《编译技术》课程

设计文档

学号： 张婉怡

姓名： 16182671

2019 年 1 月 3 日

## 一．需求说明

### 1．文法说明

＜加法运算符＞ ::= +｜-  
＜乘法运算符＞  ::= \*｜/  
＜关系运算符＞  ::=  <｜<=｜>｜>=｜!=｜==  
＜字母＞   ::= ＿｜a｜．．．｜z｜A｜．．．｜Z  
＜数字＞   ::= ０｜＜非零数字＞  
＜非零数字＞  ::= １｜．．．｜９  
＜字符＞    ::=  '＜加法运算符＞'｜'＜乘法运算符＞'｜'＜字母＞'｜'＜数字＞'  
＜字符串＞   ::=  "｛十进制编码为32,33,35-126的ASCII字符｝"  
＜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＞  
＜常量说明＞ ::=  const＜常量定义＞;{ const＜常量定义＞;}  
＜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＞}  
＜无符号整数＞  ::= ＜非零数字＞｛＜数字＞｝| 0  
＜整数＞        ::= ［＋｜－］＜无符号整数＞  
＜标识符＞    ::=  ＜字母＞｛＜字母＞｜＜数字＞｝  
＜声明头部＞   ::=  int＜标识符＞ |char＜标识符＞  
＜变量说明＞  ::= ＜变量定义＞;{＜变量定义＞;}  
＜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  
＜常量＞   ::=  ＜整数＞|＜字符＞  
＜类型标识符＞      ::=  int | char  
＜有返回值函数定义＞  ::=  ＜声明头部＞'('＜参数表＞')' '{'＜复合语句＞'}'  
＜无返回值函数定义＞  ::= void＜标识符＞'('＜参数表＞')''{'＜复合语句＞'}'  
＜复合语句＞   ::=  ［＜常量说明＞］［＜变量说明＞］＜语句列＞  
＜参数表＞    ::= ＜参数＞{,＜参数＞}| ＜空>  
＜参数＞    ::=  ＜类型标识符＞＜标识符＞  
＜主函数＞    ::= void main'('')''{'＜复合语句＞'}'  
＜表达式＞    ::= ［＋｜－］＜项＞{＜加法运算符＞＜项＞}   //[+|-]只作用于第一个<项>  
＜项＞     ::= ＜因子＞{＜乘法运算符＞＜因子＞}  
＜因子＞    ::= ＜标识符＞｜＜标识符＞'['＜表达式＞']'|'('＜表达式＞')'｜＜整数＞|＜字符＞｜＜有返回值函数调用语句＞           
＜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＞;|＜情况语句＞｜＜返回语句＞;  
＜赋值语句＞   ::=  ＜标识符＞＝＜表达式＞|＜标识符＞'['＜表达式＞']'=＜表达式＞  
＜条件语句＞  ::=  if '('＜条件＞')'＜语句＞  
＜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  
＜循环语句＞   ::=  while '('＜条件＞')'＜语句＞  
＜情况语句＞  ::=  switch '('＜表达式＞')' '{'＜情况表＞＜缺省＞ '}'  
＜情况表＞   ::=  ＜情况子语句＞{＜情况子语句＞}  
＜情况子语句＞  ::=  case＜常量＞：＜语句＞  
＜缺省＞   ::=  default : ＜语句＞|＜空＞  
＜有返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'  
＜无返回值函数调用语句＞ ::= ＜标识符＞'('＜值参数表＞')'  
＜值参数表＞   ::= ＜表达式＞{,＜表达式＞}｜＜空＞  
＜语句列＞   ::= ｛＜语句＞｝  
＜读语句＞    ::=  scanf '('＜标识符＞{,＜标识符＞}')'  
＜写语句＞    ::= printf '(' ＜字符串＞,＜表达式＞ ')'| printf '('＜字符串＞ ')'| printf '('＜表达式＞')'  
＜返回语句＞   ::=  return['('＜表达式＞')']

