合肥工業大學

编译原理课程设计报告

设计题目 while循环语句翻译成四元式

学生姓名 丁盛鹏

学　　号 2021214813

专业班级 计算机科学与技术 21-3

指导教师

完成日期 2023.6.25

目录

一、设计目的及设计要求

设计一个语法制导翻译器,将 WHILE 语句翻译成四元式。要求:先确定一个定义 WHILE 语句的文法，为其设计一个语法分析程序，为每条产生式配备一个语义子程序，按照一遍扫描的语法制导翻译方法，实现翻译程序。 对用户输入的任意一个正确的 WHILE 语句，程序将其转换成四元式输出(可按一定格式输出到指定文件中)。

二、开发环境描述

操作系统：WindowsOS

IDE：Microsoft Visual Studio 2019

开发语言：C++

三、设计内容

3.1 词法分析模块

3.1.1 主要功能

1) 程序自动读取文件中 do-while 循环程序
2) 逐行进行字符读取
3) 遇到空格类字符，包括回车、制表符空格跳过，视为无效字符。
4) 读取到的字符拼成单词（关键字、标识符、常数、运算符、关系运算符、分界符号），并用（值，种别）二元式表示，保存至词法分析结果表中。
5) 如果发现错误则报告出错
6) 根据需要是否填写标识符表供以后各阶段使用
7) 识别注释，忽略注释行后内容
8) 判别浮点数和整数，都按照常数处理

单词符号	类别	种别码
"if","while","cout","cin","do","else","main","return","int","enum","char","string","struct","class","float","double","const","case","continue","break","delete","for","false","true","switch","this","new","long","template","static","try","except","public","private","protected"	关键字	0-34

",",";","(",")","{","}","[","]","+","","">","<","=","/","*","|","&","\'",":","%","++","==","+=","-=","/=","*=","--","<<",">>","<=",">=","#"	运算符和界符	35-66
A,b,c,d…	标识符	67
1,2,3…	常量	68
关键字一词一码 0-34，运算符一词一码 35-66，标识符多词一码 67，常量多词一码 68		

表 3.1 字符集

3.1.2 设计思想

1）通过预处理程序将输入串放到输入缓冲区里，并剔除空白字符。

2）程序通过扫描器逐个字符读入输入缓冲区中的字符串，并放在扫描缓冲区中。

3）根据有限自动机的状态转换，将扫描缓冲区中的串识别一个个单词（标识符、常数、界符…）

4）关键字识别需要查找关键字表，识别程序可以与标识符识别程序放在一起，通过查表判断是标识符还是关键字。

5）运算符界符放在同一张表中，识别需要查表。

6）部分运算符如++，--等需要超前搜索，读到第一个字符后再向前读取一个判断是双字符运算符还是单字符运算符。

7）将识别的单词结果表示为二元式（值，种别）形式。

3.1.3 流程设计

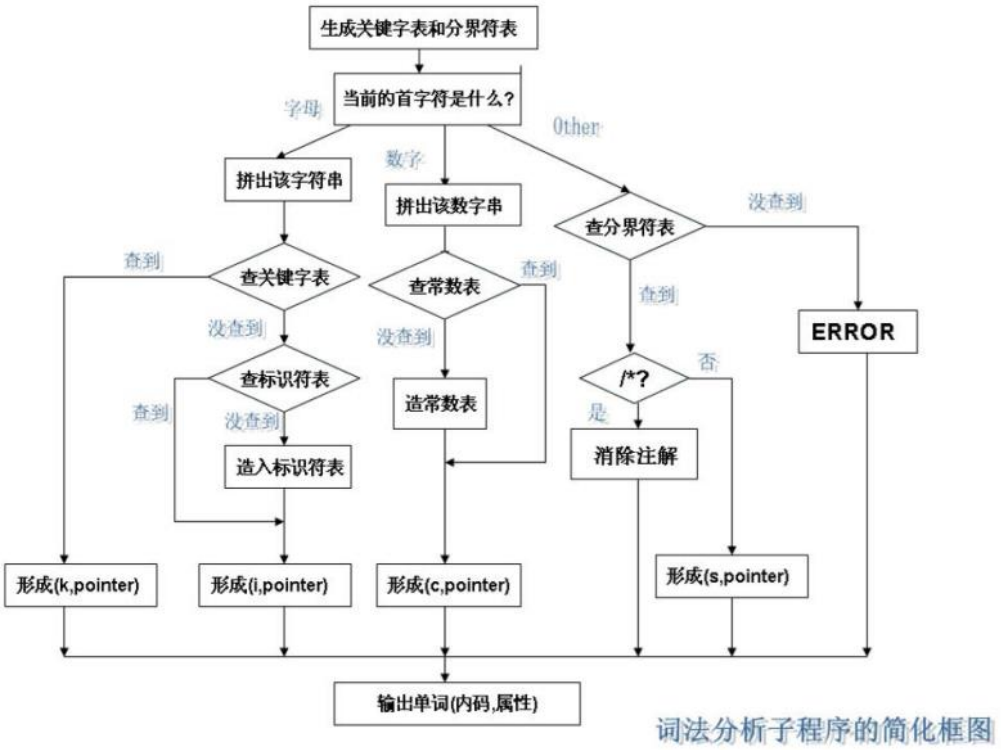

图 3.1

3.1.4 主要算法

词法分析过程算法如下：

```
For input in lines:
    While ch 不是结束符 do:
        向前读取一个字符 ch;
        清除空白字符
        If 标识符（ch 以字母或下划线开头）
            While ch is digit or letter or _
                连接到读取字符串后
                向前读取一个字符
            Check if key
            If yes: 关键字进单词表
            Else :  标识符进单词表
        Else if digit
            While ch is digit or “.”:
                连接到读取字符串后
                向前读取一个字符
                If ch is letter: wrong
                If ch is ”.”: float
        Else if 运算符 or 界符：
                Insertchar()
```

3.1.5 结构设计

单词表示

```
struct word {
    int sym=-1;
    string data = "";
};
```

名称	word	元素	类型	功能
结构	Struct	data	string	保存单词的值
		Sym	int	保存单词的种别码
功能	保存词法分析程序中识别的单词。			

