哈尔滨工业大学(深圳)

《编译原理》实验报告

学　院: 计算机科学与技术

姓　名: 谭泽玖

学　号: 190110102

专　业: 计算机科学与技术

日　期: 2021-11-26

1 实验目的与方法

1) 实验环境

语言	C++
编译器	MinGW
IDE	Visual Studio
开发环境	Windows10

2) 词法分析器

目的：加深对词法分析程序的功能及实现方法的理解。加深对类 C 语言的文法的认识，理解自动机、编码表和符号表在编译过程中的应用。

方法：采用正则文法，有限自动机。绘制状态转化图并合并，为文法符号制定编码，使用符号表记录符号信息，按照读入顺序输出 token 串。

3) 语法分析

目的：深入了解语法分析程序实现原理及文法。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。

方法：LR(1)分析法。定义描述程序设计语言语法的文法，并编写拓广文法，通过编译工作台生成 LR 分析表，编写 LR 主程序完成 LR 分析表移进、归约、出错和接受四个动作。

4) 典型语句和语义分析及中间代码生成

目的：加深对自底向上语法制导翻译技术的理解与掌握，巩固对语义分析的基本功能和原理的认识，理解生成中间代码的作用。

方法：对实验 2 中的 LR(1)分析法加以改动，采用自底向上的分析法。

5) 目标代码生成

目的：加深对编译器总体结构的理解与掌握。加深对汇编指令的理解与掌握。

对指令选择、寄存器分配和计算顺序有较深的理解。

方法：采用目标代码生成算法，生成汇编代码。

2 实验内容及要求

1） 词法分析器

内容：输入以文件形式存储的单词代码文件，输出以文件形式存储的 Token 串，简单符号表。

要求：实现词法分析器的基本功能包括：标识符、关键字、常数、运算符合分界符。

2） 语法分析

内容：输入由词法分析器生成的 Token 串，简单符号表，进行语法分析。输出推导过程中使用的产生式序列。

要求：构造所给文法的 LR(1)分析表，设计分析表的存储结构，保留单词属性值。

3） 典型语义分析及中间代码生成

内容：输入由词法分析器生成的 Token 串，符号表，语法分析器生成的产生式序列，输出四元码形式的中间代码。在实验二的基础上，为归约式添加相应的动作

要求：在实验一符号表的基础上拓展符号表，将名字的属性填入到符号表中。能够为基本的算术运算生成中间代码。

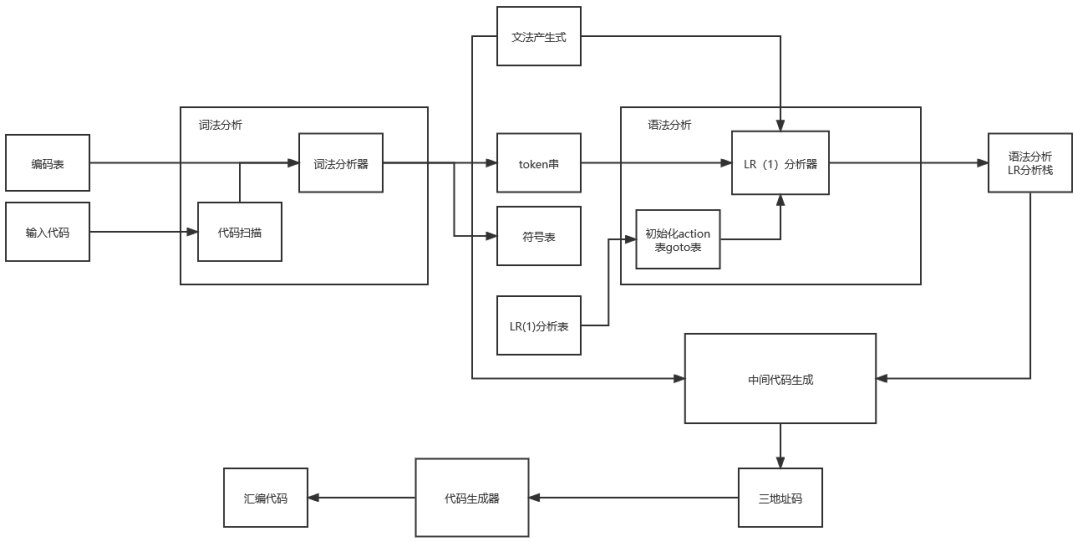
4） 目标代码生成

内容：输入由语法分析器生成的 Token 串，符号表，语法分析器生成的产生式序列，由中间代码生成器生成的中间代码，输出 x86 形式的汇编代码。

要求：能够翻译基本的赋值语句。

3 实验总体流程与函数功能描述

1） 流程



2） 函数功能描述

1. `void code_reader();`

a） 功能：用于读入文件中待编译代码并以字符形式存储在字符数组中

b） 代码

```
void code_reader()
{
    int i = 0;
    FILE* fp = NULL;
    fp = fopen("prefiles/input_code.txt", "r");
    if ((fp = fopen("prefiles/input_code.txt", "r")) == NULL)
    {
        printf("file cannot open!");
        exit(1);
    }
    //开始读文件，并把文件input的内容存进memory数组中
    char str = fgetc(fp);
    while (str != EOF)
    {
        memory[i] = str;
        str = fgetc(fp);
        i++;
    }
    memory[i] = '\0';
```

```
    fclose(fp);

}
```