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char类型的变量或常量输出字符，其他表达式按整型输出

（4）情况语句中，switch后面的表达式和case后面的常量只允许出现int和char类型；每个情况子语句执行完毕后，不继续执行后面的情况子语句

（5）数组的下标从0开始

### 2．目标代码说明

生成的目标代码为mips汇编指令，可在MARS 4.5上运行。

### 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:

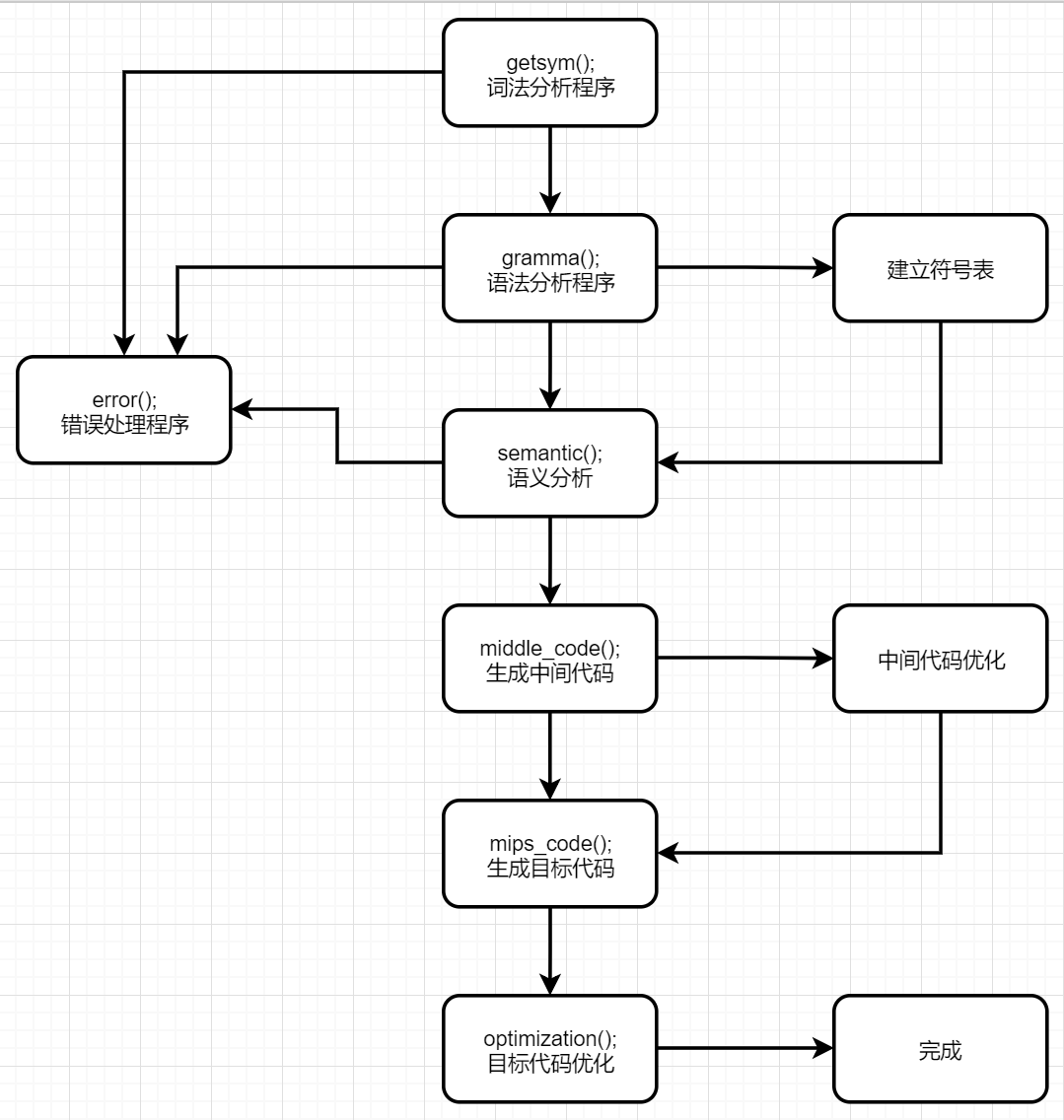
sw $x y($sp)

lw $x y($sp)

