterminal\* this\_noterminal\_temp;

char temp[2]= {0};

char \*TEMP2,\*help;

while(this\_production->source!=this\_noterminal->character)

this\_production=this\_production->next;

if(this\_noterminal->first\_number==0)

{

this\_noterminal->FIRST=(char\*)malloc(MAX\_AMOUNT+1);

memset(this\_noterminal->FIRST,0,MAX\_AMOUNT+1);

}

while(this\_production&&this\_production->source==this\_noterminal->character)

{

TEMP2=this\_production->result;

while(\*TEMP2)

{

if(is\_appeared(\*TEMP2,terminal\_all))

{

temp[0]=this\_production->result[0];

combine(this\_noterminal->FIRST,temp);

break;

}

else

{

this\_noterminal\_temp=NOTERMINAL\_HEAD.next;

while(this\_noterminal\_temp->character!=\*TEMP2)

this\_noterminal\_temp=this\_noterminal\_temp->next;

if(this\_noterminal\_temp->first\_number==0) //没求first集

find\_first(this\_noterminal\_temp,PRODUCTION\_HEAD.next);

combine(this\_noterminal->FIRST,this\_noterminal\_temp->FIRST);

help=this\_noterminal->FIRST;

while(\*help&&\*help!='^')

help++;

if(\*help&&\*(TEMP2+1)) //包含'^'

{

while(\*help)

{

\*help=\*(help+1);

help++;

}

}

else break;

TEMP2++;

}

}

this\_production=this\_production->next;

}

this\_noterminal->first\_number=strlen(this\_noterminal->FIRST);

return success;

}

/\*\*\*\*\*\*End\*\*\*\*\*\*/

/\*\*\*\*\*\*Begin\*\*\*\*\*\*/

//求FOLLOW集合

size\_t find\_follow(noterminal\* this\_noterminal, production\* this\_production)

{

char\* help=NULL;

char\* temp\_result;

int symbol=0;

char terminal\_array[2]= {0};

noterminal\* FOLLOW\_TEMP,\*FOLLOW\_HELP;

if(this\_noterminal->follow\_number==0)

{

this\_noterminal->FOLLOW=(char\* )malloc(MAX\_AMOUNT+1);

memset(this\_noterminal->FOLLOW,0,MAX\_AMOUNT+1);

}

//第一个非终结符包含有#

if(this\_noterminal==NOTERMINAL\_HEAD.next)

\*this\_noterminal->FOLLOW='#';

while(this\_production)

{

temp\_result=this\_production->result;

//一个产生式未结尾

while(\*temp\_result)

{

if(\*temp\_result!=this\_noterminal->character)

{

temp\_result++;

continue;

}

temp\_result++;

if(!\*temp\_result)

symbol=1;

while(\*temp\_result)

{

if(is\_appeared(\*temp\_result,terminal\_all))

{

terminal\_array[0]=\*temp\_result;

combine(this\_noterminal->FOLLOW,terminal\_array);

}

else

{

FOLLOW\_TEMP=NOTERMINAL\_HEAD.next;

while(FOLLOW\_TEMP->character!=\*temp\_result)

FOLLOW\_TEMP=FOLLOW\_TEMP->next;

combine(this\_noterminal->FOLLOW,FOLLOW\_TEMP->FIRST);

help=this\_noterminal->FOLLOW;

while(\*help&&\*help!='^')

help++;

if(\*help)//包含'^'

{

while(\*help)

{

\*help=\*(help+1);

help++;

}

symbol=1;

}

else

{

symbol=0;

break;

}

}

temp\_result++;

}

if(symbol&&this\_production->source!=this\_noterminal->character)

{

FOLLOW\_HELP=NOTERMINAL\_HEAD.next;

while(FOLLOW\_HELP->character!=this\_production->source)