2. `void init_production();`

   a) 功能：初始化文法产生式表用于后续编译

   b) 代码

```
void init_production()
{
    char ch = '\0';
    FILE* fp1 = NULL;
    //fp1 = fopen("prefiles/production.txt", "r");
    if ((fp1 = fopen("prefiles/production.txt", "r")) == NULL)
    {
        printf("file production cannot open!");
        exit(1);
    }
    int pro_i;
    for (pro_i = 1; ch != EOF; pro_i++)
    {
        int j = 0;
        pro[pro_i].left_code = pro_i;
        ch = fgetc(fp1);
        while (ch != '\n' && ch != EOF)
        {
            pro[pro_i].production[j++] = ch;
            ch = fgetc(fp1);
        }
    }
    fclose(fp1);

}
```

3. `void init_action();`

   a) 功能：初始化动作表用于后续编译

   b) 代码

```
void init_action()
{
	int i = 0;
	//首先进行lr表的读取
	FILE* fp = NULL;
	fp = fopen("prefiles/LR1.csv", "r");
	if ((fp = fopen("prefiles/LR1.csv", "r")) == NULL)
	{
		printf("file LR1.csv cannot open!");
		exit(1);
	}
	char ch = fgetc(fp);
	while (ch != '\n')
		ch = fgetc(fp);
	ch = fgetc(fp);
	int count = 0;
	int rem_count = 0;
	int jug = 0;
	while (ch != '\n')   //一行一行读取lr表
	{
		if (ch == ',')
		{
			count++;
			i = 0;
			ch = fgetc(fp);
		}
		else if (ch == '$')
		{
			state[count].action[0] = '$';
			jug = 1;
			ch = fgetc(fp);
			rem_count = count;
			count = 0;
		}
		else if (jug == 0)        //action表
		{
			while (ch != ',')
			{
				state[count].action[i++] = ch;
				ch = fgetc(fp);
			}
		}
		else if (jug == 1)        //goto表
		{
```

```
            while (ch != ',' && ch != '\n')
            {
                state[count].go[i++] = ch;
                ch = fgetc(fp);
            }
        }
    }
    int goto_count = count;
    int act_i = 0;
    int old_state;
    int num[2];
    int j = 0;
    while (ch != EOF)
    {
        ch = fgetc(fp);
        j = 0;
        num[0] = 0;
        num[1] = -1;
        if (ch == EOF)
            break;
        while (ch != ',')
        {
            num[j] = ch - '0';
            j++;
            ch = fgetc(fp);
        }
        if (num[1] == -1)
            old_state = num[0];
        else if (num[1] != -1)
            old_state = num[0] * 10 + num[1];
        count = 0;
        while (ch != '\n' && ch != EOF)
        {
            if (ch == ',')
            {
                count++;
                ch = fgetc(fp);
            }
            else
            {
                if (ch == 's')       //读到shift
                {
                    action[act_i].old_state = old_state;
                    action[act_i].action = ch;
```

```
            while (ch != ' ')
            {
                ch = fgetc(fp);
            }
            ch = fgetc(fp);
            j = 0;
            num[0] = 0;
            num[1] = -1;
            while (ch != ',')
            {
                num[j] = ch - '0';
                j++;
                ch = fgetc(fp);
            }
            if (num[1] == -1)
                i = num[0];
            else if (num[1] != -1)
                i = num[0] * 10 + num[1];
            action[act_i].new_state = i;
            i = 0;
            while (state[count].action[i] != '\0')
            {
                action[act_i].cause[i] = state[count].action[i];
                i++;
            }
            action[act_i++].PNO = 0;
            count++;
            ch = fgetc(fp);
        }
        else if (ch == 'r')  //读到reduce
        {
            action[act_i].old_state = old_state;
            action[act_i].action = ch;
            while (ch != ' ')
            {
                ch = fgetc(fp);
            }
            ch = fgetc(fp);
            j = 0;
            num[0] = 0;
            num[1] = -1;
            while (ch != ',')
            {
                num[j] = ch - '0';
```

```
                    j++;
                    ch = fgetc(fp);
                }
                if (num[1] == -1)
                    i = num[0];
                else if (num[1] != -1)
                    i = num[0] * 10 + num[1];
                action[act_i].PNO = i;
                action[act_i].new_state = -1;
                i = 0;
                while (pro[action[act_i].PNO].production[i] != '-' &&
pro[action[act_i].PNO].production[i] != ' ')
                {
                    action[act_i].left[i] =
pro[action[act_i].PNO].production[i];
                    i++;
                }
                j = 0;
                while (pro[action[act_i].PNO].production[i] == ' ' ||
pro[action[act_i].PNO].production[i] == '-' || pro[action[act_i].PNO].production[i]
== '>')
                    i++;
                while (pro[action[act_i].PNO].production[i] != '\0')
                {
                    action[act_i].right[j] =
pro[action[act_i].PNO].production[i];
                    i++;
                    j++;
                }
                i = 0;
                while (state[count].action[i] != '\0')
                {
                    action[act_i].cause[i] = state[count].action[i];
                    i++;
                }
                act_i++;
                count++;
                ch = fgetc(fp);
            }
            else if (ch == 'a')  //读到accept
            {
                action[act_i].action = ch;
                while (ch != ',')
                {
```

```
                    ch = fgetc(fp);
                }
                action[act_i].old_state = old_state;
                action[act_i].new_state = -2;
                i = 0;
                while (state[count].action[i] != '\0')
                {
                    action[act_i].cause[i] = state[count].action[i];
                    i++;
                }
                count++;
                act_i++;
            }
            else if (isdigit(ch))
            {
                j = 0;
                num[0] = 0;
                num[1] = -1;
                while (ch != ',' && ch != '\n')
                {
                    num[j] = ch - '0';
                    j++;
                    ch = fgetc(fp);
                }
                if (num[1] == -1)
                    i = num[0];
                else if (num[1] != -1)
                    i = num[0] * 10 + num[1];
                action[act_i].old_state = old_state;
                action[act_i].new_state = i;
                i = 0;
                while (state[count - rem_count].go[i] != '\0')
                {
                    action[act_i].cause[i] = state[count -
rem_count].go[i];
                    i++;
                }
                count++;
                action[act_i++].PNO = -1;
                if (ch != '\n')
                    ch = fgetc(fp);
            }
            else//读入lr错误
            {
```

```
                        printf("lr error!");
                        exit(0);
                    }
                }
            }
        }

}
```