此时第二条指令可删除。

## 二．详细设计

### 1．程序结构



### 2．类/方法/函数功能

|  |  |
| --- | --- |
| 词法分析 | 功能 |
| int get\_symbol(FILE \*infile,symSY\* symbol) | 获得symbol |
| bool isLetter(char ch) | 判断是否为字母 |
| bool isDigit(char ch) | 判断是否为数字 |
| int isReserve(char \*token) | 判断是否为保留字 |
| bool isPa(char ch) | 判断是否为各种括号 |
| bool isOperator(char ch) | 判断是否为运算符 |
| bool isDemarcation(char ch) | 判断是否为分界符 ‘,’ ‘:’ ‘;’ |
| void lexical\_analysis(FILE \*infile) | 调用get\_symbol，打印结果 |
| 语法、语义分析 |  |
| void grammatical\_analysis() | 语法分析主函数 |
| void inserttab() | 建立符号表 |
| int searchtab() | 查找符号表 |
| bool isRelationalOp() | 判断是否为关系运算符 |
| bool isZero(char \*token) | 判断是否为0 |
| bool isUnsint() | 判断是否为无符号整数 |
| bool isINT() | 判断是否为整数 |
| bool isCharacter() | 判断是否为字符 |
| void Constant\_definition(int global) | 常量定义 |
| bool isStatement() | 判断是否为语句开始符 |
| void Constant\_description(int global) | 常量说明 |
| void Parameter\_table() | 参数表 |
| void value\_para(string fun\_name, int id\_index) | 值参数表 |
| void Funcall\_statement(string a, int para\_num, int id\_index) | 函数调用语句 |
| void assign\_statement(string id, int sym) | 赋值语句 |
| string factor(string minu)/ string factor\_print(int &achar) | 因子 |
| string term(string a)/ string term\_print(int &achar) | 项 |
| string Expression()/string Expression\_print(int &achar) | 表达式 |
| void condition() | 条件 |
| void case\_table(string e, string label) | 情况表 |
| void statement() | 语句 |
| void Statement\_column() | 语句列 |
| void variable\_description(int falg) | 变量说明 |
| void Compound\_statement() | 复合语句 |
| void function(string s) | 函数 |
| void Variable\_function() | 识别函数前的变量说明或函数 |
| bool check\_id() | 检查标识符是否已被定义 |
| void check\_sym(int achar, int sym) | 检查类型是否匹配 |
| bool check\_function(char \*token) | 检查函数是否在函数表中 |
| int check\_id\_exist(char \*token) | 检查标识符是否在符号表中 |
| 生成中间代码 |  |
| void genmidcode(char op, string A, string B, string C) | 生成中间代码 |
| string gentmp() | 生成临时寄存器名 |
| string genlabel() | 生成标签 |
| string sym\_to\_str(symSY symbol) | 将symbol改为string类型的int或char |
| void print\_midcode() | 打印中间代码 |
| void mips\_globl() | 全局变量换名 |
| 生成目标代码 |  |
| void gen\_mips() | 生成目标代码 |
| void mips\_data() | 生成.data段代码 |
| void mips\_globl() | 生成.data全局变量部分 |
| void mips\_str() | 生成.data字符串部分 |
| void instack(vector<run\_stack>\* func\_runstack, string name, int cap) | 检查变量是否在函数运行栈中，不在栈中则入栈 |
| string getaddr(string name, string index2 = "-1") | 计算变量相对sp的偏移 |
| void mips\_fun(int func\_begin, int func\_end) | 生成每个函数的mips代码 |
| 错误处理 |  |
| void error(error\_num erroNUM) | 处理错误信息 |

### 3．调用依赖关系

语法分析程序调用词法分析程序，再进行语义分析并生成中间代码。

词法分析，语法分析和语义分析调用错误处理生成错误信息。

目标代码生成程序根据中间代码生成目标代码。

### 4．符号表管理方案

|  |  |  |
| --- | --- | --- |
| 类型 | 名称 | 备注 |
| char | name | 标识符名 |
| SYMtyp | symtyp | 类型（数组、函数、变量等） |
| symSY | sym | 类型（词法分析类型） |
| int | level | 层数（全局0，每个函数+1） |
| int | length | 数组长度、函数参数个数 |
| string | value | 常量值 |

