2021 年秋季编译原理
实验指导书

编译原理课程实验任务及相关说明

编
译
原
理

前言

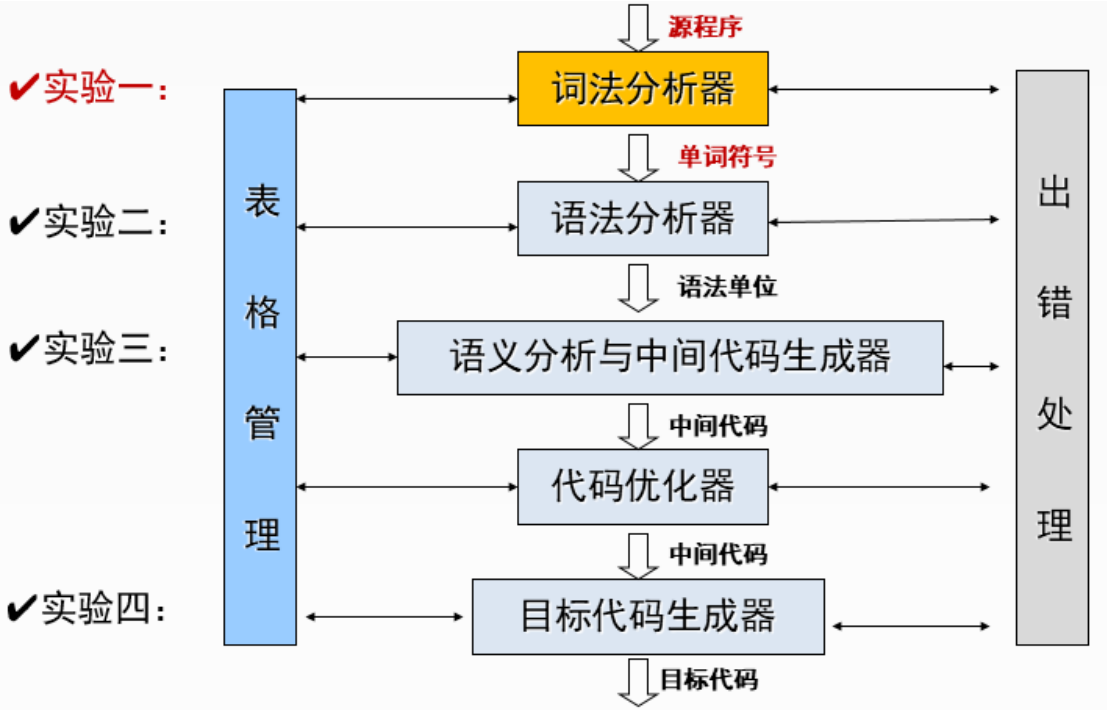
本学期的编译原理实验总共有 4 次实验，全部为设计型实验。

实验一：词法分析器的实现；

实验二：自底向上的语法分析—LR（1）；

实验三：典型语句的语义分析及中间代码生成；

实验四：目标代码生成。



实验一　词法分析器的实现

1.1 实验题目

手工设计类 C 语言的词法分析器（可以是 c 语言的子集）

1.2实验目的

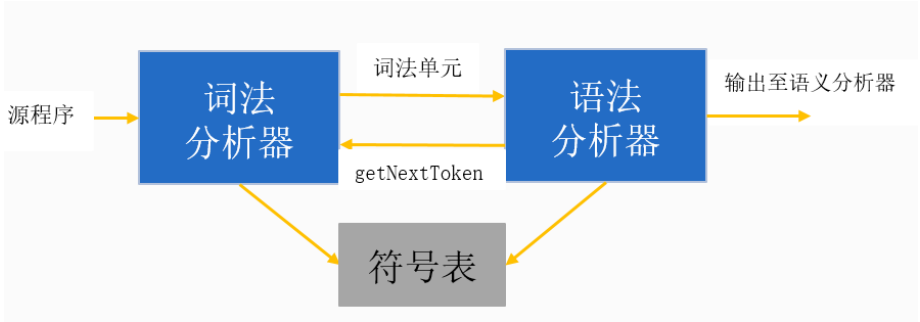
1．加深对词法分析程序的功能及实现方法的理解；
2．对类 C 语言的文法描述有更深的认识，理解有穷自动机、编码表和符号表在编译的整个过程中的应用；
3．设计并编程实现一个词法分析程序，对类 C 语言源程序段进行词法分析，加深对高级语言的认识。
4. 实验学时数：2 学时。

1.3实验内容

编写一个词法分析程序，读取代码文件，对文件内自定义的类 C 语言程序段进行词法分析。 处理 C 语言源程序，过滤掉无用符号，分解出正确的单词，以二元组形式存输出放在文件中。
1．词法分析程序输入：以文件形式存放自定义的类 C 语言程序段;
2．词法分析程序输出：以文件形式存放的 Token 串和简单符号表；
3．词法分析程序输入单词类型要求：输入的 C 语言程序段包含常见的关键字，标识符，常数，运算符和分界符等。

1.4 分析与设计

要手工设计词法分析器，实现 C 语言子集的识别，就要明白什么是词法分析器，它的功能是什么。



词法分析是编译程序进行编译时第一个要进行的任务，主要是对源程序进行编译预处理（去除注释、无用的回车换行找到包含的文件等）之后，对整个源程序进行分解，分解成一个个单词。这些单词只有五类，分别是标识符、保留字、常数、运算符、界符。可以说词法分析面向的对象是单个字符，目的是把它们组成有效的单词（字符串），从而作为语法分析的输入来分析是否符合语法规则，并且进行语法制导的语义分析产生中间代码，进而优化并生成目标代码。

综上：词法分析器的输入是源代码字符流，输出是单词序列。

1．什么是关键字（也是保留字）？

关键字是又程序语言定义的具有固定意义的标识符。例如 C 语言中的 if，else，for，while 都是保留字，这些字通常不用作一般标识符。

2．什么是标识符？

标识符用来表示各种名字，如变量名，数组名，函数名等。

3．什么是常数？

常数的类型一般是有整型，实型、布尔型、文字型等。

4．什么是运算符？

运算符如+、-、*、/等等。

5. 什么是界符？

界符如逗号、分号、括号等等。

1.4词法分析程序测试

输入程序代码：


input_code.txt

输出示例：

Token 串：

token.txt

符号表：

symbolTable.txt

1.5 实验总体步骤

1.5.1 创建编码表

词法分析器的输出是单词序列，在 5 种单词种类中，关键字、运算符、分界符都是程序设计语言预先定义的，其数量是固定的。而标识符、常数则是由程序设计人员根据具体的需要按照程序设计语言的规定自行定义的，其数量可以是无穷多个。编译程序为了处理方便，通常需要按照一定的方式对单词进行分类和编码，在此基础上，将单词表示成二元组的形式（类别编码，单词值）。