4. `void Lexcial_analyser();`

   a) 功能：词法分析器

   b) 描述：根据编码表逐字符读入输入代码，同时与编码表进行查找匹配并输出 token 串

   c) 代码

```
void code_reader()
{
    int i = 0;
    FILE* fp = NULL;
    fp = fopen("prefiles/input_code.txt", "r");
    if ((fp = fopen("prefiles/input_code.txt", "r")) == NULL)
    {
        printf("file cannot open!");
        exit(1);
    }
    //开始读文件，并把文件input的内容存进memory数组中
    char str = fgetc(fp);
    while (str != EOF)
    {
        memory[i] = str;
        str = fgetc(fp);
        i++;
    }
    memory[i] = '\0';
    fclose(fp);
}
void Lexcial_analyser()
{
    int symbol_jug = 0;
    int value_i;
    int i = 0;
    //下面是词法分析的内容
```

```
    while (memory[i] != '\0')
    {
        if (isalpha(memory[i]) != 0)   //不是字母
        {
            value_i = 0;
            while (isalpha(memory[i]) != 0 || isdigit(memory[i]) != 0)
            {
                token[token_i].value[value_i] = memory[i];
                i++;
                value_i++;
            }
            //关键字
            if (token[token_i].value[0] == 'i' &&
token[token_i].value[1] == 'n' && token[token_i].value[2] == 't')
{ token[token_i].typecode = 5; token_i++; }      //int->5
            else if (token[token_i].value[0] == 'i' &&
token[token_i].value[1] == 'f') { token[token_i].typecode = 6;
token_i++; }      //if->6
            else if (token[token_i].value[0] == 'e' &&
token[token_i].value[1] == 'l' && token[token_i].value[2] == 's' &&
token[token_i].value[3] == 'e') { token[token_i].typecode = 7;
token_i++; } //else->7
            else if (token[token_i].value[0] == 'c' &&
token[token_i].value[1] == 'h' && token[token_i].value[2] == 'a' &&
token[token_i].value[3] == 'r') { token[token_i].typecode = 8;
token_i++; } //char->8
            else if (token[token_i].value[0] == 'l' &&
token[token_i].value[1] == 'o' && token[token_i].value[2] == 'n' &&
token[token_i].value[3] == 'g') { token[token_i].typecode = 9;
token_i++; } //long->9
            else if (token[token_i].value[0] == 's' &&
token[token_i].value[1] == 'h' && token[token_i].value[2] == 'o' &&
token[token_i].value[3] == 'r' && token[token_i].value[4] == 't')
{ token[token_i].typecode = 10; token_i++; }//short->10
            else if (token[token_i].value[0] == 'f' &&
token[token_i].value[1] == 'l' && token[token_i].value[2] == 'o' &&
token[token_i].value[3] == 'a' && token[token_i].value[4] == 't')
{ token[token_i].typecode = 11; token_i++; }//float->11
            else if (token[token_i].value[0] == 'd' &&
token[token_i].value[1] == 'o') { token[token_i].typecode = 12;
token_i++; } //do->12
            else if (token[token_i].value[0] == 's' &&
token[token_i].value[1] == 'w' && token[token_i].value[2] == 'i' &&
token[token_i].value[3] == 't' && token[token_i].value[4] == 'c' &&
```

```
token[token_i].value[5] == 'h') { token[token_i].typecode = 13;
token_i++; } //switch->13
            else if (token[token_i].value[0] == 'c' &&
token[token_i].value[1] == 'a' && token[token_i].value[2] == 's' &&
token[token_i].value[3] == 'e') { token[token_i].typecode = 14;
token_i++; } //case->14
            else if (token[token_i].value[0] == 'd' &&
token[token_i].value[1] == 'e' && token[token_i].value[2] == 'f' &&
token[token_i].value[3] == 'a' && token[token_i].value[4] == 'u' &&
token[token_i].value[5] == 'l' && token[token_i].value[6] == 't')
{ token[token_i].typecode = 15; token_i++; }//default->15
            else if (token[token_i].value[0] == 'f' &&
token[token_i].value[1] == 'o' && token[token_i].value[2] == 'r')
{ token[token_i].typecode = 16; token_i++; }//for->16
            else if (token[token_i].value[0] == 'c' &&
token[token_i].value[1] == 'o' && token[token_i].value[2] == 'n' &&
token[token_i].value[3] == 't' && token[token_i].value[4] == 'i' &&
token[token_i].value[5] == 'n' && token[token_i].value[6] == 'u' &&