表 3.2 word 结构

变量设计

变量名	类型	功能
lexicalTable	vector<word>	保存词法分析得到的单词
k[35]	String	保存关键字
s[32]	string	保存界符，运算符
pint	int	作为词法分析时的读头指针
strToken	String	保存当前读取到的字符串
ch	Char	当前读取到的字符
input	string	保存要分析的程序中的一行，在程序中按行分析

表 3.3 词法分析变量

函数设计

函数名	返回类型	功能
GetBC()	Void	删除空白字符
IsLetter()	Bool	判断字母
IsDigit()	Bool	判断数字
Reserse()	Int	对 strToekn 中的单词查找关键字表,若找到返回种别码，未找到返回-1
Find_s()	Int	对 ch 中的字符查找运算符表，若找到返回种别码，未找到返回-1
Retract()	Void	搜索指示器回调一个位置，将 ch 置为空白字符
Concat()	Void	将 ch 中字符连接到 strToken 后
InsertId()	Void	插入 strToken 中的标识符到标识符表
InsertConst()	Void	插入常数

Insertchar(int i)	Void	插入运算符
Analyze()	Void	分析输入串
displayLexicalTable()	void	展示分析结果，单词数组
Run()	void	逐行读取文件将字符串送到 Analyse()中分析
getWordType(word w)	Int	判断一个单词的类别并返回。若单词为常数，返回 4 若单词为标识符则返回 3，若单词为运算符则返回 2，若单词为关键字返回 1
printWord(word w)	Void	根据单词类别打印单词
GetChar()	void	向前读取一个字符

表 3.4 词法分析函数

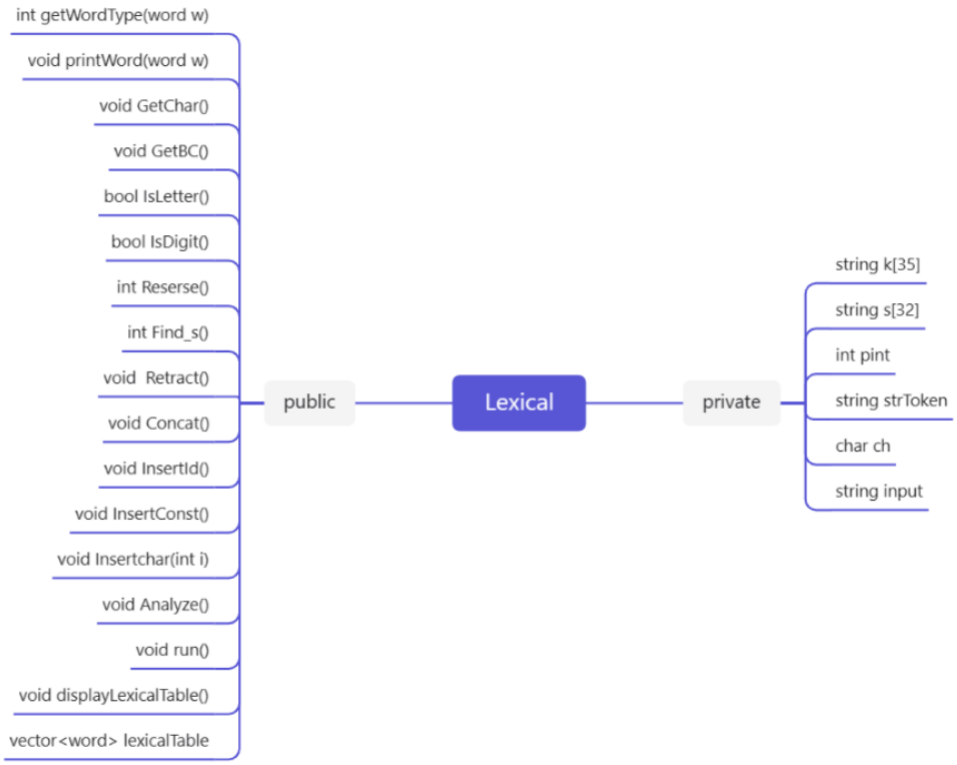

图 3.2 词法分析类结构图

3.2 语法分析与语义分析模块

3.2.1 主要功能

1）自动读取 LR1 语法分析表，进行语法分析。

2）自动填入所给文法的产生式以及终结符集、非终结符集、first 集、follow 集。

3）根据构造的分析表对词法分析产生的结果进行语法分析。

4）采用自下而上的语法分析过程，规约时按照产生式对应的语义规则进行语义计算，进行语义分析。

5）语义分析时生成对应的四元式。

3.2.2 文法设计

1）文法支持逻辑运算、算术运算、赋值操作。文法如下。

```
/*文法
1) A -> do{B}while{C};

2) C -> E ROP E

3) B -> S;B | S; | A

4) S -> i=E

5) E -> -E | F | E+F | E-F | G

6) F -> F*G | F/G

7) G -> i | n | (E)

8) ROP -> > | < | == | >= | <=

*/
```

图 3.3 所用文法

2）其中，A、B、C、E、F、G、ROP、S 为非终结符，do、while、;、=、-、+、*、/、i、n、<、>、<=、>=、(、)、{、}为终结符；i 表示标识符，n 表示常量，ROP 表示逻辑运算符。

3）第一条为 do-while 循环结构，B 为多条语句（S；B）或单条语句（S;）或嵌套另一个 do-while 循环（A）。

4）第四条定义了赋值语句的文法，i 为标识符，E 可以为算数表达式或者常数。

5）文法第五条为算术运算表达式，第六条为乘除运算表达式，乘除运算优先级高于加减运算。

6）文法第七条为标识符和常数产生式，用 i 代表标识符，n 代表常数。同时实现了括号的优先运算。

7）第七条 ROP 产生逻辑运算符，没有优先级的区别，只是简单逻辑表达式在判断循环条件时使用。

根据文法设计对应的属性文法：

```
/*属性文法
  A -> do{B}while{C}; { if C.flag goto do.add else next }
  C -> E1 ROP E2      { C.flag := E1.val ROP.op E2.val }
  B -> S;B            { }
  B -> S;             { }
  B -> A              { }
  S -> i=E            { i.val := E.val }
  E -> -E1            { E.val := E1.val * (-1) }
  E -> F              { E.val := F.val }
  E -> E1+F           { E.val := E1.val + F.val }
  E -> E1-F           { E.val := E1.val - F.val }
  F -> G              { F.val := G.val }
  F -> F1*G           { F.val := F1.val * G.val }
  F -> F1/G           { F.val := F1.val / G.val }
  G -> i              { G.val := i.val }
  G -> n              { G.val := n.val }
  G -> (E)            { G.val := E.val }
  ROP -> >            { ROP.op := > }
  ROP -> <            { ROP.op := < }
  ROP -> ==           { ROP.op := == }
  ROP -> >=           { ROP.op := >= }
  ROP -> <=           { ROP.op := <= }
*/
```