单词名称	类别编码	单词值
标识符	1	内部字符串
无符号常数(整)	2	整数值
布尔常数	3	0 或 1
字符串常数	4	内部字符串
int	5	-
if	6	-
else	7	-
……	……	……
,(逗号)	20	
;(分号)	21	-
=	22	
+	23	-
-	24	
*	25	-
(	26	-
)	27	
[	28	
]	29	

1.5.2 创建类 C 语言文法

多数程序语言单词的词法都能用正则文法来描述，基于单词的这种形式化描述会给词法分析器的设计与实现带来很大的方便，支持词法分析器的自动构造，比如 Flex、ANTLR 词法分析器生成工具。

1．正则文法表示

G=(V,T,P,S)，其中 V={S,A,B,Cdigit,no_0_digit,,char},T={任意符号}，P 定义如下

约定：用 digit 表示数字：0,1,2,…,9; no_0_digit 表示数字：1,2,…,9;

用 letter 表示字母：A,B,…,Z,a,b,…,z,_

标识符：S→letter A　　A→letter A|digit A|ε

运算符、分隔符：S->B　B->=|*|+|-|/|(|)|;

整常数：S ->no_0_digitB　　B -> digitB | ε

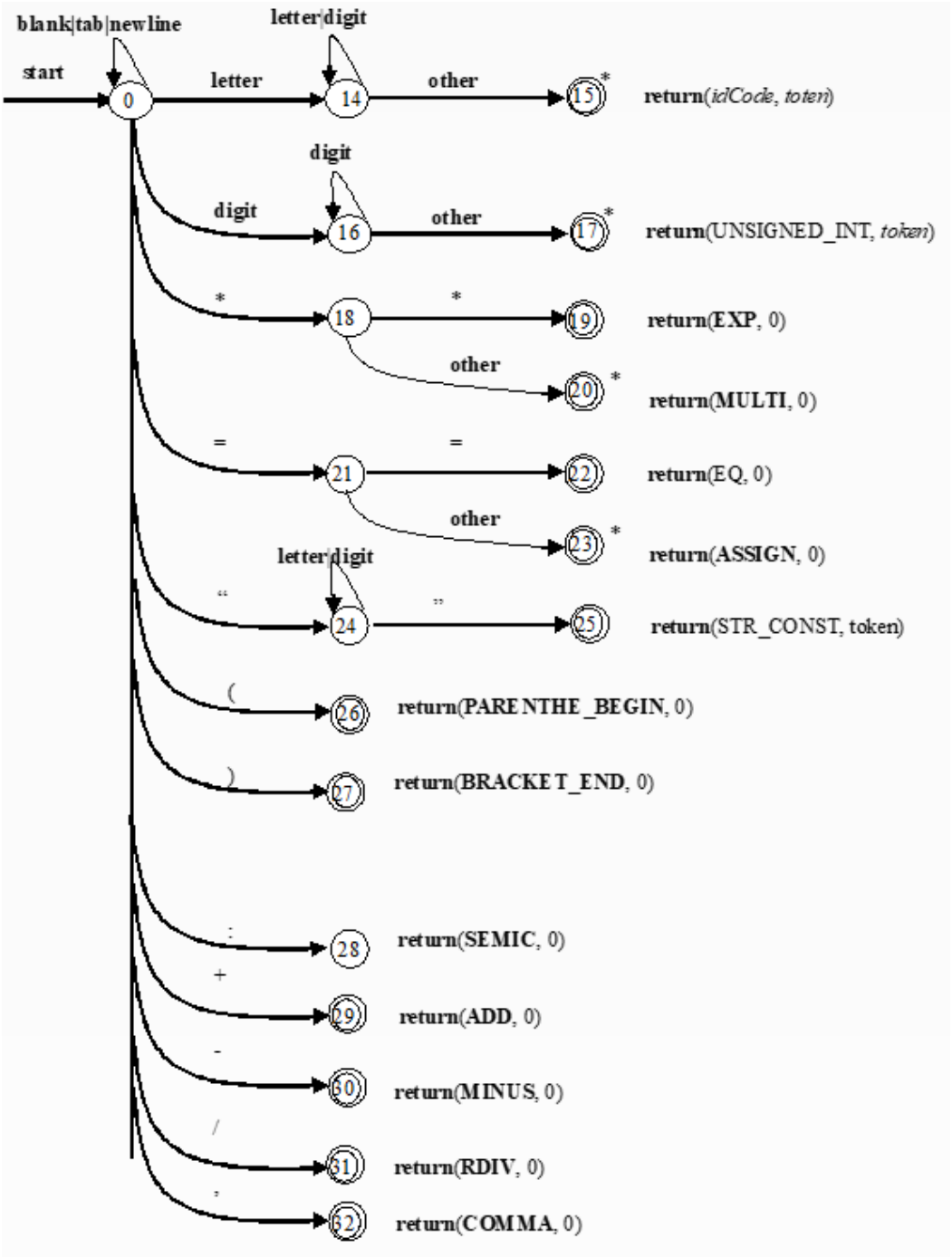
字符串常量：S -> “C”

字符常量：S->’D’

1.5.3 有限自动机

有限自动机识别的语言称为正则语言，有限自动机分为确定的有限自动机（DFA）和非确定的有限自动机（NFA）两种。DFA 和 NFA 都可以描述正则语言，DFA 规定只能有一个开始符号，且转移标记不能为空，代码实现较为方便，所以词法分析器使用 DFA 描述词法记号。

识别各类单词的状态转化图合并图：



1.5.4 编程实现

1．符号表

编译器用符号表来记录、收集和查找出现在源程序中的各种名字及其语义信息。每当源程序中出现一个新的名字时，则向符号表中填入一个新的表项来记录该名字；当在源程序中发现某个名字的新属性时，要找到该名字所对应的符号表表项，在表中记录其新发现的属性信息。因此，从数据结构的角度来看，符号表

是一个以名字为关键字来记录其信息的数据结构，支持插入和查找两个基本操作。在词法分析阶段主要是往符号表里插入名字信息。

符号表的组织结构可以是线性表或者散列表。首先设计者可能会想到选择数组作为符号表的具体物理实现结构，在符号表的规模较小的情况下，线性结构可以满足。当设计的编译器规模较大时，可以考虑引入散列表来提高查找插入的效率。

2．定义数据结构

线性结构符号表数据结构定义参考：

```
typedef struct Sym_table
{     char *name;
      int type;
      int offset;
…}SymbolTable;
```

输出二元组

```
Typedef struct tuple
{     int typeCode;
      char *value;
  };
```

3．创建关键字列表（关键字可以写入文件中，读文件读入到存储的数据结构）；

4．把源程序代码读入到输入缓冲区，利用指针移动读入下一个字符，根据有限自动机的状态转移识别一个 Token：

1）判断 Token 是标识符，查询关键字列表，判断该 Token 是否是关键字，若是：则返回关键字的种别码和关键字本身；若不是：把该 Token 插入到符号表中，返回 Token 的种别码和 Token 本身；

