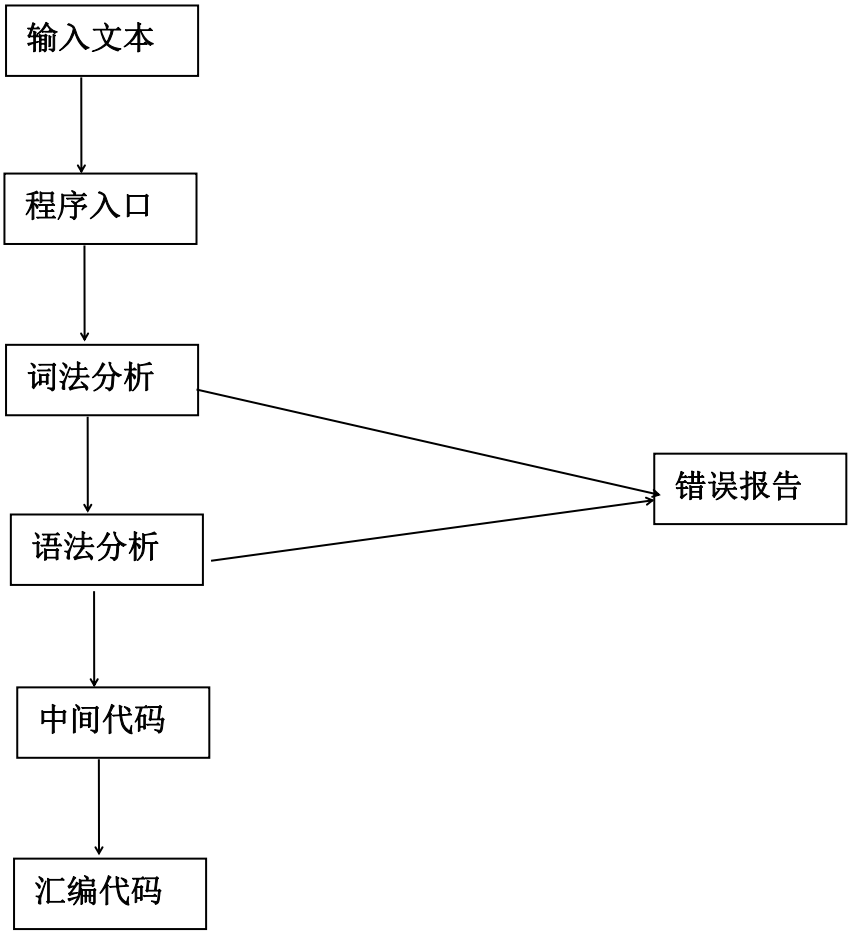
目　　录

一、整体框架：



二、语法分析：

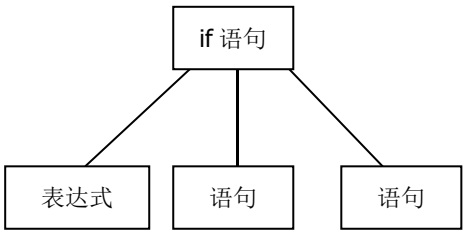
1、语法

Program ::= <声明串>
<声明串> ::=<声明>{<声明>}
<声明> ::=int　<ID>　<声明类型> | void　<ID>　<函数声明>
<声明类型>::=<变量声明> | <函数声明>
<变量声明> ::=　;
<函数声明> ::=’ (‘<形参>’) ‘<语句块>

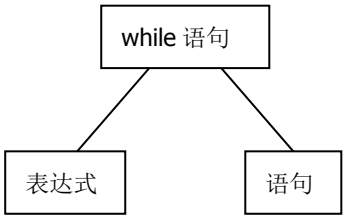
<形参>::= <参数列表> | void
<参数列表> ::= <参数> {, <参数>}
<参数> ::= int <ID>
<语句块> ::= ‘{‘<内部声明> <语句串>’}’
<内部声明> ::= 空 | <内部变量声明>{<内部变量声明>}
<内部变量声明>::=int <ID> <变量声明>
<语句串> ::= <语句>{ <语句> }
<语句> ::= <if 语句> |< while 语句> | <return 语句> | <赋值语句>
<赋值语句> ::= <ID> =<表达式>;
<return 语句> ::= return [<表达式>] **（注：[ ]中的项表示可选）**
<while 语句> ::= while ‘(‘<表达式> ‘)’ <语句块>
<if 语句> ::= if ‘(‘<表达式>’)’ <语句块> [else <语句块>]**（注：[ ]中的项表示可选）**
<表达式>::=<加法表达式>{ relop <加法表达式> } **（注：relop-> <|<=|>|>=|==|!=）**
<加法表达式> ::= <项> {+ <项> | -<项>}
<项> ::= <因子> {* <因子> | /<因子>}
<因子> ::=num | ‘(‘<表达式>’)’ |<ID> FTYPE
FTYPE ::= <call> | 空
<call> ::=’(’<实参列表> ’) ’
<实参> ::=<实参列表> | 空
<实参列表> ::= <表达式>{, <表达式>}
<ID>::=字母(字母|d 数字)*

