《编译技术》课程设计文 档

学号：\_\_\_\_\_16061040\_\_\_\_\_\_

姓名：\_\_\_\_\_\_\_王科翔\_\_\_\_\_\_\_

2018年11月17日

## 一．需求说明

### 1．文法说明

1. ＜加法运算符＞ ::= +｜-

eg: 略

analysis:略

2. ＜乘法运算符＞ ::= \*｜/

eg: 略

analysis:略

3. ＜关系运算符＞ ::= <｜<=｜>｜>=｜!=｜==

eg: 略

analysis:略

4. ＜字母＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

eg: 略

analysis:略

5. ＜数字＞ ::= ０｜＜非零数字＞

eg: 略

analysis:略

6. ＜非零数字＞ ::= １｜．．．｜９

eg: 略

analysis:略

7. ＜字符＞ ::= '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'

eg: 略

analysis:略

8. ＜字符串＞ ::= "｛十进制编码为32,33,35-126的ASCII字符｝"

eg: dfshjsdfhjksdfFFFDFDSii！ ;456sdfsd546

analysis:包括的符号有“ !#$%&'()\*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^\_`abcdefghijklmnopqrstuvwxyz{|}~”，需要注意的是不包括双引号

9. ＜程序＞ ::= ［＜常量说明＞］［＜变量说明＞］{＜有返回值函数定义＞|＜无返回值函数定义＞}＜主函数＞

eg: const int constnumber = 1; //［＜常量说明＞］

int varnumber; //［＜变量说明＞］

void print(){ // ＜有返回值函数定义＞

printf("\n");

}

void main(){ // ＜主函数＞

print();

}

analysis:最多只能有一个块用来声明常量，一个块用来声明变量。 可以没有或有一个或多个有无返回值的函数定义。

并且这三个快的顺序固定。

10. ＜常量说明＞ ::= const＜常量定义＞;{ const＜常量定义＞;}

eg: const int constnumber0 = 0;

const int constnumber1 = 1;

analysis:可以有多个或者一个常量的定义。每定义前是const，后面是“;”。

11. ＜常量定义＞ ::= int＜标识符＞＝＜整数＞{,＜标识符＞＝＜整数＞}

| char＜标识符＞＝＜字符＞{,＜标识符＞＝＜字符＞}

eg: const int constnumber0 = 10, constnumber1 = 11, constnumber2 = 12;

const char constchar0 = 'd', char constchar1 = 'd', char constchar2 = 'd';

analysis:const分为int和char两种，每一条常量定义可以定义一种多个常量，每个之间逗号间隔。

12. ＜无符号整数＞ ::= ＜非零数字＞｛＜数字＞｝｜０

eg\_0: 0

eg\_1: 50

analysis:无符号整数无前导0。

13. ＜整数＞ ::= ［＋｜－］＜无符号整数＞

eg: +5

analysis:整数可以有或无符号，最多一个。

14. ＜标识符＞ ::= ＜字母＞｛＜字母＞｜＜数字＞｝

eg: wangkexiang666

analysis:标识符只包括字母或者数字，第一个必须是字母。

15. ＜声明头部＞ ::= int＜标识符＞|char＜标识符＞

eg: int getnumber

analysis:函数声明头部只能是int或者char，且必须有且仅有一个。

16. ＜变量说明＞ ::= ＜变量定义＞;{＜变量定义＞;}

eg: int declareint;

char declarechar;

analysis:至少有一个变量定义，每个以分号结尾。

17. ＜变量定义＞ ::= ＜类型标识符＞(＜标识符＞|＜标识符＞'['＜无符号整数＞']'){,(＜标识符＞|＜标识符＞'['＜无符号整数＞']') } //＜无符号整数＞表示数组元素的个数，其值需大于0

eg: int declareint, declareIntArray0[8];

char declarechar, declareCharArray0[9];

int declareIntArray[5];

char declareCharArray[6];

analysis:一条定义中可以同时定义数组和单个变量。

18. ＜类型标识符＞ ::= int | char

eg: int

analysis:类型只包括int和char。

19. ＜有返回值函数定义＞ ::= ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'

eg: int getnumber(int n){

return 0;

}

analysis: ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'三个部分四个符号缺一不可，顺序固定。

20. ＜无返回值函数定义＞ ::= void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'

eg: void operate(int n){

n = n + 1;

}

analysis:void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'一个标识符三个部分四个符号缺一不可，顺序固定。

21. ＜复合语句＞ ::= ［＜常量说明＞］［＜变量说明＞］＜语句列＞

eg: const int constInt = 0; // ＜常量说明＞

int intint; // ＜变量说明＞

intint = 0; // ＜语句列＞

intint = 1;

analysis:最多只能有一个块用来声明常量，一个块用来声明变量。 可以没有或有一个或多个语句。

并且这三个快的顺序固定。

22. ＜参数表＞ ::= ＜类型标识符＞＜标识符＞{,＜类型标识符＞＜标识符＞}| ＜空＞

eg\_0:

eg\_1: int n

eg\_2: int a, int b

eg\_3: iut a, char b

analysis:可以有或没有参数，参数只能是int或者char

23. ＜主函数＞ ::= void main'('')' '{'＜复合语句＞'}'

eg: void main(){

int n;