2）判断 Token 是常量，返回 Token 对应常量的种别码和 Token 本身；

3）判断 Token 是运算符，返回 Token 对应运算符类型的种别码和 Token 本身；

4）判断 Token 是分界符，返回 Token 对应分界符类型的种别码和 Token 本身；

1.5.5 程序测试

词法分析器输入源代码：

```
int result;
int a;
int b;
int c;
a = 8;
b = 5;
c = 3;
result = a * b + ( a - b ) + c;
```

输出结果：

```
(5,int)
(1,result)
(21,;)
(5,int)
(1,a)
(21,;)
(5,int)
(1,b)
(21,;)
(5,int)
(1,c)
(21,;)
(1,a)
(22,=)
(2,8)
(21,;)
(1,b)
(22,=)
(2,5)
(21,;)
(1,c)
(22,=)
(2,3)
(21,;)
(1,result)
(22,=)
(1,a)
(25,*)
(1,b)
(23,+)
(26,()
(1,a)
(24,-)
(1,b)
(27,))
(23,+)
(1,c)
(21,;)
```

1.6 附加功能

本实验要求实现的词法分析器的基本功能包括：标识符（含关键字类型）、常数、运算符和分界符的识别。

在完成了基本功能的基础上，可以实现对八进制、十六进制数等 c 语言的单词识别。

附加功能是非必要完成项，如完成，请在实验报告中另行标注实现方法并画出完整的有穷自动机，酌情加分但不超出实验部分的总分。

1.7 实验报告

1．写出实验中所用的 C 语言程序段的文法描述；

2．画出完整的有穷自动机（参考教材 P93(chapter3.ppt 第 55 页)）；

3． 画出实验中用到的单词的词类编码表（参考教材 P66(chapter3.ppt 第 7 页)）；

4．设计并描述符号表的逻辑结构及存储结构；

5．描述词法分析程序中主要模块的算法；

6．词法分析程序输入测试代码文件、生成的 Token 串文件和符号表文件需保存在源程序目录下一并提交。

实验二　自底向上的语法分析—LR（1）

1.1 实验题目

手工设计自底向上的语法分析器分析器，编程实现 LR（1）分析法。

1.2 实验目的

1．深入了解语法分析程序实现原理及方法。

2．理解 LR(1)分析法是严格的从左向右扫描和自底向上的语法分析方法。

3．实验学时数：8 学时。

1.3 实验内容

1．利用 LR(1)分析法，设计一个语法分析程序，对输入单词符号串进行语法分析；

2．输出推导过程中所用产生式序列并保存在输出文件中；

3．较低完成要求：书本 P186，例 5.17 中的文法或者 PPT 中参考文法；

4．较优完成要求：自行设计文法并完成实验。

5．要求：实验一的输出作为实验二的输入。

说明：实验一测试用例较复杂的情况下，实验二可能需要比较复杂的文法来完成，难度较大；可以在完成实验二时，给相对简单的测试用例。

1.4 实验总体步骤

1．定义描述程序设计语言语法的文法，并编写拓广文法；

2．构造 LR 分析表；

3．设计数据结构读入文法、LR 分析表；

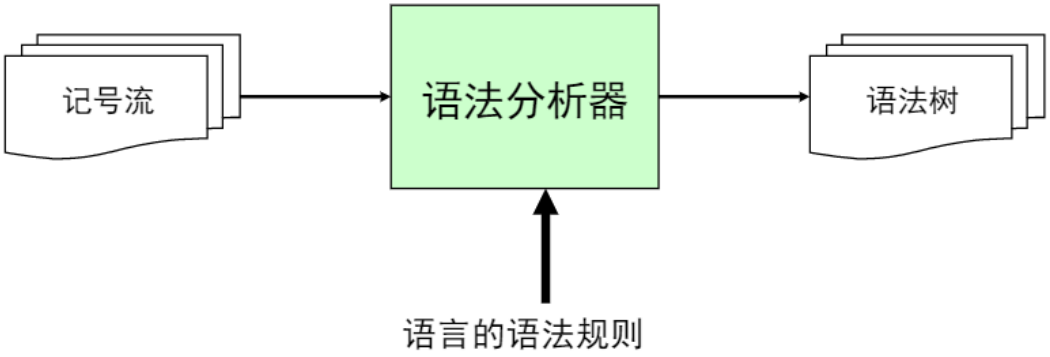
4．编写 LR 主程序完成 LR 分析器移进、归约、出错、接受四个动作；

5．输入：实验一输出的单词符号串；

6．输出：产生式序列并保存在文件中；

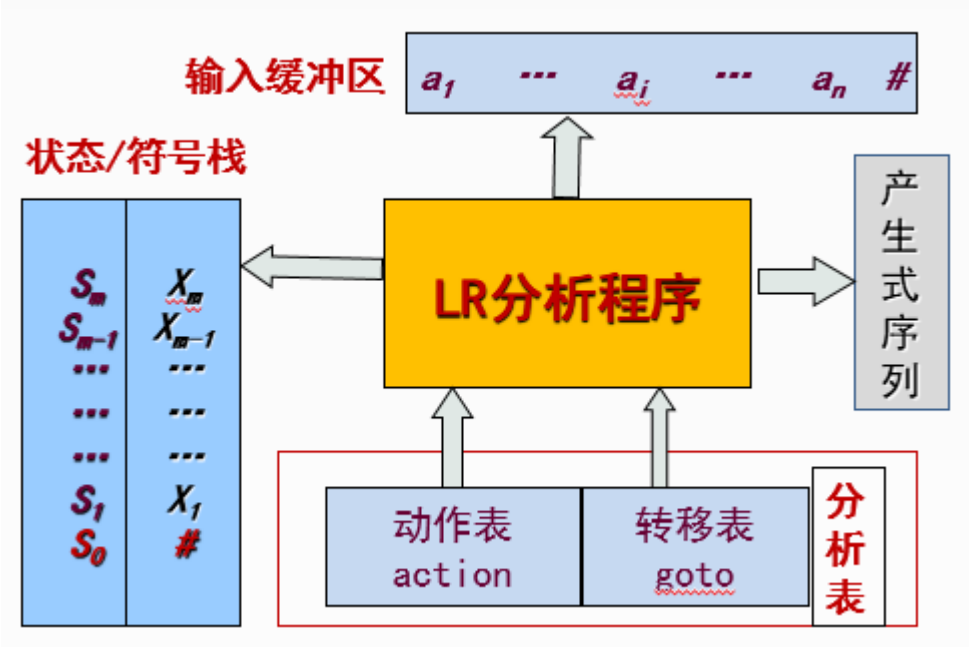
7．完成实验报告语法部分内容；

1.5 设计与实现



语法分析器的输入是词法分析器输出的单词记号流和语言的语法规则，输出是语法分析树。

1.5.1 LR 语法分析器的总体结构介绍



分析器的四种动作：

1．移进：将下一输入符号移入栈；

2．归约：用产生式左侧的非终结符替换栈顶的句柄（某产生式的右部）；

3. 接受：分析成功；

4. 出错：出错处理。

1.5.2 构造 LR(1)分析表

了解了 LR 分析器的工作过程，我们可以看到 LR 语法分析器的核心是 LR 分析程序和 LR 分析表，那么如何构造 LR 分析表呢？

构造 LR 分析表，首先需要定义描述语言的文法，并改写为拓展文法，在此基础上构造 LR(1)项目集，根据 LR(1)项目集给出基于 LR(1)项目识别所给文法全部活前缀的 DFA，有了 DFA，根据状态转移过程构造出 LR(1)分析表。