FOLLOW\_HELP=FOLLOW\_HELP->next;

if(FOLLOW\_HELP->follow\_number==0)

find\_follow(FOLLOW\_HELP,PRODUCTION\_HEAD.next);

combine(this\_noterminal->FOLLOW,FOLLOW\_HELP->FOLLOW);

symbol=0;

}

}

this\_production=this\_production->next;

}

this\_noterminal->follow\_number=strlen(this\_noterminal->FOLLOW);

return success;

}

/\*\*\*\*\*\*End\*\*\*\*\*\*/

4.消除左递归

void eliminate\_left\_recursion(void)

{

int number = 0;

char new\_char[3] = { 0 }, TEMP\_RESULT[20], temp\_empty[3] = { '^',0,0 };

production\* Temp\_production = PRODUCTION\_HEAD.next;

production\* Temp\_FREE;

terminal\* temp = TERMINAL\_HEAD.next;

while (Temp\_production)

{

if (Temp\_production->source == Temp\_production->result[0])

{

memset(TEMP\_RESULT, 0, sizeof(TEMP\_RESULT));

new\_char[0] = Temp\_production->source - 'A' + 'a';

//复制到新的产生式

strcat(TEMP\_RESULT, Temp\_production->result + 1);

strcat(TEMP\_RESULT, new\_char);

insert\_to\_noterminal(new\_char[0]);

insert\_to\_production(new\_char[0], TEMP\_RESULT);

insert\_to\_production(new\_char[0], temp\_empty);

//修改当前的产生式

memset(TEMP\_RESULT, 0, sizeof(TEMP\_RESULT));

strcat(TEMP\_RESULT, Temp\_production->next->result);

strcat(TEMP\_RESULT, new\_char);

memset(Temp\_production->result, 0, strlen(Temp\_production->result));

strncpy(Temp\_production->result, TEMP\_RESULT, strlen(TEMP\_RESULT));

Temp\_FREE = Temp\_production->next;

Temp\_production->next = Temp\_production->next->next;

free(Temp\_FREE);

continue;

}

Temp\_production = Temp\_production->next;

}

while (temp)

{

terminal\_all[number++] = temp->character;

temp = temp->next;

}

return;

}

5.产生预测分析表并且进行预测分析

/\*\*\*\*\*\*Begin\*\*\*\*\*\*/

//生成预测分析表

void prediction\_table(void)

{ int line=0,row=0,current\_character=0,number=0;

char\* FIRST\_CLUM,\*test\_exper;

noterminal\* temp\_noterminal=NOTERMINAL\_HEAD.next,\*temp\_noterminal21;

terminal\* temp\_terminal=TERMINAL\_HEAD.next;

production\* temp\_production=PRODUCTION\_HEAD.next;

char hah[5][7];

memset(hah,0,sizeof(hah));

for(line=0; line<5; line++)

{

for(row=0; row<7; row++)

hah[line][row]=0;

}

line=0;

while(temp\_production)

{

row=0;

if(is\_appeared(\*temp\_production->result,terminal\_all)&&(\*temp\_production->result!='^'))

{

while(terminal\_all[row]!=\*temp\_production->result)

row++;

hah[current\_character][row]=line+1;

}

else

{

temp\_noterminal=NOTERMINAL\_HEAD.next;

if(\*temp\_production->result=='^')

{

while(temp\_noterminal->character!=temp\_production->source)

temp\_noterminal=temp\_noterminal->next;

FIRST\_CLUM=temp\_noterminal->FOLLOW;

if(is\_appeared('#',FIRST\_CLUM))

{

row=0;

while(terminal\_all[row] != '#')

row++;

hah[current\_character][row]=line+1;

}

while(\*FIRST\_CLUM)

{

row=0;

while(terminal\_all[row]!=\*FIRST\_CLUM)

row++;

hah[current\_character][row]=line+1;

FIRST\_CLUM++;

}

if(temp\_production->next&&temp\_production->source!=temp\_production->next->source)

current\_character++;

temp\_production=temp\_production->next;

line++;

continue;