### 5．存储分配方案

|  |
| --- |
| 栈顶 |
| 数据区 |
| 返回值 |
| 返回地址 |
| 上个函数的栈顶 |

|  |
| --- |
| 函数运行栈 |
| 函数运行栈 |
| 全局变量 |

### 6. 解释执行程序\*

无解释执行程序。

### 7. 四元式设计\*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 设计 | op | A | B | C | 中间代码 | 备注 |
| 常量定义 | c | name | value | type | const int min=0 | 全局常量进行换名 \*gl1 |
| 变量定义 | v | name | length | type | var int A[10] | 全局变量换名 \*gl2 |
| v | name | ---- | type | var char a |  |
| 函数 | f | name | ---- | type | int func1 | 函数换名\*func0 |
| 参数 | p | name | ---- | type | para int x |  |
| 赋值 | a | $tn | Array\_name | index | $tn=A[0] | 数组元素赋值给临时变量 |
| 函数调用 | b | call | Fun\_name | para\_num | call func 3 | C为参数个数 |
| b | $tn | ret |  | return $tn | return临时变量 |
| 传入参数 | d | $tn | Fun\_name | index | push $tn |  |
| 赋值 | e | $tn | var\_name | ---- | $tn=var | 将变量值赋给临时变量 |
| 赋值 | g | $tn | number | ---- | $tn=1 | 将数值赋给临时变量 |
| 关系运算 | h | e1 | R\_op | e2 | e1==e2 |  |
| h | e | != | 0 | e!=0 | 条件为单一表达式 |
| 跳转指令 | j | BZ/GOTO | lable | ---- | BZ lable0 |  |
| 赋值 | k | var\_name | ---- | $tn | a=$tn |  |
| k | Array\_name | index | $tn | A[1]=$tn |  |
| 标签 | l | label0 | : | ---- | Label0: |  |
| 写语句 | w | print | str/e | 0/1 | print \*str0/a | 0为数字 1为字符 |
| 读语句 | s | scan | id\_name | type | scan int a |  |
| 运算 | op | $tn1 | $tn2 | $tn1 | $tn1=$tn1+$tn2 |  |

### 8. 目标代码生成方案\*

将中间代码转逐句转换成mips代码。

对于传递的参数，由于参数可能是又调用了其他函数的结果，需要注意参数在栈空间中的位置和调用关系。

在将每个函数翻译为目标代码时，由于在中间代码中临时变量的序号是顺序编号的，所以可以计算出每个函数需要的栈空间大小，在函数的标签后可直接将$sp减去相应大小的值开辟栈空间，避免使用$fp.

数组元素赋值和访问则需要使用地址+偏移量\*4.

### 9. 优化方案\*

预计进行简单的窥孔优化。

### 10. 出错处理

|  |  |  |
| --- | --- | --- |
| 错误 | 编号 | 含义 |
| ZEROSTARTNUM\_ERRO | 0 | 0开头数字 |
| ILIGALCHAR\_ERRO | 1 | 非法字符 |
| SINGLEQ\_ERRO | 2 | 缺少单引号 |
| DOUBLEQ\_ERRO | 3 | 缺少双引号 |
| LPAR\_ERRO | 4 | ( |
| RPAR\_ERRO | 5 | ) |
| LBRACKET\_ERROR | 6 | [ |
| RBRACKET\_ERROR | 7 | ] |
| LBRACE\_ERROR | 8 | { |
| RBRACE\_ERROR | 9 | } |
| CONST\_TYPE\_ERROR | 10 | 常量赋值类型错误 |
| CONST\_ID\_ERROR | 11 | 常量名错误 |
| ASSIGN\_SY\_ERROR | 12 | 赋值类型错误 |
| CONST\_ASSIGNTY\_ERROR | 13 | 常量赋值类型错误 |
| SEMI\_ERROR | 14 | 缺少分号 |
| ID\_ERROR | 15 | 命名错误 |
| ID\_DECLARED | 16 | 标识符重复定义 |
| ID\_UNDECLARED | 17 | 标识符未定义 |
| FUN\_UNDECLARE | 18 | 函数未定义 |
| SYM\_ERROR | 19 | 类型错误 |
| VPARA\_NUM\_ERROR | 20 | 值参数参数个数错误 |
| FUNCFACTOR\_SYM\_ERROR | 21 | 函数调用语句作为factor时，返回值为void |
| ILIGAL\_FACTOR\_ERRO | 22 | 不是合法的factor |