注意：求 LR(1)分析表的过程可以是手动构造、代码计算、工具构造三种方式中的任意一种。

1．手动构造 LR（1）分析表

请参考教材 P186 例 5.17。

步骤：

1）写出文法及拓广文法；

2）计算文法的项目集规范族；

3）给出有限自动机；

4）根据 DFA 的状态转移过程给出 LR（1）分析表。

2．编写程序构造 LR（1）分析表

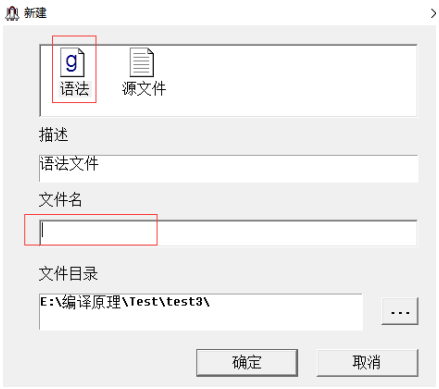
同手工构造思路，参考教材 P187 页算法 5.8 LR（1）分析表的构造。

3．工具构造

1）安装工具（Windows 平台）；

编译工作台-setup.exe

2）新建一个语法文件；



3）编写语法文件，包括四部分内容：非终结符、终结符、文法起始符、生成式，其中非终结符和终结符是指生成式中用到的符号）。

举例参考（包含声明语句、复制语句、算术表达式语句）：

非终结符：

E S P A B D

终结符：

ID ( ) + - * = INT INT_NUM

文法起始符：

P

生成式：

P -> S;

S -> D ID;

D -> INT;

S -> ID = E;

E -> E + A;

E -> E - A;

E -> A;

A -> A * B;

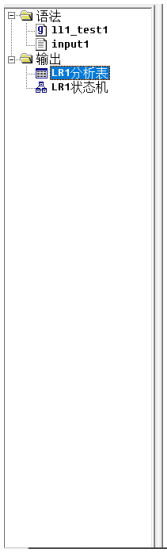
A -> B;

B -> (E);

B -> ID;

B -> INT_NUM;

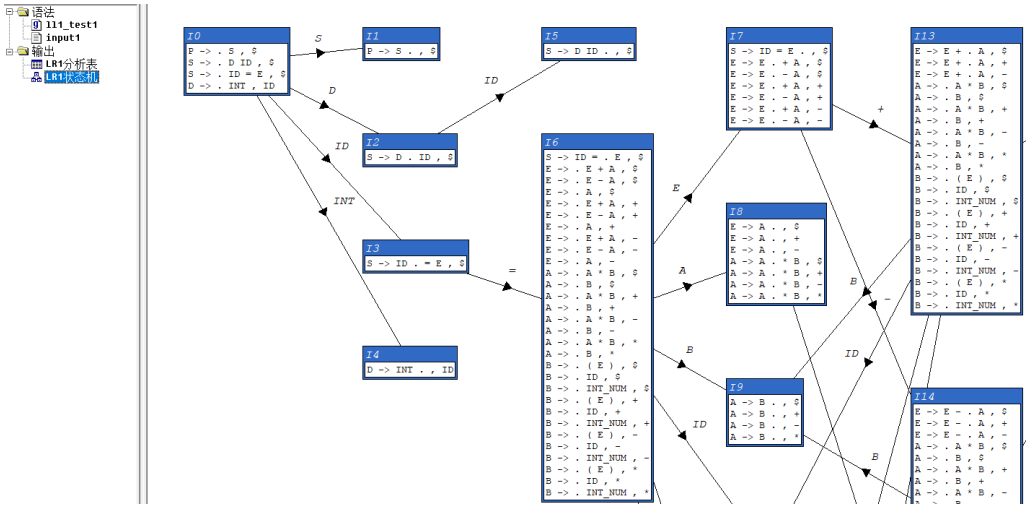
4）点击工具生成按钮，生成 LR（1）分析表和状态机；



LR1分析表

状态	ACTION								
	ID	(	)	+	-	*	=	INT	INT_NUM
0	shift 3							shift 4	
1									
2	shift 5								
3							shift 6		
4	reduce D -> INT								
5									
6	shift 10	shift 11							shift 12
7				shift 13	shift 14				
8				reduce E -> A	reduce E -> A	shift 15			
9				reduce A -> B	reduce A -> B	reduce A -> B			
10				reduce B -> ID	reduce B -> ID	reduce B -> ID			
11	shift 19	shift 20							shift 21
12				reduce B -> INT_NUM	reduce B -> INT_NUM	reduce B -> INT_NUM			
13	shift 10	shift 11							shift 12
14	shift 10	shift 11							shift 12
15	shift 10	shift 11							shift 12
16			shift 25	shift 26	shift 27				
17			reduce E -> A	reduce E -> A	reduce E -> A	shift 28			
18			reduce A -> B	reduce A -> B	reduce A -> B	reduce A -> B			
19			reduce B -> ID	reduce B -> ID	reduce B -> ID	reduce B -> ID			
20	shift 19	shift 20							shift 21
21			reduce B -> INT_NUM	reduce B -> INT_NUM	reduce B -> INT_NUM	reduce B -> INT_NUM			
22				reduce E -> E + A	reduce E -> E + A	shift 15			
23				reduce E -> E - A	reduce E -> E - A	shift 15			
24				reduce A -> A * B	reduce A -> A * B	reduce A -> A * B			