2、支持语句与数据类型

if 语句：



while 语句：



支持的语句与运算：

1) 数据类型：int，void
2) 语句：赋值（=），if, while，return=
3) 数学运算：+，－，*，/
4) 关系运算：==，>，<，>=，<=，!=
6) 支持函数的定义、调用
7) 注释语句：C 类型的 /* */ 和 C++类型的 //

3、数据结构

```
class yfq
{
private:
    int end_flag;          //程序结束标志
    int read_main;         //是否已经读过 main 函数
    int num_zdkh;          //左大括号的数目，用于判断变量是否为全局变量
    int return_flag;
    char return_words[100];
    char c;                //c = fin.get()，临时读入变量
    char bl_name[100];          //临时记录变量名(以'\0'结束)
    char hs_name[100];          //临时记录函数名（以'\0'结束）
    int len_hs;                 //从函数名至')'的长度
    int available_T;            //Ti,i=available_T，为当前可用的 i
    int available_L;            //Li,i=available_L
    char var_name[10][100];     //记录参数名
    int kh[100];                //记录读到的右括号
    int len_kh;                 //
    char headwords[100];        //句首字符
    int is_hs_head;          //是否函数居于句首标志（判断是否需要 return）
    int is_hs_dy;            //是否函数定义

    int len_hs;              //函数信息表的长度
    int break_hs;            //在函数名数组中搜索给定函数名时断开的数
    hs_info hs[500];         //函数信息表数组
    int len_global_bl;   //全局变量长度
    bl_info global_bl[500];   //全局变量名数组
    int len_local_bl;         //局部变量长度
    bl_info local_bl[500];    //局部变量名数组

    ifstream fin;
    ifstream fin1;            //
    ofstream fout1;           //输出中间代码
```

```
    ofstream fout2;             //输出汇编代码
public:
    yfq();
    ~yfq();
    bool zs_handle();                                   //判断是否为有效注释 && 去除注释
                                                        //bool u_handle(char u[], int len_u,int mode);       //因子处理函数
                                                        //bool s_handle(char c[], int len_c);  //加法表达式处理函数

    bool exist_bl(char name[], int mode);     //判断变量名是否存在, mode = 1: 局部变量   mode = 0: 全局变量
    bool exist_hs(char name[], int var_num); //判断函数名是否存在 && 参数数目是否正确
    void insert_bl(char name[], int mode);

    //将定义的变量插入 global_bl[] / local_bl[]
    //mode   0->全局变量   1->局部变量

    void insert_hs(char name[], int var_num);
    //将定义的函数插入 hs[]，var_num：参数

    void delete_bl();
    //读到'}'删除该区间读到的变量，'{'开始（并生成中间代码）
    bool right_num(char c[], int len_c);      //判断是否为合法数字
    int fh_rank(char fh);                       //计算符号等级
    bool r_handle(char c[], int len_c, int mode);         //表达式处理函数，mode: 0-> = r;   1-> if( r )   2-> while( r )
    bool bl_handle();                           //句首为变量的处理函数
    bool void_handle();                         //句首为 void 的处理函数
    bool int_handle();                          //句首为 int 的处理函数
    int get_kind(char c[], int len_c);        //合法变量名，合法数字，
    void error_rep();                           //报错及相关处理函数
    int readhead();                             //读句首
    void analyze();
};

struct hs_info                          //记录函数的数据结构
{
    char hs_name[20];
    int f_return;
    char w_return[20];
    int var_num;
```

```
};

struct bl_info                          //记录变量的数据结构
{
    char bl_name[20];
    int pos;      //0：寄存器   1：内存
    int num;      //寄存器号
};

int   available_reg[32];              //有效寄存器号数组
Int max_num;              //reg1 恒定值为 0，全局变量从$31 向前分配寄存器，
max_num 为全局变量使用的最小的寄存器
```

4、设计思想

a. 总体架构：本程序使用了 LR1 的设计思想，并做了较大的改动。我最初的想法是严格按照 LR1 算法的思想，以 program 为 S′，并根据状态转换图画出 action 表。但是，在实现的过程中，发现这样的问题很大。首先，类 c 语法有 26 个产生式，要准确画出全部的状态转换图是十分困难的。其次，这种方式也是十分不必要的。比如，当程序读到了 int，那么接下来的步骤我们可想而知是继续读，反映到 LR1 算法中，就是当前状态 i 与当前待输入字符，对应的为 Sj。而若读到了';'，则对应的一定为 rj。所以当读到';'、'}'等就一定是指向规约操作，而读到其他的就会继续读入。所以，该程序以 LR1 算法思想为指导，并根据对 c 语言语法的理解

b. readhead()：每行的第一个数据结构决定了对该行的操作。readhead()函数读出每行的第一个数据结构，一共有以下 9 种类型：变量、函数、void、int、if、while、return、{、}