token[token_i].value[7] == 'e') { token[token_i].typecode = 17;
token_i++; } //continue->17
            else if (token[token_i].value[0] == 'b' &&
token[token_i].value[1] == 'r' && token[token_i].value[2] == 'e' &&
token[token_i].value[3] == 'a' && token[token_i].value[4] == 'k')
{ token[token_i].typecode = 18; token_i++; }//break->18
            else if (token[token_i].value[0] == 'g' &&
token[token_i].value[1] == 'o' && token[token_i].value[2] == 't' &&
token[token_i].value[3] == 'o') { token[token_i].typecode = 19;
token_i++; } //goto->19
            else
            {
                token[token_i].typecode = 1;
                token_i++;
            }
        }

        else if (isdigit(memory[i]) != 0)  //常数
        {
            value_i = 0;
            while (isdigit(memory[i]) != 0)
            {
                token[token_i].value[value_i] = memory[i];
                i++;
                value_i++;
            }
```

```
            token[token_i].typecode = 2;
            token_i++;
        }

        else if (memory[i] == '\"')         //字符串
        {
            token[token_i].value[0] = '\"';
            value_i = 1;
            i++;
            while (memory[i] != '\"')
            {
                token[token_i].value[value_i] = memory[i];
                i++;
                value_i++;
            }
            token[token_i].value[value_i] = '\"';
            token[token_i].typecode = 4;
            token_i++;
            i++;
        }

        else if (isspace(memory[i]) != 0)   //空格，跳过
        {
            i++;
        }

        else//符号
        {
            if (memory[i] == ',') { token[token_i].value[0] =
memory[i]; token[token_i].typecode = 20; i++; token_i++; }
            else if (memory[i] == ';') { token[token_i].value[0] =
memory[i]; token[token_i].typecode = 21; i++; token_i++; }
            else if (memory[i] == '=') { token[token_i].value[0] =
memory[i]; token[token_i].typecode = 22; i++; token_i++; }
            else if (memory[i] == '+') { token[token_i].value[0] =
memory[i]; token[token_i].typecode = 23; i++; token_i++; }
            else if (memory[i] == '-') { token[token_i].value[0] =
memory[i]; token[token_i].typecode = 24; i++; token_i++; }
            else if (memory[i] == '*') { token[token_i].value[0] =
memory[i]; token[token_i].typecode = 25; i++; token_i++; }
            else if (memory[i] == '(') { token[token_i].value[0] =
memory[i]; token[token_i].typecode = 26; i++; token_i++; }
            else if (memory[i] == ')') { token[token_i].value[0] =
memory[i]; token[token_i].typecode = 27; i++; token_i++; }
```

```
            else if (memory[i] == '{') { token[token_i].value[0] =
memory[i]; token[token_i].typecode = 28; i++; token_i++; }
            else if (memory[i] == '}') { token[token_i].value[0] =
memory[i]; token[token_i].typecode = 29; i++; token_i++; }
        }
    }
    //fclose(fp);

    //把得到的数据symbol加入符号表中
    for (int i = 0; i < token_i; i++)
    {
        if (token[i].typecode == 1)
        {
            for (int j = 0; j < i; j++)
            {
                if (!strcmp(symbol_table[j].lname, token[i].value))
                {
                    symbol_jug = 1;
                }
            }
            if (symbol_jug == 0)       //加入到符号表中
            {
                strcpy(symbol_table[symbol_i].lname, token[i].value);
                symbol_table[symbol_i].jug = 1;
                symbol_i++;
                symbol_jug = 0;
            }
        }
    }

    //输出token表
    FILE* fpwrite = fopen("outputfiles/token.txt", "w");
    if (fpwrite == NULL)
    {
        printf("failed to write file!");
    }
    for (i = 0; i < token_i; i++)
    {
        fprintf(fpwrite, "(%d,%s)\n", token[i].typecode,
token[i].value);
    }
    fclose(fpwrite);

}
```