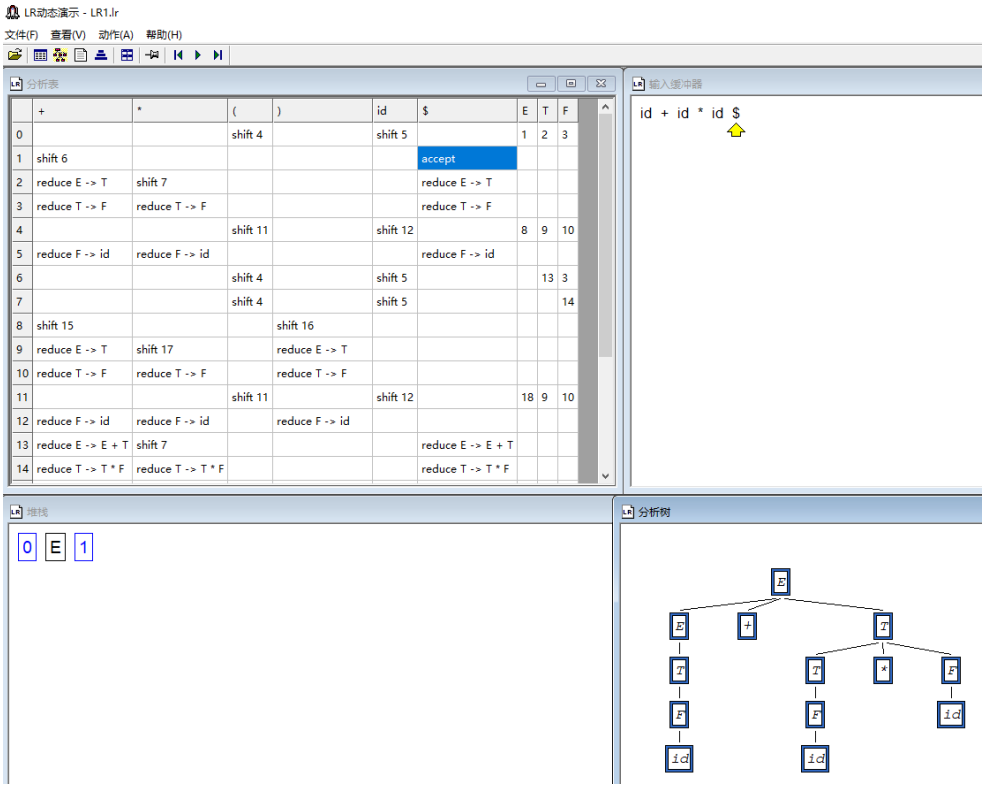
补充说明：生成的 LR（1）分析表可以从工具导出到 excle，方法：点击右键—导出到 Mircosoft Excel。



5）动态分析句子

新建一个源文件，输入要分析的句子；

点击生成-动态分析，模拟动态生成语法树的过程。



	+	*	(	)	id	$	E	T	F
0			shift 4		shift 5		1	2	3
1	shift 6					accept			
2	reduce E -> T	shift 7				reduce E -> T			
3	reduce T -> F	reduce T -> F				reduce T -> F			
4			shift 11		shift 12		8	9	10
5	reduce F -> id	reduce F -> id				reduce F -> id			
6			shift 4		shift 5			13	3
7			shift 4		shift 5				14
8	shift 15			shift 16					
9	reduce E -> T	shift 17		reduce E -> T					
10	reduce T -> F	reduce T -> F		reduce T -> F					
11			shift 11		shift 12		18	9	10
12	reduce F -> id	reduce F -> id		reduce F -> id					
13	reduce E -> E + T	shift 7				reduce E -> E + T			
14	reduce T -> T * F	reduce T -> T * F				reduce T -> T * F			

1.5.3 LR(1)分析表内容说明

1. 状态

编号 0～n 表示 LR 分析表有 n+1 个状态，对应 LR 项目集规范族的 n+1 个项目。

2. ACTION 表

也叫动作表，表头是文法定义里的所有的终结符。动作表里出现的动作有两种，shift 和 reduce。

shift x 表示将要执行的动作是移进。LR 分析器的符号栈移进读入的符号，状态栈移进状态 x，保持移进后符号栈和状态栈栈顶高度一致。

reduce X->x (假设 X->x 是某个产生式)，表示将要执行归约操作。LR 分析器从符号栈找到待归约的句柄（待归约的句柄内容应该与产生式右部相同）弹出栈将其归约成 X（X 压入符号栈），同时状态栈弹出与产生式右部 x 相同个数的状态。

3. GOTO 表

也叫转移表，表头是文法定义里的非终结符。当 LR 分析器执行归约动作后，

根据符号栈栈顶的非终结符和状态栈栈顶的状态在 GOTO 表中找到对应的状态并压入状态栈，执行完此操作后，符号栈和状态栈栈顶高度一致。

1.5.4 读入 LR 分析表

1．硬编码的方式

硬编码是指将可变变量用一个固定值来代替的方法。简单来说就是目标固定值在程序中写死，不可改变。

缺点：代码的拓展灵活性差，维护成本高；

优点：相对而言，某种程度上代码编写简单。

2．接口类

程序的输入输出用函数的方式读取并通过接口的方式传递。

读 LR 分析表接口实现思想（仅供参考）：

任务一：数据处理，把数据按照某种规格处理成易于存储的格式；

第一步：给每一个产生式编号，假如编号从 0 开始，那么上边的 12 个产生式编号为 0 到 11；

第二步：LR 分析表中的所有的归约操作的产生式用产生式对应的编号替换；

第三步：shift x 替换成 s x（其中 x 是状态），reduce x 替换成 r x（其中 x 是产生式编号）；

原 LR 分析表（仅供参考）：

old-LR-table.xlsx

处理后的 LR 分析表（仅供参考）：

n e w - L R - t a b

任务二：定义存储结构，编写接口读入 csv 格式文件（可以把 xls 文件转成 csv 格式存储，容易读文件）。

1.5.5 编写 LR 分析器代码

（参考教材 P171 算法 5.5 LR 分析算法）

1．参考数据结构

```
//产生式
typedef struct production    {
    int left_type_code;          //左部编码（把每个非终结符定义成一个整数）
    int right_len;               //右部长度
    char production[MAXSTR];     //产生式原型
}PROD;

//LR 分析表的动作，某个动作对应的状态；归约的产生式编号
typedef struct action_table    //状态转移表的每个元素
{
    char action;      //动作，s 移进，r 归约，a 接受，否则出错
    int state;        //跳转状态
    int PNO;          //选择的产生式编号
}ACT;
//LR 分析栈
typedef struct state_and_word
{
    int state;            //当前状态
    int type_code;        //当前成分的编码
    int TPNO;             //当前成分在输入二元组序列（词法分析输出）中的编号
}SC;
PROD prod[MAXPRODUCTION];     //存储产生式
ACT action[MAXTYPE][MAXTYPE];   //二维数组，第一维是 LR 分析表的状态标号，第二维是所给 LR 分析表对应文法中终结符和非终结符的集合。
```

2．算法

```
stack<SC> lr_stack;     //定义分析栈
SC tempElem;            //临时变量，栈元素，并初始化
Lr_stack.push(tempElem);   //初始状态压入栈
```