}

analysis:只能是void main(){}这样，且返回值只能是void。

24. ＜表达式＞ ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞} //[+|-]只作用于第一个<项>

eg\_0: +a-b+a\*b

eg\_1: a-b+a\*b

analysis:开头可以有一个或没有符号

25. ＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

eg: a\*b/c

analysis:第一个一定是一个因子后面可以加其他因子，之间用乘除相连。

26. ＜因子＞ ::= ＜标识符＞｜＜标识符＞'['＜表达式＞']'｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞|'('＜表达式＞')'

eg\_0: a[b]

eg\_1: 5

eg\_2: d

eg\_3: add(a,b)

eg\_4: (a+b\*c)

analysis:因子没有符号

27. ＜语句＞ ::= ＜条件语句＞｜＜循环语句＞| '{'＜语句列＞'}'｜＜有返回值函数调用语句＞;

|＜无返回值函数调用语句＞;｜＜赋值语句＞;｜＜读语句＞;｜＜写语句＞;｜＜空＞;｜＜返回语句＞;

eg\_0: if(a == 0){

return a;

} else{

return 0;

}

eg\_1: for(i = 0;i < 10; i = i + 1){

a = a + 1;

}

eg\_2: {

i = i + 1;

}

eg\_3: add(a, b);

eg\_4: merge(a, b, c);

eg\_5: a = b;

eg\_6: scanf(a, b);

eg\_7: printf("%d", a);

eg\_8: ;

eg\_9: return (0);

eg\_10: return ;

analysis:条件语句和循环语句没有分号，返回语句可以直接热return ;

28. ＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞|＜标识符＞'['＜表达式＞']'=＜表达式＞

eg: a = b;

c[4] = a\*b;

analysis:只可以给变量或者数组的某个值赋值一个表达式。

29. ＜条件语句＞ ::= if '('＜条件＞')'＜语句＞［else＜语句＞］

eg:

analysis:可以没有else的处理

30. ＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞ //表达式为0条件为假，否则为真

eg\_0: a==b

eg\_1: a

analysis:可以是一个判罚关系的表达式，也可以判断一个值是不是0。

31. ＜循环语句＞ ::= do＜语句＞while '('＜条件＞')' |for'('＜标识符＞＝＜表达式＞;＜条件＞;＜标识符＞＝＜标识符＞(+|-)＜步长＞')'＜语句＞

eg\_0: do{

i = i + 1;

} while(i < 10)

eg\_1: for(i = 0;i < 10; i = i + 1){

a = a + 1;

}

analysis:对于此文法，for内的三个块必须都存在，不为空。

32. ＜步长＞::= ＜无符号整数＞

eg\_0: 0

eg\_1: 50

analysis:这里不支持为char的情况

33. ＜有返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'

eg: add(a\*c,b)

analysis:略

34. ＜无返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'

eg: merge(a,b)

analysis:略

35. ＜值参数表＞ ::= ＜表达式＞{,＜表达式＞}｜＜空＞

eg: a\*b,c

analysis:可以有多个或者一个或者没有，每两个逗号间隔

36. ＜语句列＞ ::=｛＜语句＞｝

eg: a = a + 1;

b = a;

analysis:语句列甚至可以没有语句。

37. ＜读语句＞ ::= scanf '('＜标识符＞{,＜标识符＞}')'

eg: scanf(a,b);

analysis:和一般的不一样，功能缩减。

38. ＜写语句＞ ::= printf'('＜字符串＞,＜表达式＞')'|printf '('＜字符串＞')'|printf '('＜表达式＞')'

eg: printf("%d", a)

analysis:功能有所缩减。写语句中，字符串原样输出，单个字符类型的变量或常量输出字符，其他表达式按整型输出

39. ＜返回语句＞ ::= return['('＜表达式＞')']

eg: return (0)