Case 变量：

if（exist_bl(变量名,0) == false && exist_bl(变量名，1) == false）

{报错;}

//如果该变量在全局变量与局部变量中都没没有，则报错

else {bl_handle();} 否则，调用 bl_handle()函数继续分析

Case 函数：

if（exist_hs(name) == false ）

{报错;}

//name 为 从函数名到）的以'\0'结尾的字符串。在 exist_hs()函数中，会检测是否函数名和参数个数都正确，只有都正确才会返回 true。

置 Is_hs_dy = 0; 表明现在是调用函数。

Case void：

Void_handle();

Case int:

Int_handle();

Case if:

if_handle();

Case while:

while_handle();

Case return:

return_handle();

Case '}':

ydkh_handle();

Case 其他：

报错

关于后面几个函数，会在后面结合代码说明。

c. 关于变量和函数声明的处理：

Void 关键字一定是函数声明。Void_handle()继续读入 void 后序部分，直到'{'，若 exist_hs() == yes|| exist_hs() == error 说明已有该函数名，或函数名/参数名 命名错误。会报错，因为该编译器不支持函数重载。不然，将函数名和参数数目写入数组 Hs_info hs[]。无参函数形式为 函数名( )，不是（void）。函数是否需要返回标志 f_return = 0;（只能根据定义时是 int 还是 void 来判断，意味着如果 void 函数 有返回值或 int 函数无返回值这种错误是没办法发现的）。

Int 关键字可能是函数定义，也可能是变量定义，其中变量定义还分为局部变量定义和全局变量定义。先判断是函数定义还是变量定义，从 int 后第一个非' '的字符开始读到)或者；。然后判断方法及操作同 void。如果是变量，根据 f_inside(初始化为 0，每读到一个'{'，f_inside++；读到一个'}'，f_inside--)是否为 0 判断此时是否为全局还是局部变量。如果是全局变量，并且 exist_bl(变量名，0) == no 则插入全局变量数组；如果是局部变量，并且 exist_bl(变量名，1) == no，则插入局部变量数组。尤其是，当读到了'{'，也需要插入到局部变量数组。

变量声明的过程不会有寄存器的分配，只有在使用变量的过程中才会有寄存器分配。

```
struct bl_info
{
    char bl_name[20];
    int pos;      //0：寄存器   1：内存   2：都在
    int num;      //寄存器号
};
```

当给全局变量赋值时，会将该值存储在内存中。而当使用该变量时（等号右边），则会为该变量分配存寄存器。从$31 开始依次向前分配寄存器。其 pos 置 2。当为局部变量赋值时，会将该变量的值存储在内存中，当使用该变量的时候，会从内存中取出该值分配给寄存器。寄存器分配过程：从 available_reg 中取最小的为 1 的值 j，并把 $j 分配给该变量，同时，该变量的 pos 置 2。

```
For(j = 1;j<max_reg;j++)
{
    If(available_reg[j] == 1)
        Break;
}
Pos = 0; num = j;
```

当读到了'}'，则会在局部变量栈中持续退栈，直至第一个'{'也退栈，该过程即为局部变量的释放。

d. 函数调用的处理

函数调用若有参数，则需要先把参数入栈（将实参的 pos、num

赋予形参），类似于传地址的参数传递方式。如果函数有返回值，则先传值，再 return；否则，直接 return。

e. 表达式的处理

对有小括号的情况处理的不好，经常会程序中断，没有解决。将整个表达式 如：if(r)，则 r 为表达式 读入并传递给 r_handle()，r_handle()根据<,>,=进行拆分，如 s1 < s2，则将 s1,s2 分别传递给 s_handle()。r_handle()扫描，确定* /(rank = 2)和 + -(rank = 1)的数目 num1,num2，将栈和 num1，num2 传递给 s_handle().然后 s_handle()利用栈和递归函数的方法最终得到一个空栈：

如有 s1 = a+b*c+d

则初始的符号栈为 a+b*c+d,

读到*，则将 b*c 退栈，Ti 入栈，num2--(表示 rank = 2 的数目)，并将栈 a+Ti+b ,num1,num2 传递给下一个调用的 s_handle()。

f. If, while 的处理

if 和 while 翻译成中间代码的形式分别为：

If:

L0: (jx, a, b, L2)

L1: (j , - , - , $)

L2: (-, - , - , -)

...

Li:

while:

L0: (jx, a, b, L2)

L1: (j , - , - , $)

L2: (-, - , - , -)

...

Li-1:(j, - , - , L0)

Li:

$表示待定的地址，x 表示相应的操作符。因为无法在第一遍扫描即得知$的值，所以，要先把字符串”L0”、”:”、”(“、”jx”、”,”、”a”...”$”、”:”读到一个二维字符数组 s[500][20]。然后，当$:确定后，再把 s[][]写入中间代码文件，其中$用最后的地址(Li)来替代。

```
for(i = 0;i<len_s;i++)
{
if(strcmp(s[i],")") == 0)
{
fout1<<")"<<endl;
}
else if(strcmp(s[i],"$") == 0)
{
fout1<<return_addr;
}
else
{
fout1<<s[i];
}
}
```

g. 程序结束

读到 main 函数后，read_main = 1; f_kh 初始化为 0，每读到一个"{"，f_kh++，读到一个"}"f_kh--。若 read_main = 1 && f_kh == 0，程序结束。整个扫描分析过程如下：

```
void yfq::analyze()
{
	while (end_flag == 0)				//程序停止标志
	{
		int headword = readhead();	//句首字符
		if (headword == _bl)			//句首为合法变量
		{

			strcpy_s(headwords, 20, bl_name);  //将句首变量名记录到 headname 中
			is_hs_head = 0;

			if (bl_handle() == true)
				continue;
		}
		else if (headword == _hs)
		{
			is_hs_head = 1;

			r_handle(hs_name, len_hs, 0);
		}
		else if (headword == _void)
		{
			is_hs_head = 0;
			is_hs_dy = 1;
			if (void_handle() == true)
			{
				is_hs_dy = 0;
				continue;
			}
		}
		else if (headword == _int)
		{
```

```
        is_hs_head = 0;
        is_hs_dy = 1;
        if (int_handle() == true)
        {
            is_hs_dy = 0;
            continue;
        }

    }
    else if (headword == _if)
    {
        is_hs_head = 0;
        kh[len_kh++] = _if;
        char temp[20];
        int len_temp = 0;
        c = fin.get();          // '('
        c = fin.get();
        while (c != ')')
        {
            temp[len_temp++] = c;
            c = fin.get();
        }
        if (r_handle(temp, len_temp, 1) == true)
            continue;
    }
    else if (headword == _while)
    {
        is_hs_head = 0;
        kh[len_kh++] = _while;
        char temp[20];
        int len_temp = 0;
        c = fin.get();          // '('
        c = fin.get();
        while (c != ')')
        {
            temp[len_temp++] = c;
            c = fin.get();
        }
        if (r_handle(temp, len_temp, 2) == true)
            continue;
    }
    else if (headword == _return)    //在函数定义的时候，将 return 的值写入 return_words
    {
        cout << "c0 =" << c << endl;
```

```
        is_hs_head = 0;
        int count = 1;
        if (read_main == 0)
        {
            //if (is_hs_dy == 1)
            //{
            char temp[30];
            int len_temp = 0;
            //c = fin.get();
            //cout << "c" << count++ << " =" << c << endl;

            while (c == ' ')
            {
                c = fin.get();
                //cout << "c" << count++ << " =" << c << endl;
            }
            if (c != ';')                    //有返回值
            {
                while (c != ';')
                {
                    temp[len_temp++] = c;
                    c = fin.get();
                    //cout << "c" << count++ << " =" << c << endl;
                }
                temp[len_temp] = '\0';
            }
            strcpy_s(return_words, 25, temp);
            //cout <<"return_words ="<< return_words;
            //}
        }
        else
        {
            end_flag = 1;
        }
    }
    else if (headword == _ydkh)
    {
        is_hs_head = 0;
        //cout << kh[0];
        //cout << len_kh;
        len_kh--;
        if (kh[len_kh] == is_main)    //在 get_kind 中入栈
        {
            end_flag = 1;
```

```
                }
                else if (kh[len_kh] == _if)
                {
                        fout1 << "L" << available_L++ << ":" << endl;
                }
                else if (kh[len_kh] == _while)
                {
                        fout1 << endl << "L" << available_L++ << endl;
                        fout1 << "( j " << " , _" << " , _" << " , " << "L" << available_L - 4 << " )" << endl;
                        fout1 << endl << "L" << available_L++ << endl;
                }

            }
            else            //出错
            {
                error_rep();
            }
        }
}
```

四、中间代码与汇编代码

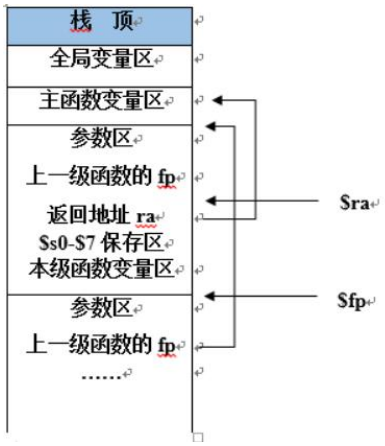
在上面的语法检测的同时，中间代码和目标代码也已经生成。

1、四元式设计

四元式种类	代表的含义
(= , 2 , , temp)	temp = 2;
(call, f , , a)	a = f()
(call, f , ,)	f()
(<=.., a , b ,)	a <= b
(jne , , , lable)	if not satisfy(==false) then jump
(jmp , , , label)	jump to label
ret	return

Par a	f(int a, ...)