TP *temp_input = p;　　//p 是传进来的指向输入二元组的首指针

//当查询 LR 分析表状态不是接受状态时，循环执行以下操作：

if(action[lr_stack.top().state][temp_input->type_code].action] == ‘s’){

　　//执行移进动作，给 tempElem 赋值，并把 tempElem 压入栈，输入指针指向下一个输入}

else if(action[lr_stack.top().state][temp_input->type_code].action] == ‘r’){

//根据栈顶状态和读入的字符在 LR 表中对应归约的产生式编号，在 prod 数组中查找到对应的产生式并打印出来

//根据在 prod 中存储的产生式右部的长度，从分析栈中弹出对应个数的元素

//根据产生式左部非终结符合栈顶状态在 action 表中找到对应的状态

//把产生式左部非终结符和状态赋值给 tempElem，并把 tempElem 压入分析栈

}else{//出错处理}

1.5.6 程序测试

　　编写文法支持分析的句子来测试 LR 分析代码；例如上文给出的文法支持以下的句子；

输入源代码：

```
int result;
int a;
int b;
int c;
a = 8;
b = 5;
c = 3;
reasult = a * b + ( a - b ) + c;
```

输出结果：

```
D->INT
S->D ID
Accept a sentence.
```

D->INT

S->D ID

Accept a sentence.

D->INT

S->D ID

Accept a sentence.

D->INT

S->D ID

Accept a sentence.

B->INT_NUM

A->B

E->A

S->ID=E

Accept a sentence.

B->INT_NUM

A->B

E->A

S->ID=E

Accept a sentence.

B->INT_NUM

A->B

E->A

S->ID=E

Accept a sentence.

B->ID

A->B

B->ID

A->A*B

E->A

B->ID

A->B

E->A

B->ID

A->B

E->E-A

B->(E)

A->B

E->E+A

B->ID

A->B

E->E+A

S->ID=E

Accept a sentence.

1.6 注意事项

1．建议在实验一的基础上完成实验二。

2．实验二中读取输入二元组完成语法分析可能只需要读取单词的种别编码，但要保留单词的属性值，在实验三的语义分析中需要用到，切记不能丢掉单词属性值。

3．语法分析器的实验也可以使用自顶向下的语法分析方法（如递归下降分析方法）实现，实验报告内容自行编写，但要求说明实验原理方法和过程。

4．编程能力较好的同学可以尝试完成稍复杂语法的语法分析程序。

1.7 实验报告

1．构造所给文法的 LR(1)分析表；

2．设计上述 LR(1)分析表的存储结构；

3．主要模块的算法功能；

4．实验中用到的特色方法或设计技巧；

5．输入二元组、LR 分析表、输出产生式序列需保存在源程序目录下一并提交。

实验三 典型语句的语义分析及中间代码生成

1.1 实验题目

在语法分析基础上，利用语法制导的翻译方法实现语义分析，生成中间代码。

1.2 实验目的

1．加深对自顶向下语法制导翻译技术的理解与掌握。

2．加深对自底向上语法制导翻译技术的理解与掌握。

3．巩固对语义分析的基本功能和原理的认识，理解中间代码生成的作用。

4．实验学时数：4 学时

1.3 实验内容

1．针对自顶向下或者自底向上分析法（二选一）中所使用的文法，在完成实验二（语法分析）的基础上为语法正确的单词串设计翻译方案。

2．利用该翻译方案，对所给程序段进行分析，输出生成的中间代码序列和符号表，并保存在相应文件中。

3．中间代码可选三地址码的三元式表示或者四元式表示（不限）。

4．较低完成要求：简单赋值语句和算术表达式的语义分析与中间代码生成（参考教材 P267 或实验指导书）。

5．较优完成要求：除赋值语句外，实现声明语句、控制结构、布尔表达式等语法结构的语义分析与中间代码生产。

1.4 实验总体步骤

1. 在完成实验二（语法分析）的基础上为语法正确的单词串设计翻译方案；

2. 利用该翻译方案，对所给程序段进行分析，编写主函数与语义子程序；

3. 输出：三地址码表示的中间代码序列，并以文件格式保存；扩展的符号表，并以文件格式保存；

1.5 设计与实现

1.5.1 S-属性定义

只含综合属性的语法制导定义称为 S-属性定义，又称为 S-属性文法；

自底向上语法制导翻译的属性计算方法：对分析树进行后根遍历，并在最后一次遍历节点 N 时计算与节点 N 相关联的属性。

postorder(*N*) {

for *N* 的每个子节点 *M*(从左到右)*postorder*(*M*);

计算与节点 *N* 相关联的属性;

}

1.5.2 L-属性定义

定义：一个语法制导定义被称为 L-属性定义，当且仅当它的每个属性或者是综合属性，或者是满足如下条件的继承属性：设有产生式 A→X1X2…Xn，其右部符号 Xi(1≤i≤n)的继承属性只依赖于下列属性：

1．A 的继承属性；

2．产生式中 Xi 左边的符号 X1、X2、…、Xi-1 的综合属性或继承属性；

3．Xi 本身的综合属性或继承属性，但前提是 Xi 的属性不能在依赖图中形成回路。

L-属性定义又称为 L-属性文法。

自顶向下语法制导翻译的属性计算方法：

```
visit(N) {
    for N 的每个子节点 M(从左到右) {
        计算节点 M 的继承属性;
        visit (M);}
    计算节点 N 的综合属性;};
```

1.5.3 翻译方案

P -> S;

{P.addr =S.addr ;}

S -> D ID;　　//变量声明语句，语义分析阶段重点是确定变量类型填符号表

{Enter(ID.name, D.type, offset);offset =offset+D.width;}　//为名字创建一个符号表表项，如已存在，更新符号表；

D -> INT;

D.type=int; D.width=4;

S -> ID = E;

{p = lookup(ID.name); if p == nil then error;

gencode(p‘=’ E.addr);}

E -> E + A;

{E.addr = newtemp();

gencode(E.addr‘=’E_1.addr‘+’A.addr);}

E -> E - A;

{E.addr = newtemp();

gencode(E.addr‘=’E_1.addr‘-’A.addr);}

E -> A;

{E.addr =A.addr ;}

A -> A * B;

{A.addr = newtemp();

gencode(A.addr‘=’A_1.addr‘*’B.addr);}

A -> B;

{A.addr =B.addr ;}

B -> (E);

{B.addr =E.addr ;}

B -> ID;

{B.addr = lookup(id.name); if B.addr == null then error;}

B -> INT_NUM;

{ B.addr = INT_NUM.lexical;}

参考学习视频：MOOC 哈尔滨工业大学《编译原理》,第 12 讲中间代码生成_2,6-3 简单复制语句的翻译。

https://www.icourse163.org/learn/HIT-1002123007?tid=1450215473#/learn/content?type=detail&id=1214538629&sm=1