}

//是非终结符

while(temp\_noterminal->character!=\*temp\_production->result)

temp\_noterminal=temp\_noterminal->next;

FIRST\_CLUM=temp\_noterminal->FIRST;

while(\*FIRST\_CLUM)

{

row=0;

while(terminal\_all[row]!=\*FIRST\_CLUM)

row++;

hah[current\_character][row]=line+1;

FIRST\_CLUM++;

}

temp\_noterminal21=NOTERMINAL\_HEAD.next;

while(temp\_noterminal21->character!=temp\_production->source)

temp\_noterminal21=temp\_noterminal21->next;

if(is\_appeared('^',temp\_noterminal->FIRST)&&is\_appeared('#',temp\_noterminal21->FOLLOW))

{

row=0;

while(terminal\_all[row]!=\*FIRST\_CLUM)

row++;

hah[line][row]=line+1;

FIRST\_CLUM++;

}

}

if(temp\_production->next&&temp\_production->source!=temp\_production->next->source)

current\_character++;

temp\_production=temp\_production->next;

line++;

}

printf("\n预测分析表\n\n");

printf(" \t");

for(temp\_terminal=TERMINAL\_HEAD.next; temp\_terminal; temp\_terminal=temp\_terminal->next)

printf("%c\t",temp\_terminal->character);

temp\_noterminal=NOTERMINAL\_HEAD.next;

for(line=0; line<5; line++)

{

printf("\n%c\t",temp\_noterminal->character);

for(row=0; row<7; row++)

printf("%c\t",hah[line][row]==0?' ':(hah[line][row]-0+'0'));

temp\_noterminal=temp\_noterminal->next;

}

printf("\n\n");

system("pause");

printf("\n\n");

memset(TEST\_STACK,0,sizeof(TEST\_STACK));

init\_stack();

test\_exper=TEST\_LIST[0];

test\_exper[strlen(test\_exper)]='#';

STACK\_PUSH(NOTERMINAL\_HEAD.next->character);

while(!STACK\_EMPTY())

{

printf("分析栈\t");

for(number=0; number<=amount; number++)

printf("%c",TEST\_STACK[number]);

printf("\t剩余字符串\t%s\n",test\_exper);

if(TEST\_STACK[amount]==\*test\_exper)

{

STACK\_POP();

test\_exper++;

}

else

{

line=0;

row=0;

temp\_noterminal=NOTERMINAL\_HEAD.next;

while(temp\_noterminal->character!=TEST\_STACK[amount])

{

temp\_noterminal=temp\_noterminal->next;

line++;

}

while(terminal\_all[row]!=\*test\_exper)

row++;

row=hah[line][row];

if(!row)

break;

temp\_production=PRODUCTION\_HEAD.next;

while(--row)

temp\_production=temp\_production->next;

FIRST\_CLUM=temp\_production->result;

current\_character=strlen(FIRST\_CLUM);

FIRST\_CLUM=FIRST\_CLUM+current\_character-1;

STACK\_POP();

while(current\_character)

{

if(\*FIRST\_CLUM!='^')

STACK\_PUSH(\*FIRST\_CLUM);

FIRST\_CLUM--;

current\_character--;

}

}

}

printf("分析栈\t");

for(number=0; number<=amount; number++)

printf("%c",TEST\_STACK[number]);

printf("\t剩余字符串\t%s\n",test\_exper);

if(STACK\_EMPTY()&&\*test\_exper=='#')

printf("\n合法输入\n");

else

printf("\n不合法输入\n");

return ;

}

/\*\*\*\*\*\*End\*\*\*\*\*\*/

2．关键步骤解析

1.对find\_first函数作说明

该函数实际是求每个非终结符的FIRST集合，利用以下规则寻找符合条件的终结符：

如果产生式右部是终结符，则FIRST(X) = {a}。

如果产生式右部是非终结符 ，且有产生式 右部 → a… 则把a加入到FIRST集合中。 若 X → ε（^）也是其中一条产生式，则把ε也加入到FIRST集合中。

如果产生式右部是非终结符串，且 X → Y… 是一条产生式，其中Y是非终结符，那么则把FIRST(Y) \ {ε（^）}（即FIRST集合去除ε（^））加入到FIRST(X)中。