5. void Grammar_analyser()

a) 功能：语法分析器
b) 描述：内含 LR（1）分析函数以及符号表写入函数
c) 代码

```
void Grammar_analyser()
{
    LR_1();
    WriteToSymbol();

}
```

6. void LR_1()

a) 功能：对词法分析产生的 token 串进行 LR（1）分析
b) 描述：根据状态转换表的各个状态来进行判断决定是否移进或进行规约
c) 代码

```
void LR_1()
{
    int i = 0;
    int count = 1;
    char str_rem[10];
    char sym_rem[10];
    memset(str_rem, 0, sizeof(str_rem));
    memset(sym_rem, 0, sizeof(sym_rem));
    int z;
    int address = -4;
    int id_i = 0;//记录一个句子里id的个数
    FILE* fp = fopen("outputfiles/syntax_analysis.txt", "w");
    if (fp == NULL)
        printf("failed to write syntax_analysis!");
    while (i < token_i)
    {
        LRstack.top = 0;
        LRstack.state[LRstack.top] = 0;
        int j = 0;
        for (z = 0; state_str[z].address != "\0" && z < 100; z++) //初
始化
            strcpy(state_str[z].address, "\0");
```

```
        while (action[j].action != 'a')
        {
            if (token[i].typecode == 5)    //int型
            {
                strcpy(str_rem, token[i].value);
                strcpy(sym_rem, token[i + 1].value);
                for (z = 0; symbol_table[z].jug == 1; z++)
                {
                    if (!strcmp(sym_rem, symbol_table[z].lname))
                    {
                        address = address + sizeof(int);
                        symbol_table[z].address = address;
                        strcpy(symbol_table[z].lcode,
token[i].value);
                        break;
                    }
                }
            }
            else if (token[i].typecode > 21 && token[i].typecode < 28)
        //符号
                strcpy(str_rem, token[i].value);
            else if (token[i].typecode == 1)    //标识符
            {
                str_rem[0] = 'I';
                str_rem[1] = 'D';
                str_rem[2] = '\0';
                strcpy(id_rem[id_i].name, token[i].value);
                strcpy(state_str[LRstack.top].address,
token[i].value);
                id_i++;
                if (token[i + 1].typecode == 22 && token[i +
2].typecode == 2)     //标识符赋值语句，记录其value值
                {
                    strcpy(sym_rem, token[i].value);
                    for (z = 0; symbol_table[z].jug == 1; z++)
                    {
                        if (!strcmp(sym_rem, symbol_table[z].lname))
                        {
                            symbol_table[z].value = atoi(token[i +
2].value);
                            break;
                        }
                    }
                }
```

```
            }
            else if (token[i].typecode == 2)    //数字
            {
                str_rem[0] = 'I';
                str_rem[1] = 'N';
                str_rem[2] = 'T';
                str_rem[3] = '_';
                str_rem[4] = 'N';
                str_rem[5] = 'U';
                str_rem[6] = 'M';
                str_rem[7] = '\0';
                strcpy(state_str[LRstack.top].address,
token[i].value);
            }
            else if (token[i].typecode == 21)   //分号
            {
                str_rem[0] = '$';
                str_rem[1] = '\0';
            }
            j = 0;
            while (action[j].old_state != LRstack.state[LRstack.top]
|| strcmp(str_rem, action[j].cause))
            {
                j++;
                if (j > 1000)
                {
                    fprintf(fp, "cant receive a sentense!!!");
                    printf("cant receive a sentense!!!");
                    exit(1);
                }
            }
            if (action[j].action == 's')        //移进
            {
                strcpy(state_str[LRstack.top].alpb, str_rem);
                LRstack.top++;
                LRstack.state[LRstack.top] = action[j].new_state;
                i++;
            }
            else if (action[j].action == 'r')   //规约
            {
                int k = 0;
                count = 1;
                for (k = 0; action[j].right[k] != '\0'; k++)
                {
```

```
                    if (action[j].right[k] == '=' ||
action[j].right[k] == '+' || action[j].right[k] == '-' ||
action[j].right[k] == '*' || action[j].right[k] == '(' ||
action[j].right[k] == ' ')
                        count = count + 2;
                }
                LRstack.top = LRstack.top - count;
                k = 0;
                while (action[k].old_state !=
LRstack.state[LRstack.top] || strcmp(action[k].cause, action[j].left))
                {
                    k++;
                    if (k > 1000)
                    {
                        fprintf(fp, "cant receive a sentense!!!");
                        printf("cant receive a sentense!!!");
                        exit(1);
                    }
                }
                Generate_IntermediateCode(fp, j, k);
            }
            else if (action[j].action == 'a')   //动作为accept，输出一
个句子。
            {
                i++;
                fprintf(fp, "accept a sentense!\n");
                id_i = 0;
                id_res = 0;
                int p;
                for (p = 0; p < 100; p++)
                    strcpy(id_rem[p].name, "\0");
            }
        }
    }

}
```

7. `void WriteToSymbol()`

   a) 功能：将属性值写入符号表

   b) 描述：根据 symbol_table 结构体数组逐个元素写入文件中

   c) 代码

```
void WriteToSymbol()
{
    //symbol表的输出
    FILE* fp = fopen("outputfiles/symbol.txt", "w");
    if (fp == NULL)
    {
        printf("failed to write symbol!");
    }
    for (int i = 0; i < token_i; i++)
    {
        if (symbol_table[i].jug == 1)
        {
            fprintf(fp, "(%s,%s,%d)\n", symbol_table[i].lname,
symbol_table[i].lcode, symbol_table[i].address);
        }
    }
    fclose(fp);

}
```