2、存储分配方案



3、Mips 指令与语义

取立即数	Li $s1, 100	$s1 = 100
加	Add s1,s1,s2,$s3	s3=s3=s1+$s2
立即数加	Addi s1,s1,s2,100	s3=s3=s1+100
立即数减	Subi s1,s1,s2,100	s3=s3=s1-100
减	sub s1,s1,s2,$s3	s3=s3=s1-$s2
乘	mult s2,s2,s3	Hi，Lo=s2* s2* s3
除	div s2,s2,s3	Lo=s2/s2/s3 Hi=s2Mods2Mods3

取字	Lw s1,100(s1,100(s2)	s1=memory[s1=memory[s2+100]
存字	sw s1,100(s1,100(s2)	memory[s2+100]=s2+100]=s1
Beq	beq s1,s1,s2,100	If（s1==s1==s2） goto PC+4+400
Bne	beq s1,s2,100	If（$s1！=$s2） goto PC+4+400
Slt	slt s1,s2, $s3	If（s2<s3） $s1=1; Else $s1=0;
j	J 2000	Goto 8000
jr	Ja $ra	Goto $ra
jal	Jal 2500	$ra=PC+4; Goto 10000;

4、存储器动态管理

```
struct bl_info
{
    char bl_name[20];
    int pos;     //0：寄存器  1：内存  2：都在
    int num;     //寄存器号
};
```

当给全局变量赋值时，会将该值存储在内存中。而当使用该变量时（等号右边），则会为该变量分配存寄存器。从$31 开始依次向前分配寄存器。其 pos 置 2。当为局部变量赋值时，会将该变量的值存储在内存中，当使用该变量的时候，会从内存中取出该值分配给寄存器。寄存器分配过程：从 available_reg 中取最小的为 1 的值 j，并把 $j 分配给该变量，同时，该变量的 pos 置 2。

```
For(j = 1;j<max_reg;j++)
{
    If(available_reg[j] == 1)
        Break;
}
Pos = 0; num = j;
```

当读到了'}'，则会在局部变量栈中持续退栈，直至第一个'{'也退栈，该过程即为局部变量的释放。

5、四元式转汇编

对于四元式(op, num1, num2 ,result):

当 op == =，若 num1 为数，则操作为

```
li    $t0   2
sw   $t0   -24($fp)
```

若 num2 为变量，则先把 num2 移动到寄存器，再进行赋值操作：

```
sw   $t0 -8($fp)
li    $t0   2
sw   $t0   -24($fp)
```

当 op == + - * / ：

1、若 num1、num2 为数，则只需要再分配一个寄存器，将 num1 的值存储进寄存器，op $1 num2

2、若 num1、num2 一个为数，一个为变量，则为变量分配寄存器，Op $1 num2，并置 b1.pos = 0，表示该变量的值只存在于内存中。记录新变量的属性。