图 3.4 属性文法

8)Do 存在一个属性值为 add,表示 do 后面第一句的地址,C.flag 表示 C 的真假,ROP.op 表示逻辑运算种类，val 属性表示数值，常数是其本身的值，标识符的 Val 是在赋值语句或运算语句中的数值。

9）A->do{B}while{C};的属性文法意思为如果 C 为真则跳转到 do.add（do 后 B 语句即第一条四元式地址），否则跳出（next）。

10)算数运算的属性文法表示在规约时产生式右部计算后的数值赋给产生式左部的文法符号。

3.2.3 设计思想

1）LR1 分析，根据拓广文法生成不同项目，根据项目生成不同项目集，根据历史、展望、现实三方面的符号构建识别活前缀的有限自动机。之后构建 LR1 分析表。

1）语法分析与语义分析共同使用三个栈，符号栈、状态栈、语义栈

2）依照分析表，采用移进-规约思想自下而上分析，将词法分析结果按单词逐个取出，移入符号找，同时将状态更新到状态栈，语义更新到语义栈。

3）遇到规约时，每个产生式对应一个规约函数，函数内进行状态栈的更新并根据属性文法进行语义计算，遇到计算表达式时，调用产生四元式式函数，在函数中完成计算更新到符号表中，并产生四元式。

4）维护一个符号表，语义分析时查找变量的语义值。

5）保存产生的四元式，输出到界面中。

3.2.4 程序流程

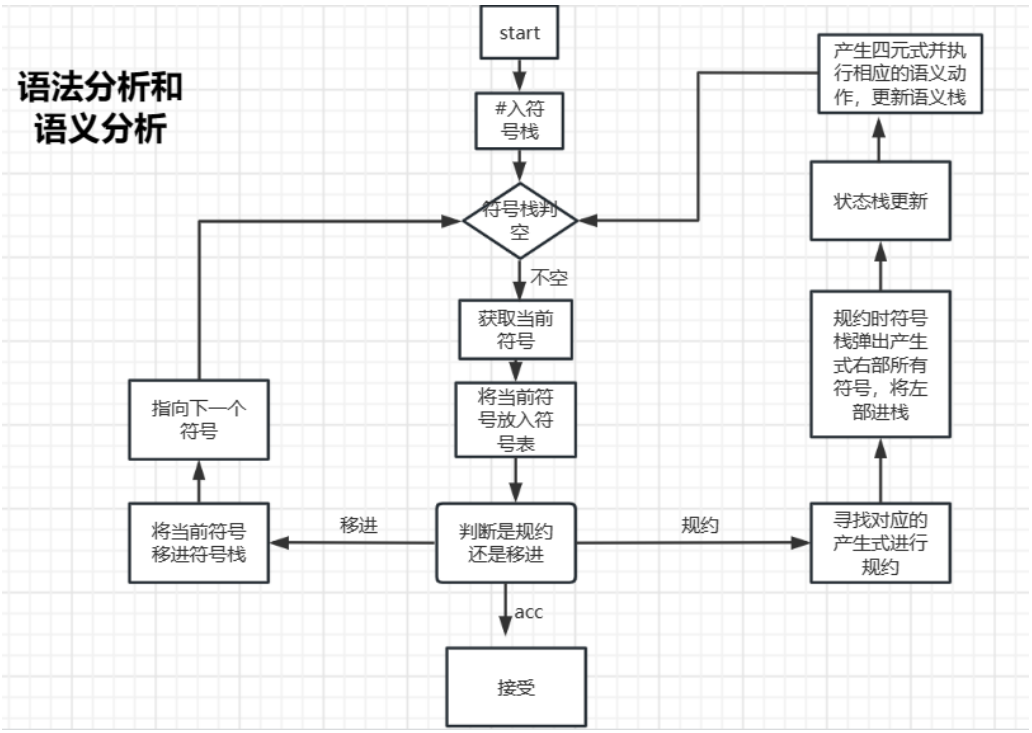

图 3.5 语法分析和语义分析流程图

3.2.5 主要算法

语法分析/语义分析算法如下：

```
While 符号栈不空 do
  If 单词表中单词没有读完
      Curword=当前单词
      Cursymbol=curword.data
      If curword 为标识符
          Cursymbol=”i”;//以 i 终结符代表标识符
          Curword.Data 进语义栈
          curword 形成新的符号 symbol 进符号表
          将 symbol 更新至语义-符号表 semantic_to_symbol
      Else if curword 为常量
          Cursymbol=”n”; //以 n 终结符代表常量
          Curword.Data 进语义栈
          curword 形成新的符号 symbol 进符号表
          将 symbol 更新至语义-符号表 semantic_to_symbol
      Else if curword 为 do
          将 do+doindex 入语义栈
          形成新的符号 symbol 入符号表
          将 symbol 更新至语义-符号表 semantic_to_symbol
          Doindex++;
      Else 不是任何终结符非终结符
```

报错
获取当前状态
查找分析表中的动作
If 是移进
找到的状态入状态栈
cursymbol 入符号栈
指针指向下一个单词
else if 是规约
根据规约的产生式序号转到相应的规约函数
Else if acc
接受
Else 报错

3.2.6 程序结构

产生式表示

```
struct production {
    string left;
    vector<string>right;
};
```

名称	Production	元素	类型	功能
结构	Struct	left	string	产生式左部符号
		right	vector<string>	产生式所有右部
功能	保存文法的产生式			

表 3.5 产生式结构

动作表示

```
struct action {
    string act="-1";
    int gonum=0;
};
```

名称	Action	元素	类型	功能
结构	Struct	act	string	语法分析动作(规约 or 移进)
		gonum	int	跳转到的状态号
功能	保存动作类型及动作后转到的状态。			
解释说明	Act 分为 S：移进；r:规约；acc:接受 若 act=S,Gonum 为移进后转到的状态号；若 act=r，gonum 为规约用到的产生式序号；act 初始值均为-1，gonum 初始值为 0，acc 的 gonum 为 0			