1.5.4 扩展符号表

1．符号表

以名字为关键字记录其信息的数据结构。支持的基本操作包括插入、查找和删除。

2．符号表的数据结构类型

线性表（优点：数据结构简单直观；缺点：时间复杂度高）；

散列表（优点：查找插入效率高；缺点：数据结构较复杂）。

3．实验一的基础上扩展符号表：

说明：实验一符号表里只添加了名字，经过语义分析后分析出来名字的属性（包括符号种类、类型、地址以及扩展属性等）并填入到符号表中。

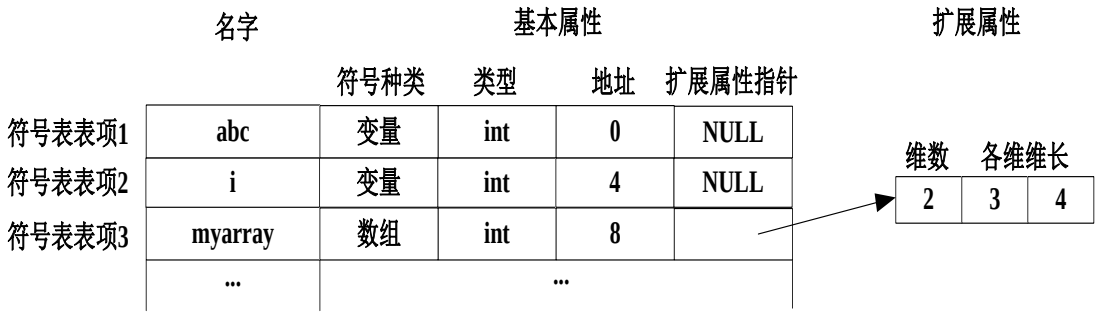


图 多种符号共用符号表的一种实现结构

1.5.5 输出中间代码

三地址码是指这种代码的每条指令最多只能包含 3 个地址，即两个操作数地址和一个结果地址。在三地址码中，一条指令的右部最多只允许出现一个运算符。

1．三地址码的两种表示

四元式：是一种比较常用的中间代码形式，由 4 个域组成，分别称为 op、arg1,、arg2 和 result。op 是一个一元或二元运算符，arg1 和 arg2 分别是 op 的两个运算对象，它们可以是变量、常量或编译器生成的临时变量，运算结果放入 result 中。

三元式：为了节省临时变量的开销，有时也可以使用只有 3 个域的三元式来表示三地址码。三元式的 3 个域是 op、arg1 和 arg2，如下图（b）所示。

三元式表示：区别只是 arg1 和 arg2 可以是某个三元式的编号，(图 (b)中用圆括号中的数字)表示用该三元式的运算结果作为运算对象。

	op	arg_1	arg_2	result
0	minus	b		t_1
1	+	c	d	t_2
2	*	t_1	t_2	t_3
3	+	c	d	t_4
4	-	t_3	t_4	t_5
5	assign	t_5		a
		...		

图（a）四元式

	op	arg_1	arg_2
0	minus	b	
1	+	c	d
2	*	(0)	(1)
3	+	c	d
4	-	(2)	(3)
5	assign	a	(4)
		...	

图（b）三元式

1.6 附加功能

非必要完成项，如完成，请在实验报告中另行标注实现方法，酌情加分但不超出实验部分的总分。

完成附加功能：补全嵌套的声明语句，控制结构，布尔表达式的语义分析与中间代码生成。

1.7 注意事项

1．采用自底向上或自顶向下语法制导翻译任一方案完成实验均可，结合实验二选择合适的方式完成实验三。

2．语义分析输出中间代码和扩展符号表都需要单词的属性值（词法分析输出二元组里单词词法值），请在实验二保留词法值，以便于实验三在实验二的基础上完成。

1.8 实验报告

1．写出翻译方案；

2．写出主要产生式语义翻译时要用到的数据结构；

3．写出语义分析后生成的中间代码序列和符号表；

4．使用中用到的特色方法或设计技巧。

实验四 目标代码生成

1.1 实验题目

在实验三的基础上，完成目标代码生成。

1.2 实验目的

1．加深对编译器总体结构的理解与掌握；

2．加深对汇编指令的理解与掌握；

3．对指令选择，寄存器分配和计算顺序选择有较深的理解。

4．实验学时数：4 学时。

1.3 实验内容

1．将中间代码所给的地址码生成目标代码（汇编指令）；

2．写出代码序列表（参考 P407 页）；

3．减少程序与指令的开销，进行部分优化（可选）；

4．较低完成要求：将赋值语句 d=(a-b)+(a-c)+(a-c)翻译为中间代码，并将其转化为目标汇编指令；

5．较优完成要求：自定义程序段(可使用实验三的中间代码)，并将其转化为目标汇编指令。完成赋值，一元运算，数组元素引用，数组元素赋值，指针引用，指针赋值，无条件跳转，条件跳转等中的其中任选三个及以上。

6．汇编指令（可自选，任选其一）

	对应课程
X86	汇编语言
Arm	嵌入式
Mips	计算机组成原理，计算机设计与实践，计算机体系结构

1.4 代码生成算法

1.4.1 目标代码生成算法（参考 P406）

对每个形如 x = y op z 得三地址语句，给出总体框架：

1）调用函数 getreg(i:x:= y op z)确定可用于保存 y op z 的计算结果的位置 L。L 通常是寄存器，也可能是内存单元。

2）查看 y 的地址描述符以确定 y 值当前的一个位置 y′。如果 y 值当前既在内存单元中又在寄存器中，则选择寄存器作为 y'。如果 y 的值还不在 L 中，则生成指令 MOV y′，L。

3）生成指令 op z′，L，其中 z′是 z 的当前位置之一。

4）如果 y 和/或 z 的当前值没有后续引用，在块的出口也不活跃，并且还在寄存器中，则修改寄存器描述符以表示在执行了 x:=y op z 之后，这些寄存器分别不再包含 y 和(或)z 的值。

1.4.2 寄存器选择函数 getreg

函数 getreg 返回保存 x = y op z 的 x 值的位置 L：

1）如果变量 y 在 R 中,且 R 不含其它变量的值,并且在执行 x:=y op z 后 y 不会再被引用,则返回 R 作为 L；

2）否则，返回一个空闲寄存器，如果有的话；

3）否则，如果 x 在块中还会再被引用，或者 op 是必须使用寄存器的算符，则找一个已被占用的寄存器 R(可能产生 MOV R，M 指令，并修改 M 的地址描述符);

4）否则，如果 x 在基本块中不会再被引用，或找不到适当的被占用寄存器，则选择 x 的内存单元作为 L。

1.4.3 常用三地址码的代码生成

复制：a:=b

1. 如果 b 的当前值在寄存器 R 中，则不必生成代码，只要将 a 添加到 R 的寄存

器描述符中，并把 a 的地址描述符置为 R 即可。

2．如果 b 在基本块中不会再被引用且在基本块的出口也不活跃，则还要从 R 的寄存器描述符中删除 b，并从 b 的地址描述符中删除 R。

3．但若 b 的当前值仅在内存单元中，如果只是简单地将 a 的地址描述符置为 b 的内存地址，那么，若不对 a 的值采取保护措施，a 的值将会为 b 的再次定义所影响。此时，生成一条形如 MOV b, R 的指令会较为稳妥。