3、若 num1、num2 都为变量，则为这两个变量都分配寄存器，然后，Op $1 $2 同时置 b11.pos = 0，记录新变量的属性。

相关代码：

```
//    add, a, b, c
void addasm() {
    int addr1, addr2, addr3;
    if (isdigit(midcode[mi].var1[0]) || midcode[mi].var1[0] == '-' || midcode[mi].var1[0] ==
'+') {
        rsT << "\t\tli\t$t0\t" << midcode[mi].var1 << endl;
    }
    else {
        addr1 = varaddr(midcode[mi].var1);
        if (isglob)
            rsT << "\t\tlw\t$t0\t" << addr1 << "($t9)" << endl;
        else
            rsT << "\t\tlw\t$t0\t" << addr1 << "($fp)" << endl;
    }
    if (isdigit(midcode[mi].var2[0]) || midcode[mi].var2[0] == '-' || midcode[mi].var2[0] ==
'+') {
        rsT << "\t\tli\t$t1\t" << midcode[mi].var2 << endl;
    }
    else {
        addr2 = varaddr(midcode[mi].var2);
        if (isglob)
            rsT << "\t\tlw\t$t1\t" << addr2 << "($t9)" << endl;
        else
            rsT << "\t\tlw\t$t1\t" << addr2 << "($fp)" << endl;
    }
    addr3 = varaddr(midcode[mi].var3);
    rsT << "\t\tadd\t$t0\t$t0\t$t1" << endl;
    if (isglob)
        rsT << "\t\tsw\t$t0\t" << addr1 << "($t9)" << endl;
    else
        rsT << "\t\tsw\t$t0\t" << addr3 << "($fp)" << endl;
}

//    sub, a, b, c
void subasm() {
    int addr1, addr2, addr3;
    if (isdigit(midcode[mi].var1[0]) || midcode[mi].var1[0] == '-' || midcode[mi].var1[0] ==
'+') {
```

```
        rsT << "\t\tli\t$t0\t" << midcode[mi].var1 << endl;
    }
    else {
        addr1 = varaddr(midcode[mi].var1);
        if (isglob)
            rsT << "\t\tlw\t$t0\t" << addr1 << "($t9)" << endl;
        else
            rsT << "\t\tlw\t$t0\t" << addr1 << "($fp)" << endl;
    }
    if (isdigit(midcode[mi].var2[0]) || midcode[mi].var2[0] == '-' || midcode[mi].var2[0] ==
'+') {
        rsT << "\t\tli\t$t1\t" << midcode[mi].var2 << endl;
    }
    else {
        addr2 = varaddr(midcode[mi].var2);
        if (isglob)
            rsT << "\t\tlw\t$t1\t" << addr2 << "($t9)" << endl;
        else
            rsT << "\t\tlw\t$t1\t" << addr2 << "($fp)" << endl;
    }
    addr3 = varaddr(midcode[mi].var3);
    rsT << "\t\tsub\t$t0\t$t0\t$t1" << endl;
    if (isglob)
        rsT << "\t\tsw\t$t0\t" << addr3 << "($t9)" << endl;
    else
        rsT << "\t\tsw\t$t0\t" << addr3 << "($fp)" << endl;
}

//    mul, a, b, c
void mulasm() {
    int addr1, addr2, addr3;
    if (isdigit(midcode[mi].var1[0]) || midcode[mi].var1[0] == '-' || midcode[mi].var1[0] ==
'+') {
        rsT << "\t\tli\t$t0\t" << midcode[mi].var1 << endl;
    }
    else {
        addr1 = varaddr(midcode[mi].var1);
        if (isglob)
            rsT << "\t\tlw\t$t0\t" << addr1 << "($t9)" << endl;
        else
            rsT << "\t\tlw\t$t0\t" << addr1 << "($fp)" << endl;
    }
    if (isdigit(midcode[mi].var2[0]) || midcode[mi].var2[0] == '-' || midcode[mi].var2[0] ==
'+') {
```

```
        rsT << "\t\tli\t$t1\t" << midcode[mi].var2 << endl;
    }
    else {
        addr2 = varaddr(midcode[mi].var2);
        if (isglob)
            rsT << "\t\tlw\t$t1\t" << addr2 << "($t9)" << endl;
        else
            rsT << "\t\tlw\t$t1\t" << addr2 << "($fp)" << endl;
    }
    addr3 = varaddr(midcode[mi].var3);
    rsT << "\t\tmul\t$t0\t$t0\t$t1" << endl;
    if (isglob)
        rsT << "\t\tsw\t$t0\t" << addr3 << "($t9)" << endl;
    else
        rsT << "\t\tsw\t$t0\t" << addr3 << "($fp)" << endl;
}

//    div, a, b, c
void divasm() {
    int addr1, addr2, addr3;
    if (isdigit(midcode[mi].var1[0]) || midcode[mi].var1[0] == '-' || midcode[mi].var1[0] ==
'+') {
        rsT << "\t\tli\t$t0\t" << midcode[mi].var1 << endl;
    }
    else {
        addr1 = varaddr(midcode[mi].var1);
        if (isglob)
            rsT << "\t\tlw\t$t0\t" << addr1 << "($t9)" << endl;
        else
            rsT << "\t\tlw\t$t0\t" << addr1 << "($fp)" << endl;
    }
    if (isdigit(midcode[mi].var2[0]) || midcode[mi].var2[0] == '-' || midcode[mi].var2[0] ==
'+') {
        rsT << "\t\tli\t$t1\t" << midcode[mi].var2 << endl;
    }
    else {
        addr2 = varaddr(midcode[mi].var2);
        if (isglob)
            rsT << "\t\tlw\t$t1\t" << addr2 << "($t9)" << endl;
        else
            rsT << "\t\tlw\t$t1\t" << addr2 << "($fp)" << endl;
    }
    addr3 = varaddr(midcode[mi].var3);
    rsT << "\t\tdiv\t$t0\t$t0\t$t1" << endl;
```

```
    if (isglob)
        rsT << "\t\tsw\t$t0\t" << addr3 << "($t9)" << endl;
    else
        rsT << "\t\tsw\t$t0\t" << addr3 << "($fp)" << endl;
}
```

if、while 通过对应 j、jnz、jne、jg、jge、jl、jle 等汇编指令即可实现。

五、测试样例

测试文件：

```
int get_max(int a,int b)
{
        if(a>b)
        {
            return a;
        }
        else
        {
            return b;
         }
}

int main()
{
        int c;
        int d;
        int e;
        c=1;
        d=2;
        e=get_max(c,d);
        return 0;
}
```

中间代码文件：

```
int,      ,        ,      a;
int,      ,        ,      b;
func,   int,       ,      program;
para,   int,       ,      a;
para,   int,       ,      b;
para,   int,       ,      c;
int,      ,        ,      i;
int,      ,        ,      j;
=,      0,         ,      i;
+,      b,       c,       $_0;
>,      a,       $_0,       ;
jne,      ,        ,      _LABEL_0;
*,      b,       c,       $_1;
+,      $_1,     1,       $_2;
+,      a,       $_2,     $_3;
=,      $_3,       ,      j;
jmp,      ,        ,      _LABEL_1;
lab:,     ,        ,      _LABEL_0;
=,      a,         ,      j;
lab:,     ,        ,      _LABEL_1;
<=,     i,       100,       ;
jne,      ,        ,      _LABEL_3;
*,      j,       2,       $_4;
=,      $_4,       ,      i;
jmp,      ,        ,      _LABEL_1;
lab:,     ,        ,      _LABEL_3;
ret,      ,        ,      i;
end,      ,        ,      program;


func,   int,       ,      demo;
para,   int,       ,      a;
+,      a,       2,       $_0;
=,      $_0,       ,      a;
*,      a,       2,       $_1;
=,      $_1,       ,      a;
ret,      ,        ,      a;
end,      ,        ,      demo;


func,     ,        ,      main;
int,      ,        ,      a;
int,      ,        ,      b;
int,      ,        ,      c;
=,      3,         ,      a;
=,      4,         ,      b;
=,      2,         ,      c;
fupa,     ,        ,      c;
call,   demo,      ,      $_0;
fupa,     ,        ,      a;
fupa,     ,        ,      b;
fupa,     ,        ,      $_0;
call,   program,           ,      $_1;
=,      $_1,       ,      a;
ret,      ,        ,        ;
end,      ,        ,      main;
```