analysis:必须有括号要不然不返回任何值

附加说明：

（1）char类型的变量或常量，用字符的ASCII码对应的整数参加运算

（2）标识符区分大小写字母

（3）写语句中，字符串原样输出，单个字符类型的变量或常量输出字符，其他表达式按整型输出

（4）数组的下标从0开始

### 2．目标代码说明

生成的目标代码为mips32汇编指令，以下列举了一部分可能用到的指令，后续可能会继续扩展

|  |  |  |
| --- | --- | --- |
| 指令 | 指令格式 | 指令意义 |
| syscall | syscall | 根据c0参数不同功能不同 |
| li | li s1,100 | s1 = 100 |
| la | la $t1, array1 | $t1 = array1的首地址 |
| add | add s1,s2,s3 | s1 = s2 + s3 |
| sub | sub s1,s2,s3 | s1 = s2 – s3 |
| mult | mult s1,s2 | hi,lo = s1\*s2 |
| div | div s1,s2 | lo = s1/s2, hi = s1mod s2 |
| addi | addi s1,s2,100 | s1 = s2 + 100 |
| subi | subi s1,s2,100 | s1 = s2 - 100 |
| mfhi | mfhi | a = [hi] |
| mflo | mflo a | a = [lo] |
| move | move s1 s2 | s1 = s2 |
| lw | lw $t2,4($t1) | $t2 = [$t1+1] |
| sw | sw $t2,4($t1) | [$t1+1] = $t2 |
| beq | beq s1,s2,label | if(s1==s2) goto label |
| bne | bne s1,s2,label | if(s1!=s2) goto label |
| blt | blt s1,s2,label | if(s1<s2 ) goto label |
| bgt | bgt s1,s2,label | if(s1>s2) goto label |
| ble | ble s1,s2,label | if(s1<=s2) goto label |
| bge | bge s1,s2,label | if(s1>=s2) goto label |
| j | j label | goto label |
| jr | jr $ra | goto [$ra] |
| jal | jal label | goto and link label |

### 3. 优化方案\*

1. 窥孔优化

对于目标代码中出现的如下结构的连续的两条指令:

sw $x y($sp)

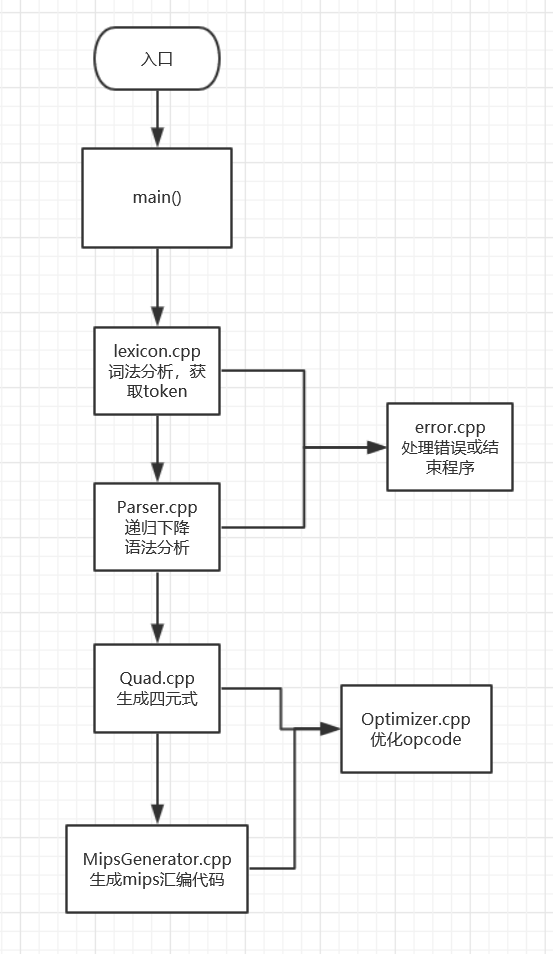
lw $x y($sp)

显然第二条指令是多余的，可以删除。

1. 常数合并，对于两个操作数均为常数的中间代码，优化时可以直接算出结果
2. 常量替换，在生成中间代码时，就将所有的常量名替换成其常量值。
3. 对每一个基本块，建立DAG图,消除公共子表达式
4. 恒等式的删除,比如 x = x + 0; x = x - 0; a[3] = a[3] + 0;
5. 临时变量的重复利用(一个函数内存在大量的临时变量只使用过一次就再也没有用的情况)针对基本块的重新规划
6. 将临时变量的起始地址放到$k0寄存器中,$k1寄存器存放临时变量的最大编号
7. 全局寄存器分配（引用计数或着色算法）
8. 数据流分析（通过活跃变量分析，或利用定义-使用链建网等方法建立冲突图）

## 二．详细设计

### 1．程序结构



### 2．类/方法/函数功能

|  |  |
| --- | --- |
| ErrorHandler |  |
| bool errorHaveOccured(); | 执行完毕后，判断编译有无错误 |
| void report(int lineCount,string currentLine, ErrorType errortype); | 报告错误 |

|  |  |
| --- | --- |
| CodeParser |  |
| void parseExpression(); | 分析表达式 |
| void parseTerm(); | 分析项 |
| void parseFactor(); | 分析因子 |
| void parseStatement(); | 分析语句 |
| void parseStatementSequence(); | 分析语句列 |
| void parseCompareStatement(); | 分析比教育局比较语句 |
| void parseFunction(); | 分析函数 |
| void parseConstDeclare(); | 分析常量声明 |
| void parseVarDeclare(); | 分析变量声明 |
| void parseVarAndFuncDeclare(); | 分析变量声明和函数声明 |
| void parseIf(); | 分析if语句 |
| void parseElse(); | 分析else语句 |
| void parseDoWhile(); | 分析dowhile语句 |
| void parseFor(); | 分析for语句 |
| void parseScanf(); | 分析scanf语句 |
| void parsePrintf(); | 分析printf语句 |
| void parseCode(); | 分析code语句 |
| void parseReturn(); | 分析return语句 |
| void jumpToToken(Token token); | 跳转到指定的token |