错误处理：调用error(error\_num erroNUM)函数打印错误类型和行数。

## 三．操作说明

### 1．运行环境

运行环境：Windows 10

开发环境：Visual Studio 2017

### 2．操作步骤

在控制台输入文件源路径，将运行结果中的目标代码文件中的mips指令在Mars中运行。

## 四．测试报告

### 1．测试程序及测试结果

(1) 测试程序一：

int Factorial(int n) {

if (n <= 1) {

return (1);

}

if (n > 1) {

return (n \* Factorial(n - 1));

}

}

void main(){

int n;

n=10;

printf("factorial result:", Factorial(n));

}

输出结果：3628800

(2) 测试程序二:

void fun2(int k1, int k2) {

char op;

int S;

printf("please input op:");

scanf(op);

switch (op)

{

case '+' : S = k1 + k2;

case '-' : S = k1 - k2;

case '/' : S = k1 / k2;

case '\*' : S = k1 \* k2;

default: S=-0;

}

printf("The result is", S);

}

void main(){

int k1, k2;

scanf(k1, k2);

fun2(k1, k2);

}

运行结果：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <<12 | 12 | 12 | -12 | 12 |
| <<4 | 4 | -4 | 4 | 4 |
| >>pleaseinput op: | pleaseinput op: | pleaseinput op: | pleaseinput op: | pleaseinput op: |
| <<+ | - | \* | / | null |
| >>The result is | The result is | The result is | The result is | The result is |
| >>16 | 8 | -48 | -3 | 0 |

(3) 测试程序三:

const int Max = 20, Min=0;

const char CHAR\_A='A', char\_b='b';

char String[3];

int Num[2], len;

void fun3(){

int k;

k=0;

String[0]='a';

String[1]='b';

String[2]='c';

Num[0]='a'+1;

Num[1]='b'+1;

while(k<3){

printf(String[k]);

k=k+1;

if (k!=3){

printf(Num[k]);

}

}

len=k;

printf(k);

printf(Max+CHAR\_A);

printf(Min-char\_b);

printf(Max);

}

void main(){

fun3();

}

输出结果：

a 99 b 0 c 3 85 -98 20

(4) 测试程序四:

char fun4(int n, char flag){

const int Max=20;

int N[30], i;

if (n<Max){

i=0;

while(i<n){

if (i>3){

N[i] = N[i - 1] \* (N[i - 2] - 2\*N[0]);

}

if (i<=3){

N[i]=i\*(Max-5)+1;

}

i=i+1;

}

}

printf(N[i-2]);

if (flag=='a'){

return('a');

}

if (flag != 'a'){

return('b');

}

}

void main(){

const int m=15;

const char flag='a';

printf(fun4(m,flag));

printf(fun4(m-5,'b'));

}

输出结果：

2032726016 a 2123303040 b

(5) 测试程序五:

void main()

{

int a[10];

int i, j, temp;

printf("Please input ten numbers:");

i=0;

while (i<10){

scanf(temp);

a[i]=temp;

i=i+1;

}

i=0;

j=0;

while (i<9){

j=0;

while(j<9-i){

if (a[j] > a[j+1])

{

temp = a[j];

a[j] = a[j+1];

a[j+1] = temp;

}

j=j+1;

}

i=i+1;

}

i=0;

while (i<10){

printf(a[i]);

i=i+1;

}

}

输出结果：-23 -1 1 6 12 14 23 45 67 89

(6) 测试程序六:

const int a=1

void main(){

int i

int j

i=2

j=3

i=j+i

printf(i)

}

输出结果：

SEMI\_ERROR at line 1

SEMI\_ERROR at line 3

SEMI\_ERROR at line 4

SEMI\_ERROR at line 5

SEMI\_ERROR at line 6

SEMI\_ERROR at line 7

SEMI\_ERROR at line 8

(7) 测试程序七：

void fun1(){

int a;

a=1;

}

void main(){

fun(1+2);

a=a+1;

}

输出结果：

ID\_UNDECLARED at line 7

VPARA\_NUM\_ERROR at line 7

ID\_UNDECLARED at line 8

ID\_UNDECLARED at line 8

(8) 测试程序八：

int fun(int n){

n=n+2;