一元运算：a:= op b

与二元运算的处理类似。

数组元素引用：a:=b[i]

假设 a 在基本块中还会再被引用，而且寄存器 R 是可用的，则将 a 保留在寄存器 R 中。于是，如果 i 的当前值不在寄存器中，则生成如下指令序列：

MOV i, R

MOV b(R), R　　开销=4

如果 i 的当前值在寄存器 R_i 中，则生成如下指令：

MOV b(R_i), R　　开销=2

与二元运算的处理类似。

3．示例

例：赋值语句 d = (a – b) + (a –c) + (a – c)

编译产生的三地址码序列为：

$t_1 = a - b$

$t_2 = a - c$

$t_3 = t_1 + t_2$

$d = t_3 + t_2$

假设 d 在基本块的出口是活跃的。根据代码生成算法讲为以上三地址码生成下标所示的代码序列，表中给出了代码生成过程中寄存器描述符和地址描述符的值

1.5 附加功能

非必要完成项，如完成，请在实验报告中另行撰写相应的实验内容，酌情加分但不超出实验部分的总分。

本课程全部四个实验（词法分析，语法分析，语义分析和中间代码生成，目标代码生成）整合为一个连贯的编译器程序，包括必要的功能选择窗口与各阶段的输出，并附有简单的用户说明文档。

语　句	生成的代码	寄存器描述符	名字地址描述符
$t_1 := a - b$	MOV a, R_0 SUB b, R_0	R_0 含 t_1	t_1 在 R_0 中
$t_2 := a - c$	MOV a, R_1 SUB c, R_1	R_0 含 t_1 R_1 含 t_2	t_1 在 R_0 中 t_2 在 R_1 中
$t_3 := t_1 + t_2$	ADD R_1,R_0	R_0 含 t_3 R_1 含 t_2	t_3 在 R_0 中 t_2 在 R_1 中
$d := t_3 + t_2$	ADD R_1,R_0 MOV R_0, d	R_0 含 d	d 在 R_0 中 d 在 R_0 和内存中

1.6 实验报告

1．写出代码生成算法所产生的代码序列；

2．比较优化后的程序开销，并计算其开销值；

3．画出一个简单的代码生成器流程图。

第二部分 补充内容

一 工程文件提交说明

1．输入测试源程序；

2．词法分析输出存放 Token 串文件；

3．词法分析输出的符号表文件；

4．语法分析输入的 LR 分析表文件（Excel 文件或者文本文件）；

5．语法分析输入的定义语言文法的文件；

6．语法分析输出的产生式序列文件；

7．语义分析输出的中间代码文件；

8．语义分析输出更新后的符号表文件（如果在语义分析有填充符号表）；

9．生成的目标代码文件；

10. 编译器词法分析、语法分析、语义分析、目标代码生成源程序完整工程文件。

二 实验报告说明

1　实验目的与方法

说明本课程四次实验的实验目的；

实验程序编写语言；运行软件环境。

2　实验内容及要求

编译器四个阶段（词法分析、语法分析、语义分析、目标代码生成）各实验的要求和实验内容概述。

3　实验总体流程与函数功能描述

描述编译器开发各个阶段流程（可用流程图、伪代码、自然语言表述）；

各阶段数据结构、核心功能函数算法描述；

4　实验结果与分析

编译器各阶段输入输出文件说明，各阶段运行结果截图和分析；

5　实验中遇到的困难与解决办法

描述实验中遇到的困难与解决办法，对实验的意见与建议、收获。

三 参考示例说明

1 输入输出示例（参考）

输入：

```
f = 2*10+2*4#
c = f-2#
d = c/10#
a = 10/d+3
```

输出的三地址码：

```
A = 2 * 10
A = 2 * 4
f = A + A
c = f - 2
d = c / 10
A = 10 / d
a = 3 + A
```

输出的符号表:

```
symbol:   token
value:    a
intValue:8
type:0

symbol:   token
value:    c
intValue:28
type:0

symbol:   token
value:    d
intValue:2
type:0

symbol:   token
value:    f
intValue:28
type:0
```

输入：

temp = (400-2)*10#

code = temp-12/3#

bing = (code/10)*temp#

ans = bing

实验三输出的三地址码：

S = 400 - 2

temp = S * 10

A = 12 / 3

code = temp - A

A = code / 10

bing = A * temp

S = bing + 2

ans = S / 4

输出的汇编代码

Mov 400, R0

Sub 2, R0

Mul 10, R0

Mov 12, R1

Div 3, R1

Mov R0, temp

Sub R1, R0

Mov R0, code

Div 10, R0

Mov temp, R2

Mul R2, R0

Mov R0, bing

Add 2, R0

Div 4, R0

Mov R0, ans

2 参考文法（c 语言子集）

1. CompUnit → [CompUnit] (Decl | FuncDef)

2. Decl → ConstDecl | VarDecl

3. ConstDecl → 'const' BType ConstDef { ',' ConstDef } ';'

4. BType → 'int' | 'float'

5. ConstDef → Ident '=' Exp

| Ident '[' [Exp] ']' '=' '{' Exp { ',' Exp } '}'

6. VarDecl → BType VarDef { ',' VarDef } ';'

7. VarDef → Ident

| Ident '[' Exp ']'

| Ident '=' Exp

| Ident '[' [Exp] ']' '=' '{' Exp { ',' Exp } '}'

8. FuncDef → void Ident '(' ')' Block

9. Block → '{' { BlockItem } '}'

10. BlockItem → Decl | Stmt

11. Stmt → LVal '=' Exp ';'

| Ident '(' ')' ';'

| Block

| 'if' '(Cond ')' Stmt ['else' Stmt]

| 'while' '(' Cond ')' Stmt22. | ';'

12. LVal → Ident

| Ident '[' Exp ']'

13. Cond → Exp RelOp Exp

14. RelOp → '==' | '!=' | '<' | '>' | '<=' | '>='

15. Exp → Exp BinOp Exp

| UnaryOp Exp

| '(' Exp ')'

| LVal

| Number

16. Number → FloatConst

| IntConst

17. BinOp → '+' | '−' | '*' | '/' | '%'

18. UnaryOp → '+' | '−'

注：EBNF 中的符号含义：

[...]内包含的为可选项；

{...}内包含的为可重复 0 次至无数次的项。

3 往届优秀代码（仅供学习）

https://github.com/Lincyaw/CompilePriciple

SysY 语言定义（可以参考用来写自己的文法）：

SysY语言定义.pdf

注：代码仅供学生参考设计思想，请勿抄袭！