表 3.6 动作结构

符号表示

```
struct symbol {
    string name;//符号名
    string value;//符号值
    int pos;//符号表中的位置
};
```

名称	symbol	元素	类型	功能
结构	Struct	Name	string	符号名
		Value	string	符号语义值
		pos	Int	在符号表中的位置
功能	保存语法/语义分析中的一个符号			
解释说明	Name 表示符号名，若是标识符，name 为标识符名称;若是常数，name 为常数值；若是运算符，name 为运算符本身。 Value 为语义值，标识符的语义值为本身，常量的语义值为其数值，运算符语义值为空。			

表 3.7 symbol 结构

四元式表示

```
struct item {//四元式
    string op;
    int A1index=-1;
    int A2index=-1;
    int resultindex=-1;
};
```

名称	item	元素	类型	功能
结构	Struct	op	string	操作符
		A1index	Int	第一个操作数在符号表中的位置
		A2index	Int	第二个操作数在符号表中的位置
		resultindex	int	结果操作数在符号表中的位置
功能	表示四元式			

表 3.8 item 四元式结构

函数设计(Grammar)

函数名	类型	功能
initial_grammar()	Void	初始化文法，将文法产生式和符号初始化
getVIndex(string V)	int	根据符号返回其标号

表 3.9 Grammar 类方法

变量设计（Grammar）

变量名	类型	功能

all_production	vector<production>	产生式集合
vnSet	set<string>	非终结符集
vtSet	set<string>	终结符集
firstSet	map<string, set<string>>	first 集
followSet	map<string, set<string>>	follow 集

表 3.10 Grammar 类成员

变量设计（Analysis）

变量名	类型	功能
G	Grammar	Grammar 类的对象,用于获取文法及各种符号
L	Lexical	Lexical 类的对象,用于获取词法分析的结果
symTable	vector<symbol>	分析时的符号表
siyuanshiTable	vector<item>	保存生成的四元式
semantic_to_symbol	map<string, int>	根据语义值获取符号在符号表中的位置
statusStack	stack<int>	状态栈
symStack	stack<symbol>	符号栈
semanticStack	stack<string>	语义栈
actionTable[119][20]	action	动作数组
gotoTable[119][9]	Int	Goto 数组
symbolnum	int	生成临时变量 T 时的标号

表 3.11 Analysis 类成员

函数设计（Analysis）

函数名	类型	功能
initialize()	void	初始化，将各变量初始化，栈清空
analyze()	void	进行语法分析和语义分析
initialize_actionTable()	void	加载分析表
display_table()	void	展示分析表
display_siyuanshiTable()	void	展示四元式
GEN(string op, int a1,int a2,int result)	void	根据操作符 op,a1,a2,result 在符号表中的位置产生一个四元式
newType()	symbol	产生一个临时变量 T
A_do_B_while_C()	Void	对 A -> do{B}while{C}；进行规约并执行相应的语义动作，语义分析。
C_E1_ROP_E2()	Void	对 C -> E1 ROP E2 进行规约并执行相应的语义动作
B_S_B()	Void	对 B -> S;B 进行规约并执行相应的语义动作
B_S()	Void	对 B -> S; 进行规约并执行

		相应的语义动作
B_A()	void	对 B -> A 进行规约并执行相应的语义动作
S_i_E()	void	对 S -> i=E 进行规约并执行相应的语义动作
E_neg_E1()	void	对 E -> -E1 进行规约并执行相应的语义动作
E_F()	Void	对 E -> F 进行规约并执行相应的语义动作
E_E1_add_F()	Void	对 E -> E1+F 进行规约并执行相应的语义动作
E_E1_minus_F()	Void	对 E -> E1-F 进行规约并执行相应的语义动作
F_G()	Void	对 F -> G 进行规约并执行相应的语义动作
F_F1_mul_G()	Void	对 F -> F1*G 进行规约并执行相应的语义动作
F_F1_div_G()	Void	对 F -> F1/G 进行规约并执行相应的语义动作
G_i()	Void	对 G -> i 进行规约并执行相应的语义动作
G_n()	void	对 G -> n 进行规约并执行相应的语义动作
G_E()	void	对 G -> (E) 进行规约并执行相应的语义动作
ROP_larger ()	void	对 ROP -> >进行规约并执行相应的语义动作
ROP_lower()	void	对 ROP -> <进行规约并执行相应的语义动作
ROP_logical_equal ()	void	对 ROP -> ==进行规约并执行相应的语义动作
ROP_larger_equal()	void	对 ROP -> >=进行规约并执行相应的语义动作
ROP_lower_equal()	void	对 ROP -> <=进行规约并执行相应的语义动作

表 3.12 语法语义分析函数

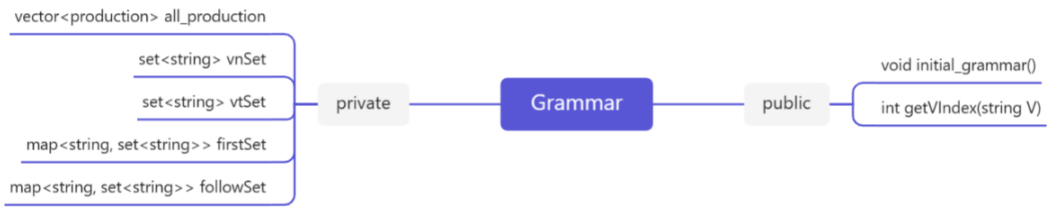

图 3.6 Grammar 类结构

图 3.7 Analysis 类结构

3.3 整体设计

3.3.1 整体设计思路

1）将词法分析作为独立的阶段，在进行语法分析前执行词法分析。

2）语法分析需要用到词法分析结果，故词法分析使用一个表格将结果保存。然后再语法分析中声明一个词法分析类的对象，使其进行词法分析过程，将源程序翻译为单词二元组形式。如此将词法分析结果传递到语法分析过程中。

3）语法分析和语义分析通常是在一起，采用 LR1 自下而上的语法分析，在规约产生式时进行语义计算。将每个产生式规约过程设计成一个函数，在函数中进行规约、语义计算及四元式生成。