汇编指令：

```
.text
    ori   $fp  $sp  0
    li    $t9  0x7fffeffc#global stack bottom
    li    $t8  0x10010000   #save word
    li    $t0  0    #a
    sw    $t0  ($sp)
    subi  $sp  $sp  4
    li    $t0  0    #a
```

```
        sw   $t0  ($sp)
        subi $sp  $sp  4
        j    __main
program:
        #Save Register
        sw   $fp  ($sp)
        add  $fp  $sp  $0
        subi $sp  $sp  4
        sw   $ra  ($sp)
        subi $sp  $sp  4
        #Save Register Done!
        li   $t0  0    #$_0
        sw   $t0  ($sp)
        subi $sp  $sp  4
        li   $t0  0    #$_1
        sw   $t0  ($sp)
        subi $sp  $sp  4
        li   $t0  0    #$_2
        sw   $t0  ($sp)
        subi $sp  $sp  4
        li   $t0  0    #$_3
        sw   $t0  ($sp)
        subi $sp  $sp  4
        li   $t0  0    #$_4
        sw   $t0  ($sp)
        subi $sp  $sp  4
        li   $t0  0    #program
        sw   $t0  ($sp)
        subi $sp  $sp  4
        li   $t0  0    #program
        sw   $t0  ($sp)
        subi $sp  $sp  4
        li   $t0  0
        sw   $t0  -28($fp)
        lw   $t0  8($fp)
        lw   $t1  4($fp)
        add  $t0  $t0  $t1
        sw   $t0  -8($fp)
        lw   $t0  12($fp)
        lw   $t1  -8($fp)
        slt  $t0  $t1  $t0
        bne  $t0  1    _LABEL_0
        lw   $t0  8($fp)
        lw   $t1  4($fp)
```

```
        mul $t0  $t0  $t1
        sw  $t0  -12($fp)
        lw  $t0  -12($fp)
        li  $t1  1
        add $t0  $t0  $t1
        sw  $t0  -16($fp)
        lw  $t0  12($fp)
        lw  $t1  -16($fp)
        add $t0  $t0  $t1
        sw  $t0  -20($fp)
        lw  $t0  -20($fp)
        sw  $t0  -32($fp)
        j   _LABEL_1
_LABEL_0:
        lw  $t0  12($fp)
        sw  $t0  -32($fp)
_LABEL_1:
_LABEL_2:
        lw  $t0  -28($fp)
        li  $t1  100
        slt $t0  $t1  $t0
        li  $t1  1
        sub $t0  $t1  $t0
        bne $t0  1    _LABEL_3
        lw  $t0  -32($fp)
        li  $t1  2
        mul $t0  $t0  $t1
        sw  $t0  -24($fp)
        lw  $t0  -24($fp)
        sw  $t0  -28($fp)
        j   _LABEL_2
_LABEL_3:
        lw  $v0  -28($fp)
        j   __FEND_LAB_1
__FEND_LAB_1:
        lw  $ra  -4($fp)
        add $sp  $fp  $0
        lw  $fp  ($fp)
        jr  $ra
demo:
        #Save Register
        sw  $fp  ($sp)
        add $fp  $sp  $0
        subi $sp  $sp  4
```

```
        sw   $ra  ($sp)
        subi $sp  $sp  4
        #Save Register Done!
        li   $t0  0    #$_0
        sw   $t0  ($sp)
        subi $sp  $sp  4
        li   $t0  0    #$_1
        sw   $t0  ($sp)
        subi $sp  $sp  4
        lw   $t0  4($fp)
        li   $t1  2
        add  $t0  $t0  $t1
        sw   $t0  -8($fp)
        lw   $t0  -8($fp)
        sw   $t0  4($fp)
        lw   $t0  4($fp)
        li   $t1  2
        mul  $t0  $t0  $t1
        sw   $t0  -12($fp)
        lw   $t0  -12($fp)
        sw   $t0  4($fp)
        lw   $v0  4($fp)
        j    __FEND_LAB_2
__FEND_LAB_2:
        lw   $ra  -4($fp)
        add  $sp  $fp  $0
        lw   $fp  ($fp)
        jr   $ra
__main:
        #Save Register
        sw   $fp  ($sp)
        add  $fp  $sp  $0
        subi $sp  $sp  4
        sw   $ra  ($sp)
        subi $sp  $sp  4
        #Save Register Done!
        li   $t0  0    #$_0
        sw   $t0  ($sp)
        subi $sp  $sp  4
        li   $t0  0    #$_1
        sw   $t0  ($sp)
        subi $sp  $sp  4
        li   $t0  0    #main
        sw   $t0  ($sp)
```

```
        subi $sp  $sp  4
        li    $t0  0    #main
        sw    $t0  ($sp)
        subi $sp  $sp  4
        li    $t0  0    #main
        sw    $t0  ($sp)
        subi $sp  $sp  4
        li    $t0  3
        sw    $t0  -16($fp)
        li    $t0  4
        sw    $t0  -20($fp)
        li    $t0  2
        sw    $t0  -24($fp)
        lw    $t0  -24($fp)
        sw    $t0  ($sp)
        subi $sp  $sp  4
        jal   demo
        nop
        sw    $v0  -8($fp)
        lw    $t0  -16($fp)
        sw    $t0  ($sp)
        subi $sp  $sp  4
        lw    $t0  -20($fp)
        sw    $t0  ($sp)
        subi $sp  $sp  4
        lw    $t0  -8($fp)
        sw    $t0  ($sp)
        subi $sp  $sp  4
        jal   program
        nop
        sw    $v0  -12($fp)
        lw    $t0  -12($fp)
        sw    $t0  -16($fp)
        j     __FEND_LAB_3
__FEND_LAB_3:
        lw    $ra  -4($fp)
        add  $sp  $fp  $0
        lw    $fp  ($fp)
        li    $v0  10
        syscall
```