2.对find\_follow函数作说明

该函数实际是求每个非终结符的FOLLOW集合，利用以下规则寻找符合条件的终结符：对于文法的开始符号S，置 # 于FOLLOW(S)中。

若有产生式 A → αBβ，其中α和β是非终结符和终结符构成的串，B为非终结符，则把FIRST(β) \ {ε（^）}加入到FOLLOW(B)中。

若有产生式 A → αB 或 A → αBβ ε（^）∈FIRST(β) ，则把FOLLOW(A)加入到FOLLOW(B)中。

3.对prediction\_table函数及预测分析表构造函数作说明：

根据产生式的右部的FIRST集合，如果FIRST集合中包含终结符，那么就把这条产生式放入对应非终结符和终结符的格子中。

如果ε（^）在FIRST集合中，则计算该非终结符的FOLLOW集合，如果求出来的FOLLOW集合中包含终结符，那么就把这条产生式放入对应的非终结符和终结符的格子中。

4.关于预测分析规则说明

1.获取栈顶的元素A，获取输入串目前指针指向的元素a

2.若A = ‘#’ ,a = ‘#’ 则匹配成功

3.若A = a 但是A和a不为’#’，则pop栈顶元素，输入串指针+1

4.若A为非终结符，这查询预测分析表，把由A和a确定的产生式的右部从右往左依次压入到栈中，若右部是ε，那就不做操作

四 实验运行结果

（一）词法分析器运行结果

输入内容：

using namespace std;

int main() {

int year;

cout << "hello" << endl;

return 0;

}

#

输出结果：

(1,using)

(1,namespace)

(2,std)

(5,;)

(1,int)

(2,main)

(5,()

(5,))

(5,{)

(1,int)

(2,year)

(5,;)

(2,cout)

(4,<<)

(5,")

(2,hello)

(5,")

(4,<<)

(2,endl)

(5,;)

(1,return)

(3,0)

(5,;)

(5,})

输入内容：

using namespace std;

int main()

{

int a = 21;

int b = 10;

int c;

c = a + b;

cout << "Line 1 - c 的值是 " << c << endl ;

return 0;

}//#

输出结果：

(1,using)

(1,namespace)

(2,std)

(5,;)

(1,int)

(2,main)

(5,()

(5,))

(5,{)

(1,int)

(2,a)

(4,=)

(3,21)

(5,;)

(1,int)

(2,b)

(4,=)

(3,10)

(5,;)

(1,int)

(2,c)

(5,;)

(2,c)

(4,=)

(2,a)

(4,+)

(2,b)

(5,;)

(2,cout)

(4,<<)

(5,")

(2,Line)

(3,1)

(4,-)

(2,c)

Error in row 8!

Error in row 8!

Error in row 8!

Error in row 8!

Error in row 8!

Error in row 8!

(5,")

(4,<<)

(2,c)

(4,<<)

(2,endl)

(5,;)

(1,return)

(3,0)

(5,;)

(5,})

(5,//)

（二）语法分析器运行结果

输入内容：

E T F

+ \* ( ) i

E->E+T|T

T->T\*F|F

F->(E)|i

(i)\*i

(i)\*iii

输出结果：

产生式

1 E E+T

2 E T

3 T T\*F

4 T F

5 F (E)

6 F i

终结符

# ^ + \* ( ) i

非终结符

E T F

读取测试

(i)\*i

(i)\*iii

消除左递归

产生式

1 E Te

2 T Ft

3 F (E)

4 F i

5 e +Te

6 e ^

7 t \*Ft

8 t ^

终结符

# ^ + \* ( ) i

非终结符

E T F e t

读取测试

(i)\*i

(i)\*iii

求FIRST集

E first number=2 first=(i

T first number=2 first=(i

F first number=2 first=(i

e first number=2 first=+^

t first number=2 first=\*^

求FOLLOW集

E follow number=2 follow=#)