4）并不是每个规约进行时都生成四元式，四元式生成只在赋值语句及算术运算中进行。每个赋值及算数运算过程都涉及语义计算，而其他产生式仅涉及符号与语义的更新连接（如 G->i 规约时将 i 对应的语义连接到 G 上，所以将语义计算过程放在四元式生成函数中。

```
void Analysis::GEN(string op, int Alindex,int A2index, int resultindex) {
    item new_item;
    if (op == "neg") {
        symTable[resultindex].value = "-" + symTable[Alindex].value;
    }
    if (op == "+") {
        symTable[resultindex].value = to_string(stoi(symTable[Alindex].value) + stoi(symTable[A2index].value));
    }
    else if (op == "-") {
        symTable[resultindex].value = to_string(stoi(symTable[Alindex].value) - stoi(symTable[A2index].value));
    }
    else if (op == "*") {
        symTable[resultindex].value = to_string(stoi(symTable[Alindex].value) * stoi(symTable[A2index].value));
    }
```

图 3.8 四元式产生函数

5)语法分析与语义分析共同维护一个符号表，在产生新变量或变量赋值时更新。语法分析主要在读入单词时添加到符号表中，供语义分析过程使用符号值。

3.3.2 整体分析流程

1）预先设计好文法，将文法在程序启动时通过调用函数或自动进行产生式写入。

2）程序启动时自行进行 VT、VN、First、follow 集写入相应存储结构。

3）将预先构造好的 LR1 分析表按一定格式保存在文件中，在程序启动时读取文件进行 LR1 分析表的读入，读入的分析表放入特定的存储结构中。

4）作为测试代码，事先将需要分析的 do-while 循环程序代码写入指定文本文件。

5）程序按行读取 do-while 文件，送入词法分析程序中分析。

6）词法分析程序将文件翻译为 word 二元组，将词法分析结果传到语法分析过程中。

7）程序根据构造好的分析表以及词法分析得到的结果，进行语法分析，执行移进或规约动作。

8）在规约产生式时进行语义计算，并产生四元式。

9）程序最后输出四元式。

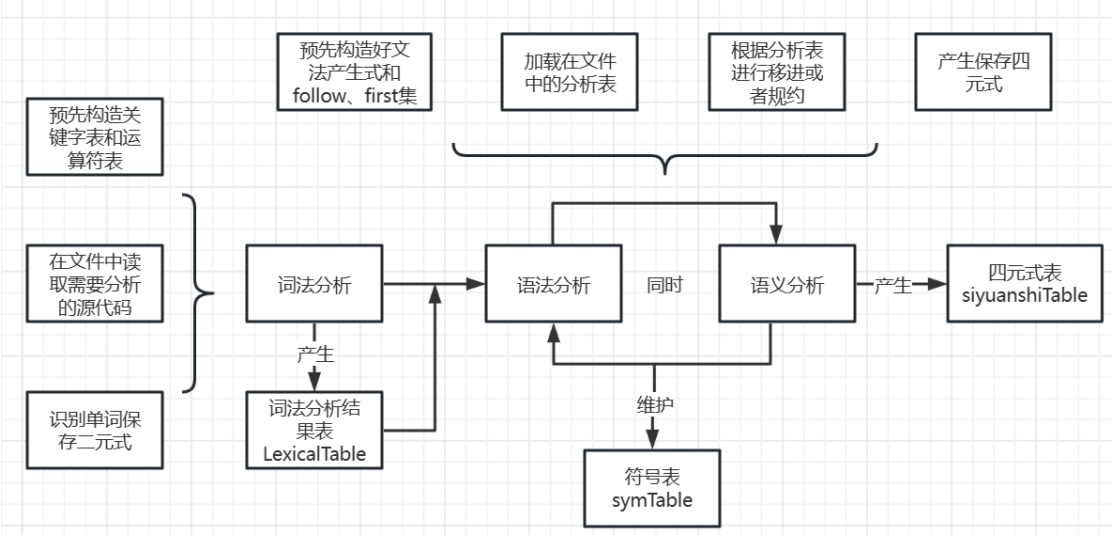

图 3.9 整体程序流程图

四、用户指南

4.1 程序功能

实现输入 do-while 语句源程序翻译为四元式代码 并显示符号表。

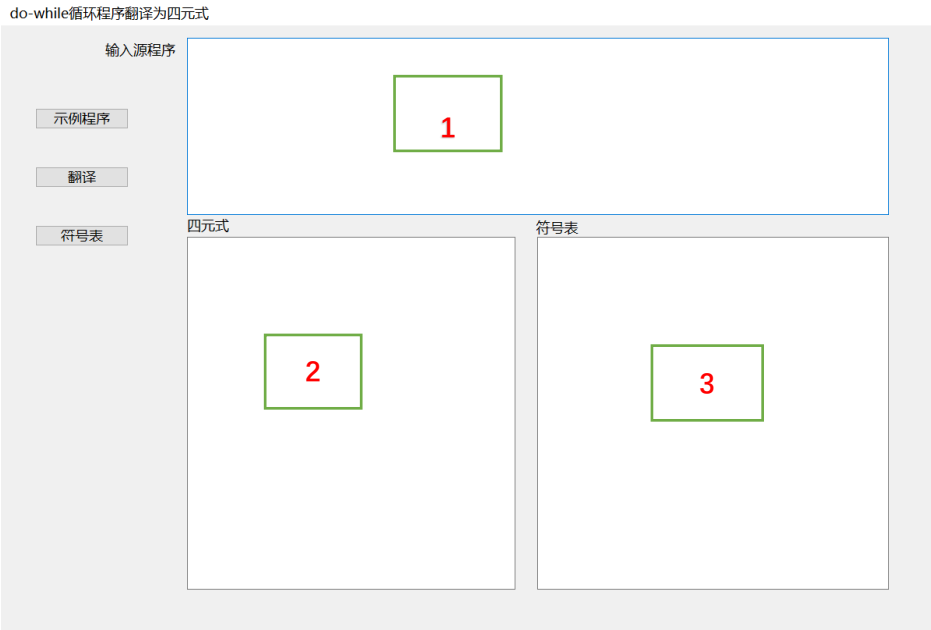

图 4.1 界面

4.2 使用方法

1）在“输入源程序”文本框（图中标号 1 显示区域）内输入基于 c++语法的 do-while 程序。或者点击左部“示例程序”按键，使用程序提供的示例程序。

2）点击左部“翻译”按键，将会在“四元式”文本框（图中标号 2 所示区域）内

显示翻译后的四元式语句。

3）点击左部“符号表”按键，将会在“符号表”文本框（图中标号 3 所示区域）内显示语法/语义分析过程中程序维护的符号表。

五、运行结果与分析

5.1 输入输出格式

5.1.1 输入格式：

1)C++语法，do-while 循环语句.