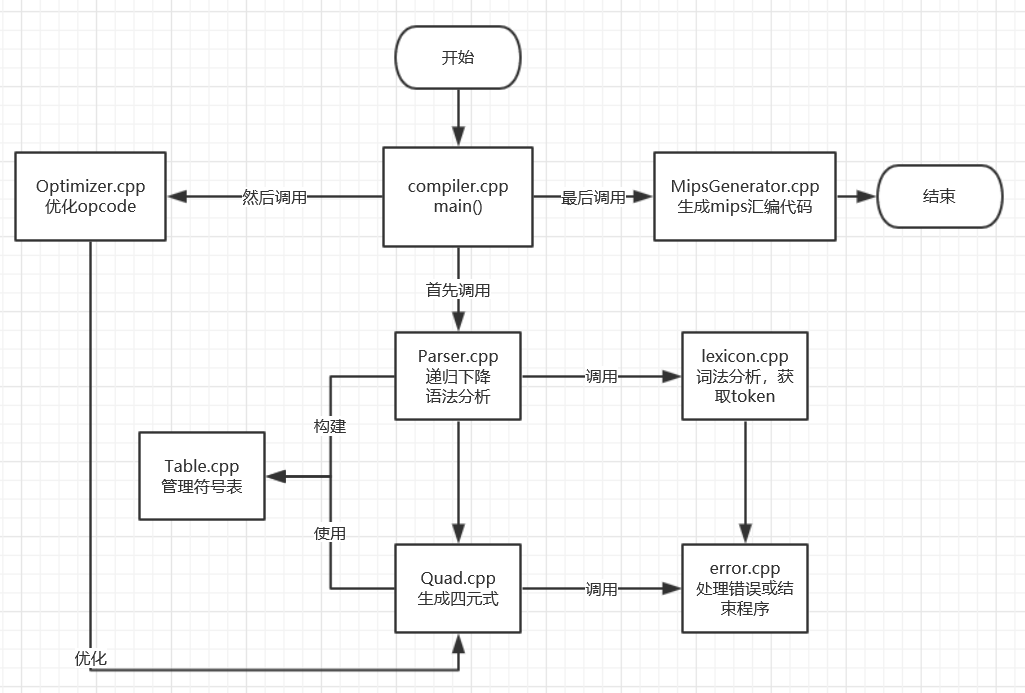
|  |  |
| --- | --- |
| Lexicon |  |
| Token nextToken(); | 获取下一个token |
| int getLineCount() const; | 获取当前行数 |
| void reportError(); | 此法错误报告 |
| void jumpToLineEnd(); | 跳转到行末 |
| void jumpToSentenceEnd(); | 跳转到句末 |
| char nextChar(); | 下一个字符 |
| inline bool isDefineChar(char c); | 判断是否为数字或者字母 |
| inline bool isCharInString(char c); | 判断是否为指定范围内的ascii |

|  |  |
| --- | --- |
| MipsGenerator |  |
| void generate() | 入口 |
| void initData(); | .data段 |
| void initRegs(); | 定义t寄存器 |
| void generateFunction(Function \*function); | 对每个函数定义块分析生成 |
| void generateAddSub(Function \*function, Quad \*quad, OPCode opCode); |  |
| void generateMultDiv(Function \*function, Quad \*quad, OPCode opCode); |  |
| void generateVar(Function \*function, Quad \*quad); |  |
| void generateArray(Function \*function, Quad \*quad); |  |
| void generateBranch(Function \*function, Quad \*quad, string branchName); |  |
| void generateBranchZ(Function \*function, Quad \*quad); |  |
| void generateJump(Function \*function, Quad \*quad); |  |
| void generateReturn(Function \*function, Quad \*quad); |  |
| void generateFunc(Function \*function, Quad \*quad, int offset); |  |
| void generateVoidFunc(Function \*function, Quad \*quad, int offset); |  |
| void generateScanf(Function \*function, Quad \*quad); |  |
| void generatePrintf(Function \*function, Quad \*quad); |  |
|  |  |
| string getReg(Function \*function, Quantity \*quantity, bool write = false, int temp = 0); | 获取已分配或未分配的寄存器 |
| Reg \* overflow(Function \*function); | 处理如果超过t7 |
| void allocateGloabal(Function \* function); | 分配局部变量和参数 |
| int getVarNumber(Function \* function); | 获得局部变量和参数的数目\*4（栈偏移量） |
| void writeBack(Function \* function); | 回写入栈 |
| void clearGlobalInTempRegs(Function \*functoin); | 清理分配如$t的全局变量 |
| void loadValue(Function \*function, Quad \*quad, string reg, int temp); | 根据类型生成取出值的mips |
| void loadValueGlobal(Function \*function, Quad \*quad, string reg, int temp); | 根据类型生成取出全局变量值的mips |
| void storeValue(Function \*function, Quad \*quad, string reg); | 回写入栈 |
| void moveToReg(Function \*functoin, Quantity \*value, string reg, int temp = 0); | 生成assign |
|  |  |
| void printCode(std::fstream &output); | 打印最终代码 |