8. `void Generate_IntermediateCode(FILE*fp,int j, int k);`

a) 功能：生成中间代码

b) 描述：根据 LR（1）的分析结果产生中间代码

c) 代码

```
void Generate_IntermediateCode(FILE *fp,int j,int k)
{
    //下面进行语法制导翻译以及中间代码生成环节，pno是产生式的编号
    if (action[j].PNO == 3) // ID = S
    {
        ast[t_i].t = t_i;
        strcpy(ast[t_i].ID, state_str[LRstack.top].address);
        t_i++;
        id_res++;
    }
    else if (action[j].PNO == 4)   //S ->S+A
    {
        ast[t_i].t = t_i;
        if (state_str[LRstack.top].t != 0)
        {
            ast[t_i].arg_1 = state_str[LRstack.top].t;
            state_str[LRstack.top].t = t_i;
```

```
            }
            else
            {
                strcpy(ast[t_i].arg_1, state_str[LRstack.top].address);
                if (t_i == 0)
                    state_str[LRstack.top].t = -1;
                else
                    state_str[LRstack.top].t = t_i;
            }
            ast[t_i].op = '+';
            if (state_str[LRstack.top + 2].t != 0)
            {
                ast[t_i].arg_r = state_str[LRstack.top + 2].t;
                state_str[LRstack.top + 2].t = 0;
            }
            else
                strcpy(ast[t_i].arg_2, state_str[LRstack.top +
2].address);
            t_i++;
        }
        else if (action[j].PNO == 5)   //S ->S-A
        {
            ast[t_i].t = t_i;
            if (state_str[LRstack.top].t != 0)
            {
                ast[t_i].arg_l = state_str[LRstack.top].t;
                state_str[LRstack.top].t = t_i;
            }
            else
            {
                strcpy(ast[t_i].arg_1, state_str[LRstack.top].address);
                if (t_i == 0)
                    state_str[LRstack.top].t = -1;
                else
                    state_str[LRstack.top].t = t_i;
            }
            ast[t_i].op = '-';
            if (state_str[LRstack.top + 2].t != 0)
            {
                ast[t_i].arg_r = state_str[LRstack.top + 2].t;
                state_str[LRstack.top + 2].t = 0;
            }
            else
                strcpy(ast[t_i].arg_2, state_str[LRstack.top +
```

```
2].address);
            t_i++;
        }
        else if (action[j].PNO == 7)   //A ->A*B
        {
            ast[t_i].t = t_i;
            if (state_str[LRstack.top].t != 0)
            {
                ast[t_i].arg_1 = state_str[LRstack.top].t;
                state_str[LRstack.top].t = t_i;
            }
            else
            {
                strcpy(ast[t_i].arg_1, state_str[LRstack.top].address);
                if (t_i == 0)
                    state_str[LRstack.top].t = -1;
                else
                    state_str[LRstack.top].t = t_i;
            }
            ast[t_i].op = '*';
            if (state_str[LRstack.top + 2].t != 0)
            {
                ast[t_i].arg_r = state_str[LRstack.top + 2].t;
                state_str[LRstack.top + 2].t = 0;
            }
            else
                strcpy(ast[t_i].arg_2, state_str[LRstack.top +
2].address);
            t_i++;
        }
        else if (action[j].PNO == 9)   //B->(S)
        {
            state_str[LRstack.top].t = state_str[LRstack.top + 1].t;
            state_str[LRstack.top + 1].t = 0;
        }
        else if (action[j].PNO == 11)  //B->INT_NUM
        {
            ast[t_i].t = t_i;
            ast[t_i].arg_r = atoi(state_str[LRstack.top].address);
        }
        strcpy(state_str[LRstack.top].alpb, action[j].left);
        LRstack.top++;
        LRstack.state[LRstack.top] = action[k].new_state;
        fprintf(fp, "%s->%s\n", action[j].left, action[j].right);
```

```
}
```

9. void WriteToAst()