2)支持变量赋值，算数逻辑运算.

3)每条语句结束符为分号，程序结束符为井号.

4)支持单行注释，注释符为//.

5.1.2 输出格式：

1)序号 四元式（操作符，第一个操作数，第二个操作数，结果操作数）

2)操作数不存在输出在四元式内表示为-1

3）ui 界面顺序输出四元式即程序分析中所维护的符号表。符号表格式为：

(Name value position)

5.2 测试结果

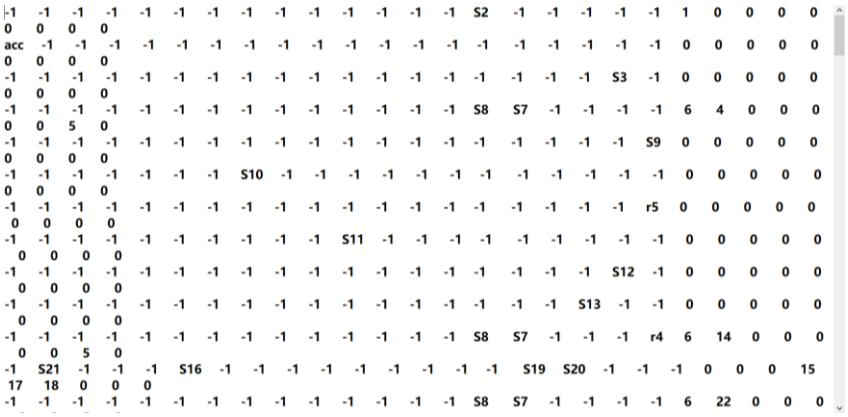

图 5.1 分析表

5.2.1 测试用例

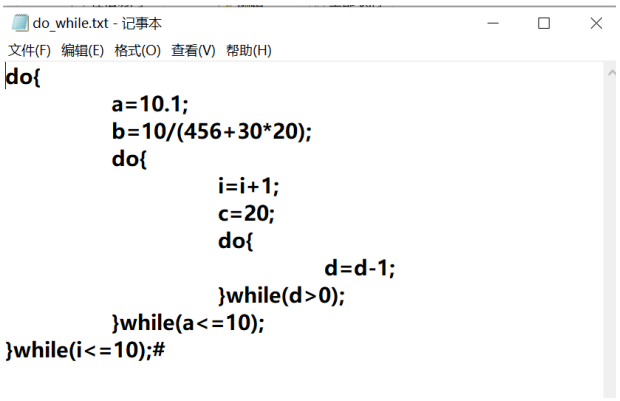

图 5.2 文件格式

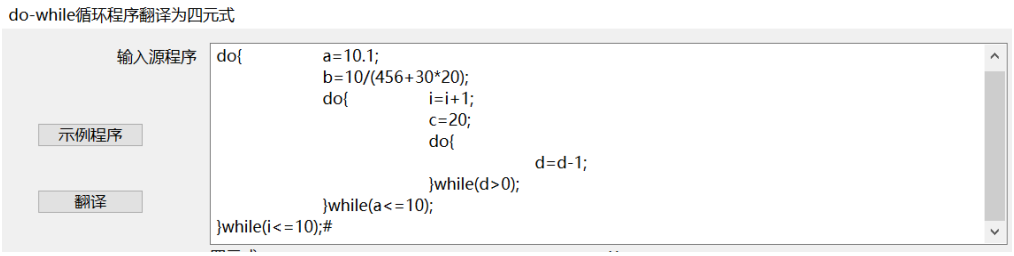

图 5.3 ui 界面格式

测试用例解释与分析：三重嵌套 do-while 循环程序。最外层对标识符 a 进行赋值，使用浮点数对其进行赋值，检测其识别整数和浮点数的能力。对 b 进行赋值，赋值号左边是运算表达式，检测进行加减乘除运算的能力。第二层循环中对 i，c 进行赋值,此均是标识符。第三层对 d 变量进行循环判断，并在循环体中自减，检测逻辑表达式的识别运算能力。

5.2.2 测试结果

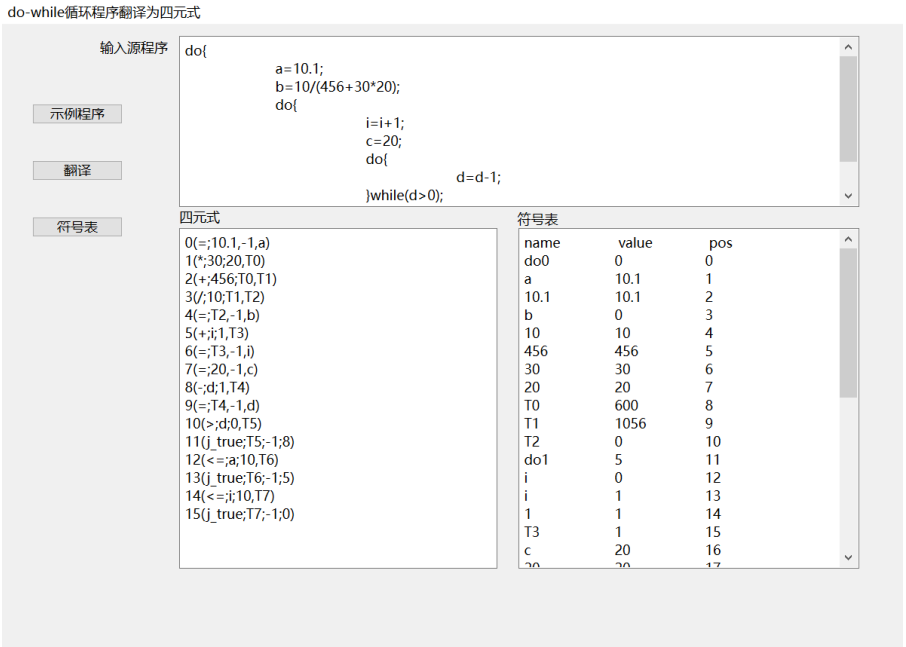

图 5.4 ui 界面输出

解释与分析：输入源程序后，左边为翻译成四元式的形式，右边为程序中维护的符号表。正确输出四元式，可以看到使用四元式正确表示第一条语句 a=10.1；浮点数识别无误，while 循环结束后 jmp 到 do 处四元式标号，三个循环产生三个 jmp 四元式。