|  |  |
| --- | --- |
| Optimizer |  |
| void optimize(string originalQuads ="", string newQuads = ""); | 优化入口，按基本快调用deg分析 |
| void printQuad(string filename); | 打印现有的四元式 |
| void deleteGlobal(); | 删除节点表中的全局变量 |
| Quantity \*traceNode(Quantity \*quantity); | 根据节点表追踪 |
| Quantity \*find(Quantity \*quantity); | 寻找量 |
| void dagAnalyze(QuadTable \*quadTable); | 按基本块进行dag分析 |

【描述各类/方法或函数的功能，以及关键算法】

### 3．调用依赖关系



### 4．符号表管理方案

class TableElement {

string name; // name

int addr; // address

int kind; // const or var or func

DataType dataType; // datatype

int value; // value

int number; // number of parameter or array elements

};

class Table

{

public:

Table();

~Table();

vector<TableElement> tableVector;

int top;

};

### 5．存储分配方案

|  |
| --- |
| 栈顶 |
| 上个函数栈底 |
| 局部变量区 |
| 参数区 |

函数运行栈从高地址向低地址分配，将栈顶地址保存在$sp，将栈底地址保存在$fp，所以如果f1调用f2，则f1的栈顶是f2的栈底。

运行栈则类似如下格式

|  |
| --- |
| 函数运行栈1 |
| 函数运行栈2 |
| 函数运行栈3 |
| ...... |

将全局量分配在.data段，即内存地址最低的区域。

### 6. 解释执行程序\*

难度3的C0文法不解释执行，而是生成mips汇编。

### 7. 四元式设计\*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 第一个 | 第二个 | 第三个 | 第四个 | 含义 | 备注 |
| OP\_VAR | VAR\_name | = | quantity |  | name = quantity |  |
| OP\_ARRAY | ARRAY\_name | [index] | = | quantity | name[index] = quantity |  |
| OP\_FUNC | FUNCCALL | func\_name | (parameters) |  | func\_name(parameters) |  |
| OP\_VOIDFUNC | VOIDCALL | func\_name | (parameters) |  | func\_name(parameters) |  |
| OP\_ADD | add\_1 | a | b |  | add\_1=a+b |  |
| OP\_SUB | sub\_1 | a | b |  | sub\_1=a-b |  |
| OP\_MULT | mult\_1 | a | b |  | mult\_1=a\*b |  |
| OP\_DIV | div\_1 | a | b |  | div\_1=a/b |  |
| OP\_BEQZ | BEQZ | a |  | 1 | beqz a, label\_1 |  |
| OP\_EQU | EQU | a | b | 1 | equ a, b, label\_1 | OP\_BRANCH同理 |
| OP\_RETURN | a |  |  |  | return(a) |  |
| OP\_SCANF | a |  |  |  | Scanf(a) |  |
| OP\_PRINTF | a | b |  |  | Printf(a,b) |  |

### 8. 目标代码生成方案\*

生成目标代码的关键是将每条中间代码转换成对应的目标代码，为此对于每一种类别的中间代码，都需要为之安排对应的mips代码。

对于函数调用：call function，需要使用跳转指令跳到function对应的代码段，而在跳转之前，需要为被调用函数在栈上开辟空间，将$sp、$fp的值修改。另外在跳转之前，需要将传来的参数值赋给形参。而在每个函数的开头，需要保存上一个函数的运行栈地址，即$sp和$fp。

数组元素的访问以及赋值，需要先用数组地址+偏移量\*4计算出数组元素的地址，再进行内存操作。

### 9. 优化方案\*

1. 常数合并，对于两个操作数均为常数的中间代码，优化时可以直接算出结果
2. 常量替换，在生成中间代码时，就将所有的常量名替换成其常量值。
3. 对每一个基本块，建立DAG图,消除公共子表达式
4. 临时变量的重复利用(一个函数内存在大量的临时变量只使用过一次就再也没有用的情况)针对基本块的重新规划
5. 将临时变量的起始地址放到$k0寄存器中,$k1寄存器存放临时变量的最大编号
6. 全局寄存器分配（引用计数或着色算法）；