a) 功能：生成汇编代码

b) 描述：根据翻译结果将每条语句转为汇编代码

c) 代码

```
void WriteToAst()
{
    FILE* fp= fopen("outputfiles/ast.txt", "w");
    if (fp == NULL)
    {
        printf("failed to write ast!");
    }
    int num_jug[32] = { 0 };
    int ram[32];
    for (int i = 0; i < 32; i++)
    {
        ram[i] = i;
    }
    int j = 0;
    for (int i = 0; i < t_i; i++)
    {
        if (ast[i].op == 0 && !strcmp(ast[i].arg_1, "\0")
&& !strcmp(ast[i].arg_2, "\0") && ast[i].arg_1 == 0 && ast[i].arg_r !=
0)  //表示ID = INT_NUM
        {
            fprintf(fp, "%s = %d\n", ast[i].ID, ast[i].arg_r);
            fprintf(fp, "t%d = %s\n", i, ast[i].ID);
            printf("MOV\t%s,%d\n", ast[i].ID, ast[i].arg_r);
            num_jug[ast[i].t] = j;
            for (int k = ast[i].t; k < 32; k++)
                num_jug[k] = j;
            j++;
        }
        else
        {
            if (ast[i].op != 0 && ast[i].arg_1 == 0 && ast[i].arg_r ==
0)  //t = a op b
            {
                fprintf(fp, "t%d = %s %c %s\n", i, ast[i].arg_1,
```

```
ast[i].op, ast[i].arg_2);
                printf("MOV \tR%d,%s\n", i - num_jug[i],
ast[i].arg_1);
                if (ast[i].op == '+')
                    printf("ADD \tR%d,%s\n", i - num_jug[i],
ast[i].arg_2);
                else if (ast[i].op == '-')
                    printf("SUB \tR%d,%s\n", i - num_jug[i],
ast[i].arg_2);
                else if (ast[i].op == '*')
                    printf("MUL \tR%d,%s\n", i - num_jug[i],
ast[i].arg_2);
                else if (ast[i].op == '/')
                    printf("DIV \tR%d,%s\n", i - num_jug[i],
ast[i].arg_2);
            }
            else if (ast[i].arg_l != 0 && ast[i].arg_r != 0) //for t0
            {
                if (ast[i].arg_l == -1)
                {
                    fprintf(fp, "t%d = t0 %c t%d\n", i, ast[i].op,
ast[i].arg_r);
                    if (ast[i].op == '+')
                        printf("ADD \tR%d,R%d\n", ram[ast[i].arg_l -
num_jug[i]], ram[ast[i].arg_r - num_jug[i]]);
                    else if (ast[i].op == '-')
                        printf("SUB \tR%d,R%d\n", ram[ast[i].arg_l -
num_jug[i]], ram[ast[i].arg_r - num_jug[i]]);
                    else if (ast[i].op == '*')
                        printf("MUL \tR%d,R%d\n", ram[ast[i].arg_l -
num_jug[i]], ram[ast[i].arg_r - num_jug[i]]);
                    else if (ast[i].op == '/')
                        printf("DIV \tR%d,R%d\n", ram[ast[i].arg_l -
num_jug[i]], ram[ast[i].arg_r - num_jug[i]]);
                    ram[ast[i].arg_r - num_jug[i]] = 0;
                    j++;
                    for (int k = ast[i].t; k < 32; k++)
                        num_jug[k] = j;
                }
                else if (ast[i].arg_r == -1)
                    fprintf(fp, "t%d = t%d %c t0\n", i, ast[i].arg_l,
ast[i].op);
                else
                {
```

```
                    fprintf(fp, "t%d = t%d %c t%d\n", i,
ast[i].arg_l, ast[i].op, ast[i].arg_r);
                    if (ast[i].op == '+')
                        printf("ADD \tR%d,R%d\n", ram[ast[i].arg_l -
num_jug[i]], ram[ast[i].arg_r - num_jug[i]]);
                    else if (ast[i].op == '-')
                        printf("SUB \tR%d,R%d\n", ram[ast[i].arg_l -
num_jug[i]], ram[ast[i].arg_r - num_jug[i]]);
                    else if (ast[i].op == '*')
                        printf("MUL \tR%d,R%d\n", ram[ast[i].arg_l -
num_jug[i]], ram[ast[i].arg_r - num_jug[i]]);
                    else if (ast[i].op == '/')
                        printf("DIV \tR%d,R%d\n", ram[ast[i].arg_l -
num_jug[i]], ram[ast[i].arg_r - num_jug[i]]);
                    ram[ast[i].arg_r - num_jug[i]] = ast[i].arg_l -
num_jug[i];
                    j++;
                    for (int k = ast[i].t; k < 32; k++)
                        num_jug[k] = j;
                }
            }
            else if (ast[i].arg_l == 0 && ast[i].arg_r != 0) //t = a
op t
            {
                fprintf(fp, "t%d = %s %c t%d\n", i, ast[i].arg_l,
ast[i].op, ast[i].arg_r);
                if (ast[i].op == '+')
                    printf("ADD \tR%d,R%d\n", ast[i].arg_l -
num_jug[i], ast[i].arg_r - num_jug[i]);
                else if (ast[i].op == '-')
                    printf("SUB \tR%d,R%d\n", ast[i].arg_l -
num_jug[i], ast[i].arg_r - num_jug[i]);
                else if (ast[i].op == '*')
                    printf("MUL \tR%d,R%d\n", ast[i].arg_l -
num_jug[i], ast[i].arg_r - num_jug[i]);
                else if (ast[i].op == '/')
                    printf("DIV \tR%d,R%d\n", ast[i].arg_l -
num_jug[i], ast[i].arg_r - num_jug[i]);
            }
            else if (ast[i].arg_l != 0 && ast[i].arg_r == 0) //t = t
op a
            {
                fprintf(fp, "t%d = t%d %c %s\n", i, ast[i].arg_l,
ast[i].op, ast[i].arg_2);
```

```
                if (ast[i].op == '+')
                    printf("ADD \tR%d,%s\n", ast[i].arg_1 -
num_jug[i], ast[i].arg_2);
                else if (ast[i].op == '-')
                    printf("SUB \tR%d,%s\n", ast[i].arg_1 -
num_jug[i], ast[i].arg_2);
                else if (ast[i].op == '*')
                    printf("MUL \tR%d,%s\n", ast[i].arg_1 -
num_jug[i], ast[i].arg_2);
                else if (ast[i].op == '/')
                    printf("DIV \tR%d,%s\n", ast[i].arg_1 -
num_jug[i], ast[i].arg_2);
                j++;
                for (int k = ast[i].t; k < 32; k++)
                    num_jug[k] = j;
            }
            if (!strcmp(ast[i].arg_1, "\0") && !strcmp(ast[i].arg_2,
"\0") && ast[i].arg_1 == 0 && ast[i].arg_r == 0)
            {
                fprintf(fp, "%s = t%d\n", ast[i].ID, i - 1);
                int u = ram[i - 1 - num_jug[i]];
                while (ram[u] != u)
                    u = ram[u];
                printf("MOV \t%s,R%d\n", ast[i].ID, u);
            }
        }
    }
    fclose(fp);

}
```

4 实验结果与分析

对实验的输入输出结果进行展示与分析。注意：要求给出编译器各阶段（词法分析、语法分析、中间代码生成、目标代码生成）的输入输出并进行分析说明。

1. 词法分析

a) 输入：类 c 代码

```
int result;
int a;
int b;
int c;
a = 8;
```

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

b) 输出：token

```
(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,d)
(22,=)
(26,()
(1,a)
(24,-)
(1,b)
(27,))
(23,+)
(26,()
(1,a)
(24,-)
(1,c)
(27,))
(23,+)
(26,()
(1,a)
(24,-)
(1,c)
(27,))
(21,;)
```