5.3 运行过程分析

```
int main() {
    Analysis a;
    a.initialize();
    a.initialize_actionTable();
    a.display_table();
    a.analyze();
    a.display_siyuanshiTable();
}
```

图 5.5 main()函数

下面分析从程序开始到第一条四元式产生的运行过程

1）程序运行先构造 Analysis 类对象 a,a 中有公有成员 Lexical L 和 Grammar G。L 生成时构造好关键字表 k 和运算符表 s。G 生成时构造好文法产生式，终结符集，非终结符集，first 集以及 follow 集。

2）运行 a,initialize（）,初始化对象 a 中的各类变量。

3)a.initialize_actionTable(),将文件中的分析表读入程序中 actionTable 和 gotoTable。

4)a.display_table（），展示读入的分析表。

5）a.analyse 开始 LR1 语法分析及语义分析。此处为整个程序的核心部分。

a)首先#入符号栈作为初始符号，0 入状态栈作为初始状态。声明一个词法分析类对象 L 并执行 L.run()生成词法分析单词表。

b)进入循环，只要符号栈不空，持续进行语法分析动作。

c)单词指针指向单词表第一个单词 do，让新变量 cursymbol 等于当前单词的 data 即为 do。判断当前单词 do 的类型，进入 do 的分支语句，将这个新单词更新至语义栈和符号表；

L.lexicalTable	{ size=63 }
[capacity]	63
[allocator]	allocator
[0]	{sym=4 data="do" }
[1]	{sym=39 data="{" }
[2]	{sym=67 data="a" }
[3]	{sym=47 data="=" }
[4]	{sym=68 data="10.1" }
[5]	{sym=36 data=";" }
[6]	{sym=67 data="b" }
[7]	{sym=47 data="=" }
[8]	{sym=68 data="10" }
[9]	{sym=48 data="/" }
[10]	{sym=37 data="(" }
[11]	{sym=68 data="456" }
[12]	{sym=43 data="+" }
[13]	{sym=68 data="30" }
[14]	{sym=49 data="*" }
[15]	{sym=68 data="20" }

图 5.6 单词表

首先生成一个 symbo 变量 s 用来保存符号；Do 作为特殊的单词符号，需要在 while 判断时返回到此处，故 s.value 语义值应该为当前四元式的序号；s.pos 为在符号表中的位置为 symTable.size()；由于要区分多重 do-while 循环嵌套，故 do.name 为 do+doindex，此时 doindex=0，执行完此句后 doindex++；将 s 推入符号表中；更新语义-符号表 semantic_to_symbol[s.name] = s.pos，表示 s.name 语义的符号在符号表的 s.pos 处；此时并不将 do 入符号栈，因为此时只是检测到了 do 这个单词，是移进还是规约并不确定，还需向下进行来判断是否将其入栈，但无论移进还是规约，下一次符号表中被移进的总是这个符号，故先入符号表再判断下一步是移进还是规约。之后获取当前状态，根据状态和当前符号在分析表中寻找动作.如果是移进，将要转到的状态移进状态栈，当前符号名 cursymbol 移进符号栈。当前符号为 do，故移进 do 入符号栈，状态 2 入状态栈；指向下一个单词。

c	{ size=2 }
[allocator]	allocator
[0]	0
[1]	2
[原始视图]	{_Mypair=allocator }
[原始视图]	{c={ size=2 } }
symStack	{ size=2 }
c	{ size=2 }
[allocator]	allocator
[0]	{name="#" value="" pos=0 }
[1]	{name="do" value="" pos=0 }
[原始视图]	{_Mypair=allocator }

状态栈

符号栈

图 5.7 读完“do”后栈情况

d)下一个单词为{，既不是常量、标识符、do，故直接转到获取当前状态并查找动作为 S3，将 3 入状态栈，{入符号栈，指向下一个单词。

e)下一个单词为 a,判断其为标识符，令 cursymbol=i，用来统一代表标识符；标识符的语义为其本身，语义栈推入 a；生成符号 s,s.name=a,此处只是扫描到标识符，但还未赋值，赋值需要赋值语句在规约时赋值，故 s.value 设置初值 0，s.pos 为 symTable.Size()；符号表推入 s,更新语义-符号表。获取当前状态，查找分析表 S7，7 入栈，cursymbol=i 入栈，指向下一个单词。

f)下一个为=，直接查询分析表得到动作 S11，移进后指向下一个单词

g)下一个为常数 10.1，cursymbol=n 统一代表常量；语义栈入 10.1；生成新变量 s，s.name=10.1,s.value=10.1,s.pos=symTable.size();s 入符号表，更新语义-符号表；查询分析表 S20，入栈后指向下一个单词

h)下一个为；，直接查询分析表 r15,以 15 号产生式 G->n 规约;将符号栈栈顶出栈，此时符号栈#，do,I,=,n,出栈后栈顶为=；同时状态栈出栈，栈顶为 11；将产生式左部 G 入符号栈，状态栈入 11 号状态遇到 G 时的状态为 18；将 n 对应的 10.1 语义在符号表中的位置赋给 G 的 pos,即将 n 的语义关联到 G。

```
void Analysis::G_n() {
    symStack.pop();  已用时间 <= 1ms
    statusStack.pop();
    symStack.push(symbol{ "G" });
    int curstatus = statusStack.top();
    statusStack.push(gotoTable[curstatus][5]);
    symStack.top().pos = semantic_to_symbol[semanticStack.top()];
}
```

n 出符号栈，对应状态出栈，G 进符号栈，对应状态进栈，语义关联

图 5.8 G->n 的规约函数

i)规约结束后当前符号仍为；，直接到分析表中查询为 r11;以 11 号产生式规约 F->G，与上一次规约类似，不再赘述。

j)符号仍为；，查找分析表为 r8，以 E->F 规约，与上一层类似。

k)符号不变，查找分析表为 r6，以 S->i=E 规约；符号栈将产生式右部所有符号出栈，在出栈过程中，定义 symbol 类型变量保存 i 和 E，同时状态栈对应状态出栈；出栈完成后将 S 推入栈中， 栈顶状态遇到 S 时跳转的状态推入状态栈，此时栈顶为 5；相应的语义出语义栈，由于赋值语句在产生四元式时需要结果操作数在符号表中的位置，故语义栈出栈时保存赋值号左部符号 i 的语义 a，使用语义-符号表查找其在符号表中的位置。调用函数 GEN("=", E.pos, -1, semantic_to_symbol[i.name])产生四元式;函数中将符号表 A1index 处的值赋给 resultindex 处的值，即将常数 10.1 作为值赋给符号 a。同时产生一条四元式推入四元式表中。