}

char fun1(){

return('a');

}

void func2(){

int i;

i=0;

}

void main(){

char a;

int n;

n=1;

a=fun(n);

a=fun1()+1;

}

输出结果：

SYM\_ERROR at line 15

SYM\_ERROR at line 16

(9) 测试程序九：

const int A=0;

int fun(char a){

return (1);

}

void main(){

int A;

int A[10];

char S[11];

S[1]='a';

S[0]=fun(A);

}

输出结果：

Identifier has been declared at line 7

SYM\_ERROR at line 10

SYM\_ERROR at line 10

(10) 测试程序十：

int fun(){

printf("func1");

void main(){

int S[10];

fun();

输出结果：

RBRACE\_ERROR at line 3

RBRACE\_ERROR at line 6

### 2．测试结果分析

1. 测试程序一（正确）

有返回值函数，阶乘计算，测试递归调用函数，printf中先打印字符串再打印表达式结果，此处表达式为函数调用结果，return语句中为表达式，此处表达式为乘法，其中的一个factor为函数调用语句，还测试了if语句，条件为<=和>。

2. 测试程序二（正确）

无返回值函数，测试switch语句，输入语句的多变量和单变量不同类型的输入，以及简单的数字运算，输出单独的字符串，局部变量声明。

3. 测试程序三（正确）

测试while和if语句；测试printf语句打印表达式值时的类型是否正确；测试不同类型的全局常量和全局变量，其中全局变量有数组和单个变量的形式，一行可有多个或单个常量变量。

4. 测试程序四（正确）

测试较为复杂的表达式；测试局部常量和局部变量；测试返回类型为char的函数，测试if和while临近时标签是否存在问题。

5. 测试程序五（正确）

冒泡排序，主要测试对数组元素进行赋值和读取是否正确；测试while嵌套是否正确。

6. 测试程序六（错误）

测试缺少分号错误是否正常

7. 测试程序七（错误）

测试ID未声明错误是否正常，在函数未声明的情况下调用函数，会报未定义和参数个数错误。

8. 测试程序八（错误）

测试类型错误是否正常，给char类型变量赋值为int，函数调用语句作为factor时返回类型为void.

9. 测试程序九（错误）

测试重复定义标识符，类型错误，传入参数类型错误和赋值类型错误。

10. 测试程序十（错误）

测试缺少大括号报错。

## 五．总结感想

历时两个多月时间，简陋的编译器终于算是能运行一些正确的小程序了，虽然它生成的目标代码还有极大的优化空间，但由于时间关系，很遗憾可能没有机会在最后提交期限之前完成了。但这依旧是我触编程以来完成的行数最多，最复杂的工程了，过程中还是有不少收获。

最开始的理论课听得还算认真，但到后来难度增加，又或者是感到枯燥以及自己长久以来的怠惰导致大概从语法分析那部分开始就没有学扎实，平时的小编程作业通过参考课本尚可一写，到了真的写自己的小编译器时会碰到许多概念上的问题，而且C0与课本上主要讲的Pascal还是有一些不同，在每个星期两次的ddl追赶下还是有些吃力，特别是在目标代码生成的那个星期，突如其来的病倒更是打乱了节奏。虽然老师一再强调要尽量提前完成，拖延的习惯还是没能克服，这需要吸取教训。

另外，整体代码的可维护性较差，模块之间耦合性太高，对于这种还不算大的规模来说就已经相当致命，这一点仍需要更多的改进。代码结构的设计也并不完善，不同的分支内用大量的重复代码，以及多层if-else嵌套也使得代码阅读困难，难以修改。抽取出一些常用的小函数也会有极大的帮助，以及对于一些简单函数运用也可以大大减少工作量。这些煎熬的经历让我更清晰地认识到在动手写代码前进行良好的设计是十分必要和重要的。大量的debug工作有些枯燥，但在找到bug时也是欣喜的，有一些是设计错误，但更多的是一些完全由于自己粗心而产生的，希望自己以后做事能更细致一些。

最后感谢课程组的各位老师和助教一学期以来对我们的悉心教导、答疑解惑。也感谢一路走来一起相互鼓励相互帮助的同学们。