T follow number=3 follow=+#)

F follow number=4 follow=\*+#)

e follow number=2 follow=#)

t follow number=3 follow=+#)

预测分析表

# ^ + \* ( ) i

E 1 1

T 2 2

F 3 4

e 6 5 6

t 8 8 7 8

分析栈 #E 剩余字符串 (i)\*i#

分析栈 #eT 剩余字符串 (i)\*i#

分析栈 #etF 剩余字符串 (i)\*i#

分析栈 #et)E( 剩余字符串 (i)\*i#

分析栈 #et)E 剩余字符串 i)\*i#

分析栈 #et)eT 剩余字符串 i)\*i#

分析栈 #et)etF 剩余字符串 i)\*i#

分析栈 #et)eti 剩余字符串 i)\*i#

分析栈 #et)et 剩余字符串 )\*i#

分析栈 #et)e 剩余字符串 )\*i#

分析栈 #et) 剩余字符串 )\*i#

分析栈 #et 剩余字符串 \*i#

分析栈 #etF\* 剩余字符串 \*i#

分析栈 #etF 剩余字符串 i#

分析栈 #eti 剩余字符串 i#

分析栈 #et 剩余字符串 #

分析栈 #e 剩余字符串 #

分析栈 # 剩余字符串 #

合法输入

退出成功

结束语

编译原理这门课让我对计算机科学的底层逻辑有了更深入的理解，特别是在完成词法分析器和预测分析程序的实践作业后，我更是收获颇丰。

词法分析器是编译过程中的第一步，它负责将输入的源代码分割成一个个有意义的符号或标记，即词法单元。在实现词法分析器的过程中，我首先学习了正则表达式和有限自动机的相关知识，它们构成了词法分析的理论基础。通过编写代码来识别和处理源代码中的关键字、标识符、数字、运算符等词法单元，我深刻体会到了理论与实践相结合的重要性。

预测分析程序则是编译过程中的一个重要环节，它根据给定的语法规则来预测下一个应该出现的词法单元。在编写预测分析程序时，我遇到了不少挑战。例如，如何处理栈的信息，如何优化算法以提高效率等。通过不断地调试和优化，我逐渐掌握了预测分析的核心技术，也提高了自己的编程能力和解决问题的能力。

这次实践作业让我意识到编译原理不仅仅是一门理论课程，更是一门实践课程。只有将理论知识应用到实践中，才能真正掌握编译技术的精髓。同时，我也意识到自己在编程和算法设计方面还有很大的提升空间，需要不断学习和进步。

展望未来，我希望能够继续深入学习编译原理的相关知识，并将其应用到实际项目中。同时，我也希望能够加强自己的编程能力，提高解决复杂问题的能力。

同时，这次实践作业不仅让我对编译原理有了更深入的理解，也让我认识到了自己的不足和进步的空间。我将以此为契机，不断努力学习，提升自己的能力，为未来的学习和工作打下坚实的基础。

参考文献

[1] 百度百科.词法分析器https://baike.baidu.com/item/%E8%AF%8D%E6%B3%95%E5%88%86%E6%9E%90%E5%99%A8?fromModule=lemma\_search-box

[2]百度百科.语法分析器 <https://baike.baidu.com/item/%E8%AF%AD%E6%B3%95%E5%88%86%E6%9E%90%E5%99%A8/10598664>

[3]周静,张敏,崔艳利. 一种基于c语言词法分析器的设计[J]. 中国水运（学术版）,2007,7(5):122-123.

[4]党文婕,黄宁. 编译器语法分析器(自上而下)模块的实现—LL(1)预测分析程序[J]. 数字化用户,2018,24(28):170. DOI:10.3969/j.issn.1009-0843.2018.28.155.