分析：

将以上 mips 代码在 mars 环境中运行，结果如下：

0x00400294	0x20010004	addi $1,$0,0x00000004	156:	subi	$sp	$sp	4
0x00400298	0x03a1e822	sub $29,$29,$1					
0x0040029c	0x0c100061	jal 0x00400184	157:	jal	demo		
0x004002a0	0x00000000	nop	158:	nop			
0x004002a4	0xafc2fff8	sw $2,0xfffffff8($30)	159:	sw	$v0	-8($fp)	
0x004002a8	0x8fc8fff0	lw $8,0xfffffff0($30)	160:	lw	$t0	-16($fp)	
0x004002ac	0xafa80000	sw $8,0x00000000($29)	161:	sw	$t0	($sp)	
0x004002b0	0x20010004	addi $1,$0,0x00000004	162:	subi	$sp	$sp	4
0x004002b4	0x03a1e822	sub $29,$29,$1					
0x004002b8	0x8fc8ffec	lw $8,0xffffffec($30)	163:	lw	$t0	-20($fp)	
0x004002bc	0xafa80000	sw $8,0x00000000($29)	164:	sw	$t0	($sp)	
0x004002c0	0x20010004	addi $1,$0,0x00000004	165:	subi	$sp	$sp	4
0x004002c4	0x03a1e822	sub $29,$29,$1					
0x004002c8	0x8fc8fff8	lw $8,0xfffffff8($30)	166:	lw	$t0	-8($fp)	
0x004002cc	0xafa80000	sw $8,0x00000000($29)	167:	sw	$t0	($sp)	
0x004002d0	0x20010004	addi $1,$0,0x00000004	168:	subi	$sp	$sp	4
0x004002d4	0x03a1e822	sub $29,$29,$1					
0x004002d8	0x0c10000e	jal 0x00400038	169:	jal	program		
0x004002dc	0x00000000	nop	170:	nop			
0x004002e0	0xafc2fff4	sw $2,0xfffffff4($30)	171:	sw	$v0	-12($fp)	
0x004002e4	0x8fc8fff4	lw $8,0xfffffff4($30)	172:	lw	$t0	-12($fp)	
0x004002e8	0xafc8fff0	sw $8,0xfffffff0($30)	173:	sw	$t0	-16($fp)	
0x004002ec	0x081000bc	j 0x004002f0	174:	j	__FEND_LAB_3		
0x004002f0	0x8fdffffc	lw $31,0xfffffffc($...	176:	lw	$ra	-4($fp)	
0x004002f4	0x03c0e820	add $29,$30,$0	177:	add	$sp	$fp	$0
0x004002f8	0x8fde0000	lw $30,0x00000000($...	178:	lw	$fp	($fp)	
0x004002fc	0x2402000a	addiu $2,$0,0x00000...	179:	li	$v0	10	
0x00400300	0x0000000c	syscall	180:	syscall			

经过分析，程序运行过程及寄存器的值与预期是一致的。

六、总结

这次课程设计写了一个月了，通过这次课程设计，让我对编译器的工作原理有了较深的认识。同时也发现了自己程序功力的不足。该编译器还是有挺多不是很满意的地方。比如：

1、对测试程序格式要求较高，在编写的过程中，不是所有地方都记得加上空格是’\t’,’\r’,’\n’的处理，导致测试程序格式要求较高。

2、while 和 if（或反之）连用，即 while{if{...}}，可能会出现地址 L1: L2:这样的情况。

总的来说，这次主要自己设计编译器虽然不是很完善，但还是较为满意的。