### 10. 出错处理

|  |  |  |
| --- | --- | --- |
| TYPE\_OF\_IDENTIFIER\_EXPECTED | 0 | expect the type of the declared identifier |
| UNEXPECTED\_IDENTIFIER | 1 | unexpected identifier appear |
| UNDEFINED\_IDENTIFIER | 2 | undefined identifier appear |
| UNEXPECTED\_VAR\_DECLARE | 3 | unexpected variable declare appear |
| UNEXPECTED\_SIGN | 4 | unexpected sign appear |
| IDENTIFIER\_ALREADY\_DEFINED | 5 | the identifier already defined |
| NONAGTIVE\_INT\_EXPECTED | 6 | expect an Non-negative |
| RIGHT\_PARENTHESES\_EXPECTED | 7 | expect a right parentheses |
| RIGHT\_BRACKET\_EXPECTED | 8 | expect a right bracket |
| RIGHT\_BRACE\_EXPECTED | 9 | expect a right brace |
| EQUAL\_EXPECTED | 10 | expect an equal symbol |
| NUM\_EXPECTED | 11 | expect a number |
| ALPHA\_EXPECTED | 12 | expect a char letter |
| SEMICOLON\_EXPECTED | 13 | expect a semicolon |
| LEFT\_BRACE\_EXPECTED | 14 | expect a left brace |
| LEFT\_PARENTHESES\_EXPECTED | 15 | expect a left parentheses |
| WHILE\_EXPECTED | 16 | expect a while |
| COLON\_EXPECTED | 17 | expect a colon |
| ASSIGN\_EXPECTED | 18 | expect an assignment |
| VARIABLE\_EXPECTED | 19 | expect a variable |
| INVALID\_RETURN | 20 | encounter an invalid return |
| WRONG\_FUNCTION\_CALL | 21 | encounter a wrong function call |
| WRONG\_RETURN\_TYPE | 22 | encounter a wrong return type |
| WRONG\_ARGUMENT\_LIST | 23 | encounter a wrong arguement list |
| MODIFY\_CONST\_VALUE | 24 | encounter a modification of constant |

可能的错误处理方案：

1. 不做处理
2. 跳读到下一个双引号位置
3. 跳读到下一个逗号位置
4. 跳读到下一个括号位置
5. 跳读到某一元素集合的位置
6. 跳读到下一语句列
7. 跳读到下一个分号
8. 程序直接退出

## 三．操作说明

### 1．运行环境

工程项目所属：VS2017

中间代码运行所需：Mars4.5

运行平台：windows10操作系统

### 2．操作步骤

1. 使用VS2017打开项目工程，编译工程，运行程序。
2. 在控制台输入编译文件的绝对路径，进行编译。
3. 源文件正确编译，生成mips32指令集支持的汇编中间代码，如果源文件。
4. 存在bug，不生成中间代码文件，在控制台输出进行报错。
5. 通过Mars4.5仿真器进行中间代码的模拟运行。

## 四．测试报告

### 1．测试程序及测试结果

正确测试1：

const int length = 10, width = +8;

const char firstLowerCase = 'a', firstUpperCase = 'A';

int globalInt, globalIntArray[44];

int globalChar, globalCharArray[44];

int none(int a, int b, int c){

const int testGlobal1 = 50000;

const int testGlobal2 = 50000;

const int testGlobal3 = 50000;

const int testGlobal4 = 5;

const int testGlobal5 = 5;

const int testGlobal6 = 5;

const int testGlobal7 = 5;

const int testGlobal8 = 5;

const int testGlobal9 = 5;

const char testddddd10 = 'd';

int d, e, f, p, q, w, x, y, z;

int g, h, k;

d = 0;

e = 1;

f = 2;

g = 3;

h = 4;

k = 5;

p = 6;

q = 7;

w = 8;

x = 9;

y = 10;

z = 11;

k = a + b + c + d + e + f + g + h + k + p + q + w + x + y + z + testGlobal1 + testGlobal2 + testGlobal3;

return (k);

}

void main(){

int n ,k;

n = 5;

k = 6;

k = n \* k;

n = none(100, 100, 100);

printf(k);

printf(n);

}

结果：

30150366

正确测试2：

const int length = 10, width = +8;

const char firstLowerCase = 'a', firstUpperCase = 'A';

int globalInt, globalIntArray[44];

int globalChar, globalCharArray[44];

int listInt[100];

void none(int a, int b, int c){

int i;

printf(a);

printf(" ", b);

printf(" ", c);

return;

printf(" ", c);

}

void main(){

int n ,k, i, c;

k = 0;

n = 50;

c = 20;

i = 10;

do{

listInt[i] = c;

c = c + 1;

i = i + 1;

} while(i!=30);

for(i = 0; i < 20; i = i + 1)

k = listInt[i] + k;

k = k / 20 ;

printf("%d", k);

printf(" ");

printf("%d", n);

printf(" ");

printf("%d", c);

printf(" ");

none(90, 100, 88);

}

结果：

%d12 %d50 %d40 90 100 88

正确测试3：

void main(){

int a, b, y;

a = 2;

b = -8;

y = - a + - 9;

printf(" %d", y);

y = - a + b;

printf(" %d", y);

y = - - 10 - - 5;

printf(" %d", y);

if( a == 2){

printf("ddddddd");

}

else{

printf("kkkkkkkk");