六、主要问题及其解决

1）词法分析过程中，起初并不能识别浮点数，遇到浮点数会错误识别。

解决：在数字识别过程中加入小数点的判断。这样能识别正确的小数。

2)加入识别小数的点的判断可以识别正确的小数,但遇到 1.2.1 这样错误的小数无法报错。

解决：在程序中加入判断小数点出现次数的标志，若出现第二次报错，跳过此数字。

3）源程序读入识别时使用 fstream 不能全部读入，因为遇到空格会自动停止。

解决：使用 istringstream 对象；可将空格一并读入字符串，在词法分析时通过 getBC()可跳过空格。

4）语义分析程序中，生成新变量时需要使用变量下标加以区别。

解决：在 Analysis 类中定义一个成员 symbolnum，每次生成一个变量后加一。T0，T1…

5）起初在 G->i，F->G 此类产生式规约后，由于右部需要出栈，左部入符号栈时只有 name 属性，其语义及在符号表中的位置未知，在后续生成四元式并进行语义计算时无法获取部分信息。

解决：G->I,F->G 规约时将 i 在符号表中的位置传递给 G.pos 后再将 G 入栈。F->G 时同样。之后在语义计算或生成四元式时可以获取到这个标识符的信息。即执行{ G.val := i.val }，但传递的并不是语义值，而是所代表符号在符号表中的位置。

七、设计改进

1）语法分析程序中起初四元式的生成使用的是符号在符号表中的位置，生成时在符号表中查找符号。在调试以及结果查看时较为不便，故在输出时将其改为普通四元式输出，源文件修改时四元式结果可以清楚看到改变的位置。

2）在输入输出展示上改进，之前使用黑色输出窗口输出，改进为使用用户友好的图形化界面。可在界面上方输入要翻译的源程序，也可点击“示例程序”翻译预备的示例程序。点击翻译显示四元式，点击符号表显示翻译过程中维护的符号表。

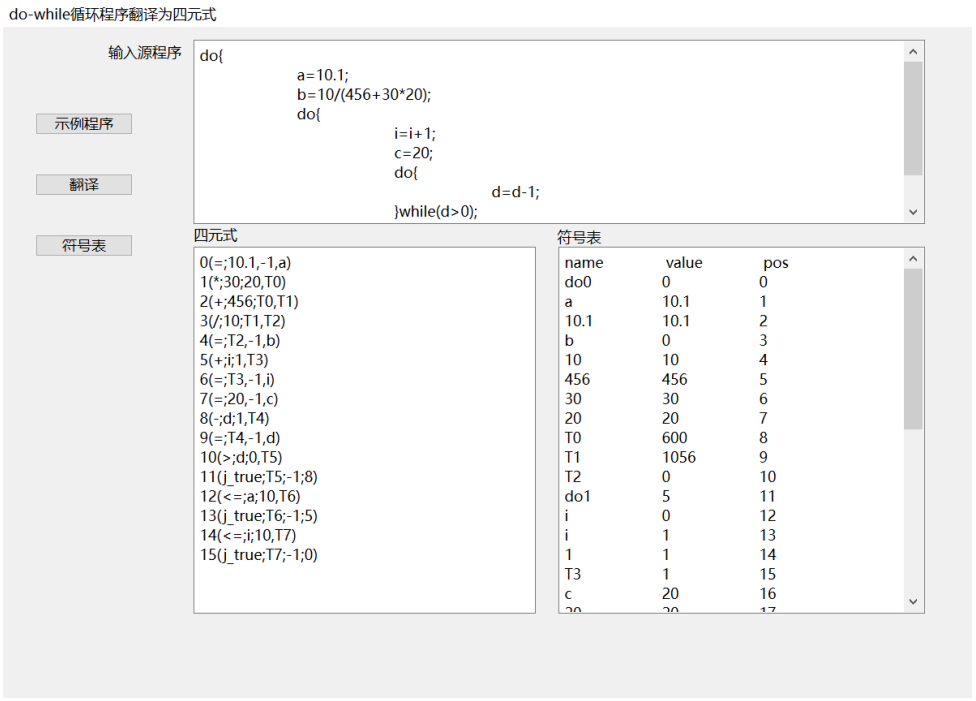

图 7.1 改进后的 ui 界面

八、设计总结

1)设计实现了词法分析器，语法分析模块，语义分析模块，能够完成对 c++程序 do-while 循环语句的基本识别与翻译成中间代码形式，完成了设计内容。

2）词法分析模块能识别 c++大部分单词，但语法分析模块对 c++语法识别主要围绕 do-while 语句，如 do-while 语句的语法，赋值语句的语法，比较语句的语法等。

3）文法设计时仍有不足，变量声明语句（int I;）并未包含，while 循环中的标识符均应为外部已定义过的标识符。

4）设计的文法较为庞大，故事先构建好 LR1 项目集与分析表，使用第三方软件并采用文件读入的方式获取分析表。之后可实现根据文法自动构建项目集以及分析表，减少工作量。

5）设计中整合了词法分析实验，LR1 语法分析实验中的内容，将两者结合起来，对两者知识体系进行了复习巩固，又进行拓展，添加实现了语义分析模块，综合运用了所学知识。同时对编译过程加深了理解，如词法分析、语法、语义分析过程的连接，编译过程各模块的作用等。

6）程序结果显示运用了 qt 实现 ui 界面，方便使用的同时学习到了 qt 图形化编程的方法。

附录

源代码

由于代码文件较多，另附文件