第六章作业

题目：

```
有如下C语言程序段，
int prime(int n){
    int sum = 0;
    int i,j,flag = 1;
    for(i = 2; i<=n; i++){
        flag = 1;
        for(j = 2; j*j <= i; j++){
            if(i%j == 0){
                flag = 0;
                break;
            }
        }
        if(flag == 1){
            sum ++;
        }
    }
    return sum;
}
int main(){
    int res = n = 0;
    do{
        res = prime(n);
    }while(n < 10);

return 0;
}
```

问题:

(1) 请根据C语言的规范，给出do-while语句的文法描述；

(2) 根据（1）中的文法，给出do-while的语法制导定义和语法制导翻译方案；

(3) 请给出上述程序段的AST（可选）；

(4) 请给出上述程序段两个函数对应的四元式表示；

(5) 如果实行的一遍编译处理，根据标号表的处理过程，填写下表。（标号的命名根据标号出现顺序，依次命名为BH1,BH2,¼）

标号名	地址	用到的返填地址
BH1		
BH2		

解答：（1） do-while的文法描述如下：（由于如果文法太过庞大，这里只根据do while具体内容给出部分文法）,同时给出说明类型语句，为了后面语法制导翻译的符号表

```
//文法
dowhileStatement -> "do" stat "while" "(" exprs ")" ";"

exprs ->  expr
          |expr ',' exprs;

stat -> expressionStatement

expr -> expr ('*'|'/'|'%') expr
     | expr ('+'|'-')  expr
     | expr ('>'|'<'|'<='|'>=') expr
     | expr ('=='|'!=') expr
     | expr Assignment_operator expr
     | functionCall
     | IntegerConstant
     | CharConstant
     | FloatConstant
     | Identifier
     | StringLiteral
     |'(' expr ')'
     ;

declarator -> Identifier
#VariableDeclarator
            |declarator '[' expr ']'
#ArrayDeclarator
            |declarator '(' arguments? ')'
#FunDeclarator
            ;
//额外的词法

Assignment_operator -> '='|'*='|'/='|'%='|'+='|'-
='|'<<='|'>>='|'&='|'^='|'|=';

IntegerConstant -> (([1-9][0-9]*)|('0'[0-7]*)|('0'[xX][0-9a-fA-F]*))([uU]?
('ll'|'LL')?|('ll'|'LL')?[uU]?|[uU]?[Ll]?|[Ll]?[Uu]?);

CharConstant -> ([cLuU]?)('\''.?'\'');

FloatConstant -> DECIFLOAT
             | HEXAFLOAT
             ;

fragment
DECIFLOAT -> ([0-9]+'.'?[0-9]*)([eE][+-]?[0-9]+)?([flFL])?;

fragment
HEXAFLOAT -> ('0'[xX][0-9a-fA-F]+'.'?[0-9a-fA-F]*)([pP][+-]?[0-9]+)?
([flFL])?;

StringLiteral -> (([uUL]?)|('u8'))('"'(.*?)'"');

Identifier -> ([_a-zA-Z][_a-zA-Z0-9]*);
```

(2) do-while语法制导定义和语法制导翻译方案如下:

文法	语法制导翻译
do-whileStatement -> "do" stat "while" "(" exprs ")" ";"	{exprs.true=newlabel; do-whileStatement.code = Gen(exprs.true':')|stat.code|exprs.code|Gen(jt,_,_,exprs.true)}
exprs -> expr	{exprs.code = expr.code; exprs.type = expr.type}
exprs1 -> expr ',' exprs2	{exprs1.code = expr.code | exprs2.code; exprs1.type = expr.type}
stat -> expressionStatement	{stat.code = expressionment.code}
expr -> expr1 ('*'|'/') expr2	{expr.place = newtemp; expr.code = Gen(op,expr1.code,expr2.code,expr.place); contype1 = (expr1.type = interger and expr2.type = float)or(expr1.type = float and expr2.type = interger) or (expr1.type = float and expr2.type = float) contype2 = (expr1.type = interger and expr2.type = interger) expr.type = if contype1||contype2 then if contype2 then interger else float else type_error } 【解释：操作数必须是interger或者float类型】
expr -> expr1 '%' expr2	{expr.place = newtemp; expr.code = Gen(op,expr1.code,expr2.code,expr.place); if expr1.type = interger and expr2.type = interger then interger else type_error 【解释：操作数必须是integer类型】
expr -> expr1 ('+'|'-') expr2	{expr.place = newtemp; expr.code = Gen(op,expr1.code,expr2.code,expr.place); contype1 = (expr1.type = interger and expr2.type = float)or(expr1.type = float and expr2.type = interger) or (expr1.type = interger and expr2.type = interger) contype2 = (expr1.type = float and expr2.type = float) expr.type = if contype1||contype2 then if contype1 then interger else float else type_error 【解释：操作数必须是interger或者是float，而且如果有一个是interger，那么表达式结果是interger类型】

expr -> expr1 ('<'|'>'|'>='|'<=') expr2	{expr.place = newtemp;
	expr.code = Gen(op,expr1.code,expr2.code,expr.place);
	if (expr1.type = interger or expr1.type = float) and (expr2.type = interger or expr2.type = float) then boolean else type_error
	【解释：操作数必须是integer或者float类型】

expr -> expr1 ('=='|'!=') expr2	同上
expr -> expr1 Assignment_operator expr2	{expr2.place = newtemp; expr.code = Gen(Assignment_operator),expr2.place,_,entry(expr1); expr.type = expr1.type}

文法	语法制导翻译
declarator -> Identifier	{declarator .type = lookup(Identifier.entry)}
declarator -> declarator1 '[' expr ']'	{declarator.type = if expr.type = interger then declarator1.ele.type else type_error}
declarator -> declarator1 '(' arguments? ')'	{declarator.type = declarator1.type}
expr -> IntegerConstant	{expr.type = interger}
expr -> CharConstant	{expr.type = char}
expr -> FloatConstant	{expr.type = float}
expr -> StringLiteral	{expr.type = string}

(3) 由于这里的ast太过庞大，所以就不展示了。

(4) 这里生成的四元式如下:

```
(proc,,,prime)
(pop,,,n)
(=,0,,sum)
(=,1,,flag)
(=,2,,i)
(label,,,%1)
(<=,i,n,%3)
(jf,%3,,%2)
(=,1,,flag)
(=,2,,j)
(label,,,%4)
(*,j,j,%7)
(<=,%7,i,%6)
(jf,%6,,%5)
(%,i,j,%9)
(==,%9,0,%8)
(jf,%8,,%10)
(=,0,,flag)
(label,,,%10)
(++,j,,%11)
(jmp,,,%4)
(label,,,%5)
(==,flag,1,%12)
(jf,%12,,%13)
(++,sum,,%14)
(label,,,%13)
(++,i,,%15)
(jmp,,,%1)
(label,,,%2)
(ret,sum,,)
(endp,,,prime)
(proc,,,main)
```

```
(=,0,,n)
(=,n,,res)
(label,,%18)
(push,,,n)
(call,prime,,%16)
(=,%16,,res)
(<,n,10,%17)
(jt,%17,,%18)
(ret,0,,)
(endp,,,main)
```

(5) 如果实行的一遍编译处理，根据标号表的处理过程，填写下表。（标号的命名根据标号出现顺序，依次命名为BH1,BH2,...）【题外话：如果我跳转语句的标号不是相应的地址，而是设置一个单独的临时变量作为标签并记录下来，就可以不采用拉链返填技术】

这里的地址是上面代码的标号。

标号名	地址	用到的返填地址
BH1（%2）	30	8
BH2（%5）	23	14
BH3（%10）	20	17
BH4（%13）	27	24