}

}

输出：

%d-11 %d-10 %d15ddddddd

正确测试4：

void main(){

printf(" \a");

printf(" \b");

printf(" \f");

printf(" \n");

printf(" \r");

printf(" \t");

printf(" \v");

printf(" \\");

printf(" \?");

printf(" \0");

printf(" \d88");

printf(" \x9f");

printf(" \a\b\f\n\r\t\v\\\?\0\d88\x9f");

}

输出：

\a \b \f \n \r \t \v \\ \? \0 \d88 \x9f \a\b\f\n\r\t\v\\\?\0\d88\x9f

正确测试5：

const int length = 10, width = +8;

const int firstLowerCase = 97, firstUpperCase = 64;

const int lastLowerCase = 123, lastUpperCase = 90;

int globalInt, globalIntArray[44], globalInt2, globalIntArray2[66], globalInt3;

int globalChar, globalCharArray[44], globalChar2, globalCharArray2[66], globalChar3;

int fib(int n, int k){

int i;

int res;

if(n <= k){

return (1);

}

else{

res = 0;

for(i = n - k; i < n; i = i + 1){

res = res + fib(i, k);

}

return (res);

}

}

void print(){

int i;

i = 32;

do{

if(i != 34){

printf(i);

}

i = i + 1;

} while(i < 127)

}

int toUpperCase(int x)

{

int c;

if (x >= firstLowerCase)

if (x <= lastLowerCase)

{

c = x - firstLowerCase + firstUpperCase;

return(c);

}

else return(x);

else return(x);

}

void changeGlobal(){

printf("original: ");

printf(globalInt);

globalInt = 16;

printf(" changed: ");

printf(globalInt);

}

void dealRequest(int operation){

const int x = 110;

int i;

int y;

printf("start ");

y = ('y');

if(operation == 0){

for(i = 1;i <= length;i = i + 1){

printf(" ", fib(i, --2));

}

} else if(operation == 1){

print();

} else if(operation == 2){

printf("%c", toUpperCase(x));

printf(toUpperCase(('x')));

} else if(operation == 3){

changeGlobal();

} else {

printf("nothing");

}

}

void main(){

int i;

int operation;

globalInt = 0;

scanf(operation);

for(i = 0; i < operation; i = i + 1){

dealRequest(i);

printf(" ");

}

}

输出：

start 1 1 2 3 5 8 13 21 34 55 start 32333536373839404142434445464748495051525354555657585960616263646566676869707172737475767778798081828384858687888990919293949596979899100101102103104105106107108109110111112113114115116117118119120121122123124125126 start %c7787 start original: 0 changed: 16 start nothing

错误测试1：

const int length = 10, width = +8;

const char firstLowerCase = 'a', firstUpperCase = 'A';

const char lastLowerCase = 'z', lastUpperCase = 'Z';

int globalInt, globalIntArray[44], globalInt2, globalIntArray2[66], globalInt3;

int globalChar, globalCharArray[44], globalChar2, globalCharArray2[66], globalChar3;

int fib(int n, int k){

int i;

int res;

if(n <= k){

return (1);

}

else{

res = 0;

for(i = n - k; i < n; i = i + 1){

res = res + fib(i, k);

}

return (res);

}

}

void print(){

char i;

i = 32;

do{

if(i != 34){

printf(i);

}

i = i + 1;

} while(i < 127)

}

char toUpperCase(char x)

{

char c;

if (x >= firstLowerCase)

if (x <= lastLowerCase)

{

c = x - firstLowerCase + firstUpperCase;

return(c);

}

else return(x);

else return(x);

}

void changeGlobal(){

printf("original: ");

printf(globalInt);

globalInt = 16;

printf(" changed: ");

printf(globalInt);

}

void dealRequest(int operation){

const char x = 'x';

int i;

char y;

printf("start ");

y = 'y';

if(operation == 0){

for(i = 1;i <= length;i = i + 1){

printf(" ", fib(i, --2));

}

} else if(operation == 1){

print();

} else if(operation == 2){

printf("%c", toUpperCase(x));

printf(toUpperCase('x'));

} else if(operation == 3){

changeGlobal();

} else {

printf("nothing");

}

}

void main(){

int i;

int operation;

globalInt = 0;

scanf(operation);

for(i = 0; i < operation; i = i + 1){

dealRequest(i);

printf(" ");

}

}

报错：

Error 31: Line: 24 " i = 32;" the type of assignment does't meet the requirements

cope with condition

quantity1 type:char

Error 30: Line: 26 " if(i != " the type of quantity does't meet the requirements

quantity2 type:int

Error 31: Line: 29 " i = i + 1;" the type of assignment does't meet the requirements

cope with condition

quantity1 type:char

Error 30: Line: 30 " } while(i < " the type of quantity does't meet the requirements

quantity2 type:int

cope with condition

quantity1 type:char

Error 30: Line: 36 " if (x >= " the type of quantity does't meet the requirements

quantity2 type:char

Error 30: Line: 36 " if (x >= firstLowerCase)" the type of quantity does't meet the requirements

cope with condition

quantity1 type:char

Error 30: Line: 37 " if (x <= " the type of quantity does't meet the requirements

quantity2 type:char

Error 30: Line: 37 " if (x <= lastLowerCase)" the type of quantity does't meet the requirements