2. 语法分析

a) 输入：token 串、action 表和 goto 表

Action 表和 goto 表

状态	ACTION										GOTO			
	ID	int	INT_NUM	=	+	-	*	(	)	$	S'	S	A	B
0	shift 2	shift 3									1			
1										accept				
2				shift 4										
3	shift 5													
4	shift 9		shift 10					shift 11				6	7	8
5										reduce S' -> int ID				
6					shift 12	shift 13				reduce S' -> ID = S				
7					reduce S -> A	reduce S -> A	shift 14			reduce S -> A				
8					reduce A -> B	reduce A -> B	reduce A -> B			reduce A -> B				
9					reduce B -> ID	reduce B -> ID	reduce B -> ID			reduce B -> ID				
10					reduce B -> INT_NUM	reduce B -> INT_NUM	reduce B -> INT_NUM			reduce B -> INT_NUM				
11	shift 18		shift 19					shift 20				15	16	17
12	shift 9		shift 10					shift 11					21	8
13	shift 9		shift 10					shift 11					22	8
14	shift 9		shift 10					shift 11						23
15					shift 24	shift 25			shift 26					
16					reduce S -> A	reduce S -> A	shift 27		reduce S -> A					
17					reduce A -> B	reduce A -> B	reduce A -> B		reduce A -> B					
18					reduce B -> ID	reduce B -> ID	reduce B -> ID		reduce B -> ID					
19					reduce B -> INT_NUM	reduce B -> INT_NUM	reduce B -> INT_NUM		reduce B -> INT_NUM					
20	shift 18		shift 19					shift 20				28	16	17
21					reduce S -> S + A	reduce S -> S + A	shift 14			reduce S -> S + A				
22					reduce S -> S - A	reduce S -> S - A	shift 14			reduce S -> S - A				
23					reduce A -> A * B	reduce A -> A * B	reduce A -> A * B			reduce A -> A * B				
24	shift 18		shift 19					shift 20					29	17
25	shift 18		shift 19					shift 20					30	17
26					reduce B -> (S)	reduce B -> (S)	reduce B -> (S)			reduce B -> (S)				
27	shift 18		shift 19					shift 20						31
28					shift 24	shift 25			shift 32					
29					reduce S -> S + A	reduce S -> S + A	shift 27		reduce S -> S + A					
30					reduce S -> S - A	reduce S -> S - A	shift 27		reduce S -> S - A					
31					reduce A -> A * B	reduce A -> A * B	reduce A -> A * B		reduce A -> A * B					
32					reduce B -> (S)	reduce B -> (S)	reduce B -> (S)		reduce B -> (S)					

b) 输出：symbol 符号表、语法分析过程

Symbol 符号表

```
(result,int,0)
(a,int,4)
(b,int,8)
(c,int,12)
```

语法分析过程 syntax_analysis

```
S'->int ID
accept a sentense!
S'->int ID
accept a sentense!
S'->int ID
accept a sentense!
S'->int ID
accept a sentense!
B->INT_NUM
A->B
S->A
S'->ID=S
accept a sentense!
B->INT_NUM
A->B
S->A
S'->ID=S
accept a sentense!
B->INT_NUM
A->B
S->A
S'->ID=S
accept a sentense!
B->ID
A->B
B->ID
A->A*B
S->A
B->ID
A->B
S->A
B->ID
A->B
S->S-A
B->(S)
A->B
S->S+A
B->ID
A->B
S->S+A
S'->ID=S
accept a sentense!
B->ID
```

```
A->B
S->A
B->ID
A->B
S->S-A
B->(S)
A->B
S->A
B->ID
A->B
S->A
B->ID
A->B
S->S-A
B->(S)
A->B
S->S+A
B->ID
A->B
S->A
B->ID
A->B
S->S-A
B->(S)
A->B
S->S+A
S'->ID=S
accept a sentense!
```

3. 中间代码生成（本实验代码中，中间代码生成与语法分析同时进行）
   a) 输入：状态栈的各个状态
   b) 输出：syntax_analysis
4. 目标代码生成
   a) 输入：syntax_analysis
   b) 输出：汇编代码、三地址码

```
a = 8
t0 = a
b = 5
t1 = b
c = 3
t2 = c
t3 = a * b
t4 = a - b
t5 = t3 + t4
```

```
t6 = t5 + c
result = t6
t8 = a - b
t9 = a - c
t10 = t8 + t9
t11 = a - c
t12 = t10 + t11
d = t12
```

```
MOV     a,8
MOV     b,5
MOV     c,3
MOV     R1,a
MUL     R1,b
MOV     R2,a
SUB     R2,b
ADD     R1,R2
ADD     R1,c
MOV     result,R1
MOV     R3,a
SUB     R3,b
MOV     R4,a
SUB     R4,c
ADD     R3,R4
MOV     R5,a
SUB     R5,c
ADD     R3,R5
MOV     d,R3
```

5 实验中遇到的困难与解决办法

困难与解决办法：

每次在实验前都需要花费很长的时间进行理论知识的复习或查阅相关资料。

由于理论部分的理解不够透彻，因此后面的语法分析以及代码生成无法完全实现在词法分析过程中预计实现的功能，限制了发挥空间。

收获：

1. 提高了程序编写能力，同时也对编译原理这门课程有了更深层次的理解
2. 了解了编译器的大体架构并从无到有构造了一个非常简易的编译器

建议：

在每个实验资料或指导书中给关键算法增加更加详细的伪代码或者流程图方便编写时进行参考。