Error 31: Line: 39 " c = x - firstLowerCase + firstUpperCase;" the type of assignment does't me

错误程序2：

void main(){

int a, b, y;

a = 2;

b = -8;

y = - a + - 9;

printf(" %d", y);

y = - a + b;

printf(" %d", y);

y = - - - 10 - - 5;

printf(" %d", y);

if( a == 2){

printf("ddddddd");

}

else{

printf("kkkkkkkk");

}

}

报错：

Error 11: Line: 8 " y = - - - " expect a number

错误测试3：

const int length = 10, width = +8;

const char firstLowerCase = 'a', firstUpperCase = 'A';

int globalInt, globalIntArray[44];

int globalChar, globalCharArray[44];

int none(int a, int b, int c){

const int testGlobal1 = 50000;

const int testGlobal2 = 50000;

const int testGlobal3 = 50000;

const int testGlobal4 = 5;

const int testGlobal5 = 5;

const int testGlobal6 = 5;

const int testGlobal7 = 5;

const int testGlobal8 = 5;

const int testGlobal9 = 5;

const char testddddd10 = 'd';

int d, e, f, p, q, w, x, y, z;

int g, h, k;

d = 0;

e = 1;

f = 2;

g = 3;

h = 4;

k = 5;

p = 6;

q = 7;

w = 8;

x = 9;

y = 10;

z = 11;

k = a + b + c + d + e + f + g + h + k + p + q + w + x + y + z + testGlobal1 + testGlobal2 + testGlobal3;

return (k);

}

void main(){

int n ,k;

n = 5;

k = 6;

k = n \* k;

n = none(100, 100, 100);

printf(k);

printf(n);

}

报错：

Error 14: Line: 31 " z " expect a semicolon

错误测试4：

void main(){

int a[5];

int b;

a[-1] = 8;

a[0] = 6;

a[6] = 4;

b = a[0];

b = a[5];

}

报错：

Error 32: Line: 5 " a[6] " index is out of array

Error 32: Line: 7 " b = a[5];" index is out of array

错误测试5：

const int length = 10, width = +8;

const char firstLowerCase = 'a', firstUpperCase = 'A';

int globalInt, globalIntArray[44];

int globalChar, globalCharArray[44];

int listInt[100];

void none(int a, int b, int c){

int i;

printf(a);

printf(" ", b);

printf(" ", c);

printf(" ", c);

return (a);

}

void main(){

int n ,k, i, c;

k = 0;

n = 50;

c = 20;

i = 10;

do{

listInt[i] = c;

c = c + 1;

i = i + 1;

} while(i!=30);

for(i = 0; i < 20; i = i + 1)

k = listInt[i] + k;

k = k / 20 ;

printf("%d", k);

printf(" ");

printf("%d", n);

printf(" ");

printf("%d", c);

printf(" ");

none(90, 100, 88);

}

报错：

Error 14: Line: 13 " return (a" expect a semicolon

### 2．测试结果分析

测试的语法成分并不全面，主要集中于后续添加的关于类型的要求的测试。同时测试了比较容易出现死循环的点。关于表达式的内容非法外层嵌套语句列的情况测试不够充分。上面的测试数量有限，但是均是为我带来困扰的典型错误程序。

## 五．总结感想

练习了c++面向对象编程。熟悉使用了c++11的各种新特性。

学习了编译器的基本原理，熟悉了编译优化的各种方法。对于我以后进行程序的编写，具有巨大的帮助。

编译课程是大三上学期的最重要的核心课程。但是这门课不均衡的课程安排，给同学的学习带来巨大的压力。大家前半个学期时间充裕。后半个学期各种课结课或者是期末考试的时候反而是编译原理的压力最大的时候。这个安排可能是为了课程内容进的安排。但是不得不承认为同学们带来了诸多不便。

同时这门课的很多要求也比较反人类。比如和优化冲突的计算顺序要求。明知道定义模糊，还要等同学自己挖掘的要求。这让我从此再也不想进行提问，我认为与甲方的沟通应该是灵活并且有预见性的。我也因此在后半个学期中，积极鼓动大家以后不要再在讨论区提问，在群内进行交流。

此外，理论老师的多次调课以及自身的讲课速度，也直接影响了课程进度，这导致学期3/4的时候，编译的理论课和课设基本已经变成完全独立的两门课，两门课的知识互相用不到。老师出于未知原因，或许是为了完成固定的进度，也没有对这个问题做出调整。

这门课给我带来的压力和烦躁，多于给我带来的编译知识和能力收获。可能更贴近于未来工作的状态吧。这是这门课给我的另一个收获。